uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Rami Abielmona

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Wireless Network of Intelligent Sensor Agents for the Mapping of Multiple Environmental Parameters

TITRE DE LA THÈSE / TITLE OF THESIS

E.M. Petriu

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

V. Groza

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Victor Aitken

N.D. Georganas

Mel Siegel

A. El Saddik

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A Wireless Network of Intelligent Sensor Agents for the Mapping of Multiple Environmental Parameters

by

Rami Abielmona

M.A.Sc., University of Ottawa, 2002

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies of the

University of Ottawa in partial fulfillment

of the requirements for the degree of

Doctorate of Philosophy in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering

2007

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-46527-1
Our file Notre référence
ISBN: 978-0-494-46527-1

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

© Rami Abielmona, Ottawa, Canada, 2007

Dedication

To Mom & Dad. May I repay you for what you have given to me for free.

To Samer. May you remain my role model forever.

To Mimo. May life never pull us apart.

Abielmona, Rami (Ph.D., Electrical Engineering)

A Wireless Network of Intelligent Sensor Agents for the Mapping of Multiple Environmental Parameters

Thesis directed by Dr. Emil Petriu and Dr. Voicu Groza

The active investigation of complex environmental parameters involves many computationally intensive operations, such as data collection, aggregation, dissemination and processing. The adoption of multi-agent systems appears as a very promising approach in the development of a new generation of intelligent autonomous robotic agents for environment monitoring. This thesis discusses a multi-agent system whose global goal is the minimization of entropy of an environment, based on a novel tree-in-motion-mapping method. The proposed method combines simplicity and speed of computation, along with low storage and communications requirements, ideally fitting a system composed of a finite number of mobile robotic agents, each possessing limited sensing, processing and communicating operational entities. The method is also extensible to include multiple factors for the determination of the environmental entropy, hence allowing measurands such as electromagnetic, thermal and optical energies to be aggregated in the entropy function. Simulation results demonstrate the efficiency of the method, compare the centralized and decentralized approaches, and present the savings, both in processing and communication times, when compared to existing methods. Experimental results are presented and analyzed to validate the method in a testbed environment. Finally, a formal specification of a generic multi-agent system intended for environment monitoring is presented, while a virtual prototyping tool intended for verifying diverse scenarios in simulation, before proceeding to implementation, is outlined.

Acknowledgements

I would like to thank my thesis supervisor Dr. Emil Petriu for his ongoing support, and great assistance with this thesis. He has provided me with a very balanced and grounded environment. I have learned so much from him, be it from an engineering perspective or a personal one, he has been my guiding light throughout this venture. I would also like to thank my second co-supervisor, Dr. Voicu Groza, for his unwavering support and help.

Heartfelt appreciations go out to Hugh Pollitt-Smith of CMC for his prompt and explanatory responses concerning the design environments, Dr. Stephen Brown of Altera Corp. for his refreshing support of our Faculty through donations and discounts on Altera products, George DiNardo and Gail DiCintio of Larus Technologies for their support through radio modems, and Alan Stewart, Roger Montcalm, Michel Racine, Keith White, Samer Sader and Marc Fortier, the SITE technical staff, for their unblemished administrative support. Great job guys!

Special thanks go out to all the graduate students of the SMRLab and the GEMS Lab, there are too many of you to name here, but you know who you are, and I thank you for all the wonderful conversations and discussions that we have add all these years.

Last, but certainly not least, I would like to express my deepest and warmest thank you to my parents and my brother Samer, without which none of this would be possible. They have been there all my life supporting and inspiring me to reach for the stars. I owe everything to their unparalleled support and exemplary role models. And to Maya: thank you for providing me with a reason to live, and a life of reason. Your determination, persistence and ardor are awe-inspiring! Thank you for being there throughout all the ups and downs.

Contents

Chapter

Tables

Table

Figures

Figure

List of Acronyms

CCS	Calculus of Communicating Systems
CRC	Cyclic Redundancy Check
CSP	Communicating Sequential Processes
CSP-OZ	CSP-Object-Z
CV	Certainty Value
DAI	Distributed Artificial Intelligence
DISS	Distributed Intelligent Sensor System
DVMT	Distributed Vehicle Monitoring Testbed
EAP	Ethernet Access Point
EKF	Extended Kalman Filter
FIPA	Foundation of Intelligent Physical Agents
FoQ	Forest of Quadtrees
FoM	Figure of Merit
HIMM	Histogram-In-Motion-Mapping
InteRCom	Intelligent Robotic Communication
ISA	Intelligent Sensor Agent
J2EE	Java 2 Enterprise Edition
KIF	Knowledge Interchange Format
LOTOS	Language of Temporal Ordering Specifications
MAPS	Multi-Agent Platform Specifications
MAPSE	Multi-Agent Platform Simulation Environment
MAS	Multi-Agent System
MaSE	Multiagent Systems Engineering
MSF	Multi-Sensor Fusion
OGF	Occupancy Grid Framework
POSE	Position and Orientation
RTOS	Real-Time Operating System
SLAM	Simultaneous Localization and Mapping

Acronyms Continued

SMART	Structured and Modular Agents and Relationship Types
TCOZ	Timed Communicating Object-Z
TIMM	Tree-In-Motion-Mapping
UML	Unified Modelling Language
VE	Virtual Environment
VRME	Virtualized Reality Model of the Environment
VRML	Virtual Reality Modeling Language
VDM	Vienna Development Method
WMR	Wheeled Mobile Robot

Chapter 1

Overview and Synopsis

"There is one thing stronger than all the armies of the world, and that is an idea whose time has come"

(Victor Hugo)

A multi-agent system is one where individual agents coordinate their activities and co-operate with each other, to avoid duplication of effort and to exploit other agents' capabilities. Multi-agent systems have been proposed and implemented for use in distributed sensing and information retrieval and management. This chapter introduces the field of multi-agent systems, before presenting the entropy-minimization-based model that will be examined in detail within the context of environment monitoring, and concluding with the scope and organization of this thesis.

1.1 Introduction

Environment monitoring is a complex task of great importance in many areas, such as the natural living environment, polluted or hazardous environments or remote, difficult-to-reach environments [1], [2]. The task falls under the general umbrella of multi-agents for distributed sensing, that consists of a system composed of a network of spatially distributed sensors. The global goal is to monitor the environment for objects and events that are within the range of the deployed sensor agents. If cooperation exists between the sensors, then as soon as an object or event is detected, the occurrence is propagated throughout the network to better predict the

dynamics of that object (e.g., its movement) or event (e.g., its periodicity). In general, environment sensing has become a major field of research that has spawned numerous applications in today's society. Witness the *sensor network (SNET)* revolution [3], [4], [5] and [6] unfold before our eyes and we notice that our society is becoming dependent on pervasive computing to enhance our perceptions, our judgments and eventually our everyday lives. Foundational work in this area has been provided by Lesser and Erman [7] with their *Distributed Vehicle Monitoring Testbed (DVMT)*; however, recently, more attention has been paid to the application of *multi-agent systems (MASses)* due to its prominence in current world affairs (i.e. homeland security and the threat of terrorism) [8], [9].

Monitoring uncertain natural environments is a task that is very much dependent on each application. The complexity of this task is a function of the nature of the parameters and the nature of the environment to be monitored. The type of sensors, their number and their bandwidth, the difficulties encountered while deploying the sensors in the field, the reliability of the sensory data and of the data communication network are all requirements that have to be considered when designing the intelligent sensor agents and the distributed sensor network architecture. It is essential that redundancy is built into the system such that loss of a sensor or communication system will not impact the efficacy of the monitoring network. There are numerous environment properties to be considered [10]:

Accessible vs. inaccessible If accessible, we can obtain complete and accurate information about the environment;

Deterministic vs. non-deterministic If deterministic, we can guarantee that a single action has a known effect on the environment;

Episodic vs. non-episodic If episodic, we can link performance of the robot to a discrete set of episodes occurring in the environment;

Static vs. dynamic If static, we can assume that the environment does not change, unless it is from an action of a robot or an agent; and

Discrete vs. continuous If discrete, we can represent the environment with a fixed and finite number of actions and perceptions.

The most complex type of environment is the real world because it is inaccessible, non-deterministic, non-episodic, dynamic and continuous. Thus, monitoring the real world, in practical terms, is a *game* with limited resources. There is a limited number of agents, that have limited operational parameters, communicating via a limited quality-of-service communications network. The agents are capable of selective environment perception focusing on parameters that are important for the specific task and avoid wasting resources on processing irrelevant data. Different sensor planning strategies are used for the placement of the fixed and mobile sensor agents in such a way as to get optimum performance during specific sensing tasks and for the real-time selection of sensing operations to minimize the observed system entropy [11], [12].

This thesis describes a novel method for multi-parameter environment mapping by intelligent sensor agent networks [13]. This method, called **tree-in-motion mapping (TIMM)**, combines environment monitoring through entropy reduction with a quadtree-based data structure to present a multi-resolution and multi-dimensional view of the environment.

1.2 Problem Definition

The problem is defined as the reduction of the entropy of the environment, using a limited set of autonomous robotic **intelligent sensor agents (ISAs)**. Each ISA has the ability to measure only a small subset of environment parameters of interest. The measured range of each sensor only covers a limited area of the environment in the neighbourhood of each ISA and its extent is a function of the actual environmental condition in that area. As individual ISAs only offer local and limited information about the environment, multiple and diverse ISAs will be needed to cover the large area and multi-parameter natural environments that are monitored in real-life situations.

To reduce the entropy in the environment, we have to develop a cost-effective strategy

to reduce the cost of the measurement act of monitoring the environment. This involves the resource management of multiple and heterogeneous ISA sensing capabilities. If one accepts that the ISAs possess autonomy, then one has to allow them to be intelligent enough to carry their specific task of reducing the entropy, within the constraint of cost-effectiveness. A virtual prototyping tool was developed [14] to study various strategies within an environment allowing us to find out the best strategy for placing a limited set of sensors within the environment, to maximize the cost-effectiveness of the sensing tasks.

In this thesis, we utilize the term entropy to indicate the level of our knowledge of the environment. Thus, instead of organizing the environment by reducing its entropy, we are actually reducing our ignorance of its organization by reducing the entropy of our knowledge of the environment, through the sensory data acquisitions. This is not to be associated with the typical reduction of entropy, found in thermodynamics, used to indicate the disorder in the environment, but rather, the term is to be associated with the measure of the amount of acquired information, from information theory, used to indicate the missing information about the environment.

1.2.1 The Distributed Intelligent Sensor System

A *Distributed intelligent sensor system (DISS)* was developed to integrate robotic ISAs, the wireless communications network, as well as the **virtualized reality model of the environment (VRME)**. ISAs could either be mobile (mISAs) or stationary (sISAs). The *virtualized reality* concept introduced by [15] represents an extension of the typically computer-generated, but synthetic virtual reality concept. VRME integrates natural environment information captured by the ISAs in the field. As the thesis is dealing with sensor agents for natural environment monitoring, we will use the term *virtual/virtualized reality model of the environment* instead of the usual term *virtual/virtualized reality environment.* This will reduce the confusion that may be present by the different meanings of the term *environment* in this case.

DISS' design requirements are broken up into five major categories (Figure 1.1):

(1) **Distributed agent-based resource management**: This framework provides a flexible, extensible and open mechanism allowing for agent interoperability. Work has already been completed on such a framework [16] and [17], that assumes a limited amount of computational and communication resources on each ISA;

(2) **Local intelligence**: A reactive-behavior paradigm [18] has been investigated and implemented in [19]. The agent reacts to sensory inputs from its environment, by actuating upon the latter, without reasoning about the performed actions. A novel retroactive agent architecture has also been devised and tested in experimental scenarios;

(3) **Network intelligence**: Since all our ISAs are instinctive information seeking agents, some of them will cooperate together towards the achievement of the overall goal, that is to maximize the information acquired from the environment about objects or events;

(4) **Wireless communication**: A network protocol, referred to as Intelligent Robotic Communication (InteRCom), has been devised to allow for the efficient utilization of the available wireless channels; and

(5) **Multi-sensor fusion**: This mechanism is used to integrate heterogeneous sensory data into a composite and coherent model of the environment. As human beings are valuable partners to the ISAs to the degree that their capabilities complement those of the ISAs, the VRME should allow the human operators to get through their own sense of vision, hearing, smell, taste and touch. The latter can be accomplished by forcing a direct feeling of the contact phenomena that a specific ISA experiences while in the field. However, we currently only support sensory feedback to a workstation VRME. In the future, head-mounted displays and haptic feedback gloves will allow the monitors to incrementally immerse themselves into the VRME.

1.2.2 Research Motivations

The field of natural and man-made environment sensing and monitoring has been gaining more prominence in real-world applications, since it has recently become very costly to invest designer efforts, time and money into a single entity for the resolution of the problem at hand. A better strategy is to design and realize numerous smaller scale entities, that are capable of coordinating their actions, to more efficiently target the application. Hence, the research motivations of this project are three fold:

- Develop a new generation of autonomous wireless robotic *Intelligent Sensor Agents (ISAs)* for complex environment monitoring;
- Fuse collected sensor data into a world model that is remotely available to human monitors; and
- Represent the model in an interactive *Virtualized Reality Model of the Environment (VRME).*

1.2.3 Technical Contributions

The proposed technique, framework, simulator, virtual environment and system solution all target environment monitoring by multi-agent systems; however, they can be extended to numerous other application domains and areas, as we shall see later. The following is a list of the technical contributions of this work in the field of distributed artificial intelligence:

- **Novel monitoring method** This real-time technique has been developed for the monitoring of an environment using a set of mobile agents that possess a limited amount of computational power and communications bandwidth;
- **Scaleable MAS formalism** The developed specifications allow for the systematic realization of the multi- agent system and extensions of the latter;

- **Virtual prototyping tool** The software simulator allows us to evaluate various strategies for the maximization of cost-effectiveness, and minimization of delay and energy;
- **Virtualized reality model of the environment** The VRME is constructed from measurements provided by the ISAs deployed in the natural environment that they are investigating; and
- **Novel agent architecture** The retroactive agent architecture is an innovative one that paves the way for the development of truly intelligent machines, ones based on the new [20] and old [21] concepts of neuroscience.

1.3 Summary

This introductory chapter has briefly overviewed the field of distributed multi-agent sensor systems. The chapter has examined and discussed the five major design requirements of such a system: distributed agent-based resource management, local intelligence, network intelligence, wireless communication and multi-sensor fusion; before going on to summarize the remaining thesis chapters.

1.4 Thesis Organization

Chapter 1, this introduction to the thesis, has briefly outlined the existing and growing problem of information overload, retrieval and management. It has presented the problem in the context of entropy minmization and has discussed and categorized the major design requirements of a system intended for distributed sensing applications. Finally, the remainder of the thesis is summarized.

Chapter 2 presents the related work that was previously undertaken in the field of multi-agent systems for distributed sensing. It also discusses the advantages and disadvantages associated with each one, as well as presents seven classifying themes for multi-agent systems. Finally, it outlines the proposed solution's similarities and differences to previous methods,

classified by those seven themes.

Chapter 3 discusses three data models: an agent motion model, a range sensor model and a multi-sensor fusion model. Both centralized and decentralized fusion models are introduced and discussed. The chapter also presents the environment mapping method, which combines the goal of reducing the sensed environment entropy, with a quadtree-based data structure representation of the environment. The combination provides a real-time embedded environment monitoring solution, capable of displaying multi-resolution and multi-dimension views of the environment. The chapter is concluded with the presentation of an integration technique based on maximizing the environment's information content, when integrating multiple sensor modalities together.

Chapter 4 presents the formal specifications of the novel agent architecture and the multi-agent system, as well as the latter's associated simulator. The retroactive agent architecture attempts to resolve many of the drawbacks associated with the proactive and reactive agent architectures, including those of calculative rationality and short-term views. To decide on a suitable specifications, numerous specification languages were researched, with CSP-OZ fulfilling most of the requirements set out at the outset of this research.

Chapter 5 presents the simulation results using centralized and decentralized (i.e. distributed) multi-sensor fusion approaches, with comparisons between them outlining the better strategies of monitoring the environment, involving a reduced set of agents, while distributing cooperative and competitive sensor relationships. Comparisons to the currently prevalent techniques are also presented. Simulations demonstrating the accuracy and noise immunity of the proposed method, as well as its sensor correlation feature (through the maximization of the mutual information metric), conclude the chapter.

Chapter 6 presents the experimental results, where an actual prototyping environment is explored by a set of physical robotic sensor agents. The realized TIMM method is demonstrated by introducing a society of agents, to explore and monitore the environment through their coordinated actions, and to feed back into a synthesized model of the real-world measurement data,

in real-time. The four metrics, used in simulation, are explored again in this chapter, with a focus on the realized TIMM method.

Chapter 7 examines the virtualized reality model of the environment (VRME), a user interface for the multi-agent system. This model allows for a real-time presentation of the real-world environment and for a better decision-making process for the remote monitors and operators. It shows how to synthesize a model from the distributed sensor measurements, and to pesent the information, extracted from those sensor data, in a consistent fashion.

Chapter 8 concludes this thesis, summarizes contributions and achievements, and then outlines areas for future work.

References supporting statements in this thesis are then listed.

Appendices provide implementation details for both the VRME, the open-core sensor and actuator modules developed in this work, as well as the robotic agents' hardware and software modules.

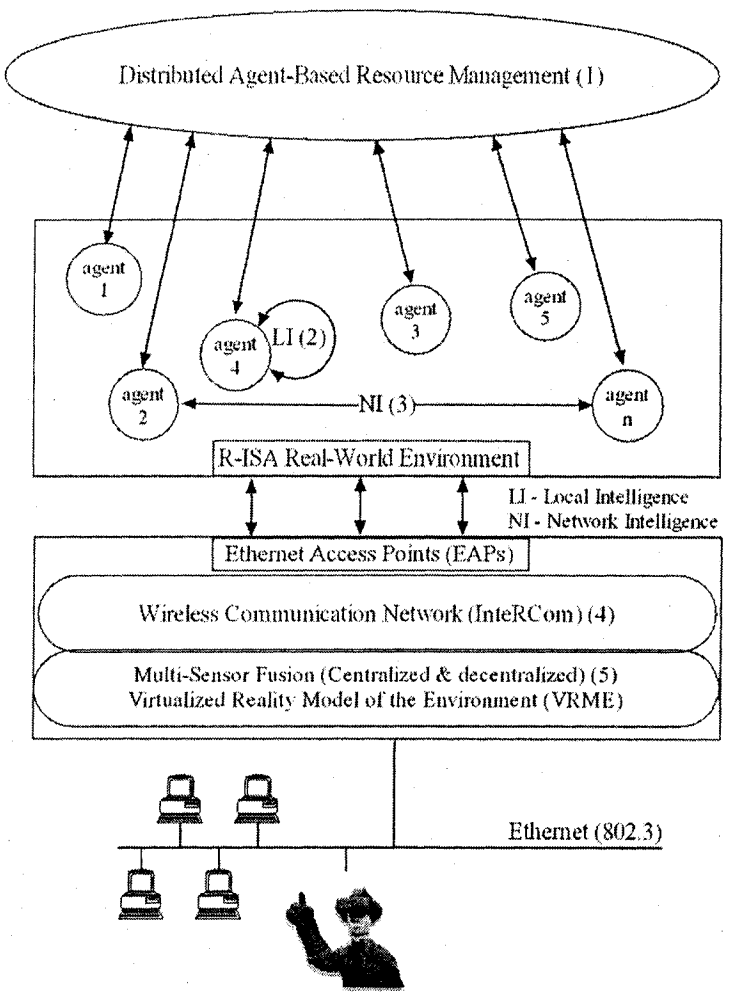


Figure 1.1: DISS flow diagram

Chapter 2

State of the Art

"How can they say my life isn't a success? Have I not for more than sixty years got enough to eat and escaped being eaten?"

(Logan Pearsall Smith)

Multi-agent systems can be applied to many areas, including spacecraft control, social simulations, e-commerce and industrial systems management. However, since this thesis concentrates on their application to distributed sensing problems, this chapter provides a discussion and analysis of the three major projects that are related to such an application. The chapter also discusses seven open questions that provide insight into the field of multi-agent systems for indoor environment monitoring, and proceeds to review five other projects that attempt to answer some of these open questions. A comparison of the researched solutions is discussed and presented by extracting and listing their respective advantages and disadvantates, before providing our proposed answers to the seven open questions as applied to our presented solution.

2.1 Related Work

Let us analyze the three major projects that attempt to apply MASses to the problem of environment monitoring. In [7], the authors use a two-tiered blackboard architecture to build a robust and fault-tolerant system for a vehicle monitoring application. However, the system suffers from static agent organizations and impractical communications of incomplete results between the agents. In [8], the authors combine a quadtree data structure with the A*

search algorithm to monitor indoor environments. The use of an Extended Kalman Filter aids in reducing odometric errors; however, a lack of explicit communication details exists, along with an overburdening hardware-dependence for the involved agents. Moreover, in [9], the authors present a solid simulation engine used for the exploration of human-robot team interactions. The use of the *social potential fields* methods yields a robust online learning algorithm; however, the design suffers from a crude robotic positioning system, as well as purely reactive agent architectures. This thesis presents a multi-agent system composed of mobile and autonomous agents that cooperate or compete to monitor an environment. The work presents a scaleable solution, based on the reduction of environment entropy, and hence the maximization of shared information. Let us now explore, in more detail, each of the surveyed related projects.

2.1.1 Lesser and Erman

The *Distributed Vehicle Monitoring Testbed (DVMT)* paved the ground for many of today's MASses for environment monitoring. The DVMT consists of a simulation of vehicle monitoring nodes, or agents, where each node is a problem-solver associated with a sensor or a group of sensors, that analyses acoustical information to locate moving vehicles within a two-dimensional space. It is mainly due to Lesser and finds its origins in the Multi-Agent Systems Lab at the University of Massachusetts. Each problem solver is equipped with a blackboard architecture fed by knowledge sources for vehicle monitoring, and monitors an *interest area* by trying to filter out noisy sensor data. Two blackboards were actually used for each node: one for the sensory data and one for the node's goals. The entire system tries to compose a map of the vehicular movement dynamics by consolidating the information gathered by the cooperating problem solvers.

In the simulation, each vehicle generates sounds that depend on the vehicle type and velocity, and that may degrade over distance and be attenuated by atmospheric effects. The simulated sensors also exhibit physical characteristics, such as varying sensing ranges and accuracies as well as varying sensory error introductions. The imperative measure of the network

was *response time*, or how long it took for the problem solvers to synthesize a map given a set of vehicle movements. Other distinguishing measures included the *fault tolerance*, or how well the system adapted to the loss of problem solvers, the *communication robustness*, or how well the system adapted to delays and possible losses of messages between the problem solvers, and *communicative reliance*, or how well did the system distribute communication versus local problem solving.

A number of distributed artificial intelligence paradigms have been spawned from the DVMT research results and have made major contributions to the advancement of the field, most notably the contract-net protocol [22], and the distributed coordination protocols [23].

2.1.1.1 Discussion

The DVMT has become somewhat of a landmark project that still is survived today in the form of two ongoing research endeavours: the DRESUN [24] distributed blackboard-based architecture and the TAEMS [25] framework for building models of multi-agent task environments. However, the DVMT suffered from a few drawbacks, mainly its predesigned organizations where problem-solving nodes are not given the freedom of forming their own relationships at run-time. This does not allow for an adaptive solution in the case of changing inter-agent relationships. A problem-solving node requiring the attention of another that is not in its static relationship chain will not be able to interact with that particular node. Also, although the agents in the testbed coordinated their actions to recognize, classify and predict the movements of the vehicles, they lacked a robust method of communicating incomplete or inconsistent information to resolve local innaccuracies. This has been later amended by the DRESUN testbed. Another drawback of the DVMT is its homogeneous agent architecture, in that all the problem solver nodes have the same architecture and granularity, restricting the study and analysis of different agent architectures in this distributed system.

2.1.2 Pauly and Kraiss

Another major research endeavour in the field of MASses for environment monitoring is discussed in [8], where the authors describe an indoor monitoring scheme using intelligent mobile robots. The system is composed of three-tiered parallel acting agents, with each tier (or layer) representing an encapsulation of human behavior. Refer to Figure 2.1 for more detail.

Knowledge-based Layer : This layer is the top-most layer, where planning agents generate navigation plans based on the knowledge acquired by the world-modelling agent. These plans include the sequence of rooms to monitor as well as the tasks to be executed in each room;

Rule-base Layer : This layer receives the plans from the previous layer. They are divided by the planning and navigation agents into sub-tasks;

Reflex-based Layer : This layer receives the sub-tasks from the previous layer. They are directly executed by the navigation and hardware agents. Steering commands are generated here, while an active collision avoidance scheme executes along the planned path.

A *hybrid agent architecture* interleaves reactive and deliberative actions, while using a blackboard system to feed the higher-level agents with lower-level details. The environment is represented using a Quadtree, and a modified A* algorithm is used for shortest *path-planning*. The latter is performed so as to avoid dangerous or recursive situations. Dangerous situations arise from the robot visiting a dangerous region, such as near doors or stairs, indicated by a high cell-cost function component. Recursive situtations arise from the robot re-visiting the same regions too often, also indicated by a high cell-cost function component. The different cost components are aggregated for every cell in the Quadtree, and a path-plan is devised so as to travel the shortest path while avoiding hazardous conditions. *World-modelling* is performed using two levels of abstraction: a semantic/topological map about the rooms and the objects

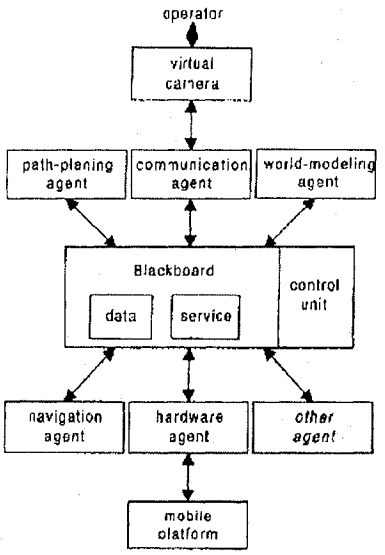


Figure 2.1: Software architecture of the realized system 2.1

contained therein, and a geometric map about the environment. An *Extended Kalman Filter (EKF)* is used to correct the POSE (position and orientation) of the robot during path traversal. *Navigation* is performed using a subsumption architecture [18] composed of five individual behaviors from highest priority to lowest: goal-following, collision avoidance, free space, wall-following and deadlock avoidance. Finally, a *virtual camera* allows a remote operator to obtain the sensory information required to control the mobile platform.

2.1.2.1 Discussion

Pauly and Kraiss conclude that the robot acts in a "robust and well-defined way" in a dynamic environment, and, they show that the workload of the entire system is low enough to elicit the use of more mobile platforms. The authors also state that the environment monitoring system has been validated in simulation and using a mobile testbed called TAURO II. We have identified some drawbacks to the system, mainly the specification of the agents involved, in that every agent does not inherit from a generic architecture the basic requirements of agency: this leads to a non-scaleable and fault-intolerant architecture, since new agents have to be re-

specified at design-time from scratch and the loss or incapacitation of certain agents will lead to a drastic degradation of the system. Another drawback of this system is the communications scheme. The authors do not mention how this communications is performed and leave it to the reader's imagination to visualize the scheme that allows the different controllers to communicate with each other. A final drawback is that the agents are targetted to a specific hardware platform, not being able to run on any other in the case of a hardware failure. This leads to an inflexible solution that requires off-line fine-tuning in the case of a system failure. Adapting to a dynamic environment becomes that much more difficult.

2.1.3 Dudenhoeffer and Bruemmer

The Command and Control Architectures for Autonomous Micro-Robotic Forces research endeavour is the closest in nature to my work. The project aims to allow the deploying, coordinating, planning and re-tasking of a robotic society by a single operator [9]. It is mainly due to Dudenhoeffer and Bruemmer, and finds its origins in the *INEEL (Idaho National Engineering and Environmental Laboratory)* [26]. The focus of the project is the military command-and-control of a large-scale robotic society while interfacing to a single operator coherently and cohesively, to better investigate the *situation awareness (SA)* where the single operator not only knows about the robots' physical locations, but their intentions as well. This factor is quite imperative in this type of application, as it portrays the confidence that humans possess about the machines and their capabilities. For this reason, a simulation system has been built by Dudenhoeffer and Bruemmer, the *AgentCommand*, that is composed of three major components:

AgentSim This is the simulation driver that allows the operators to test different control, command, coordination and behavior strategies, as the entire robotic workforce is presented in a consolidated view.

AgentCDR This is the human-robot interaction, or operator view, for the command and control of individual or groups of robots, in the field. It can receive simulated input from

AgentSim.

AgentHQ This is the command center that allows the operator to view the agent's operations by interfacing to each of the AgentCDR modules. It can receive simulated input from AgentSim.

The simulated environment proved to be very advantageous in that multiple discoveries were made before the actual implementation of the robotic force. Some of these discoveries include the *social potential fields* approach for maintaining spatial relationships between the robots, where it was found that the method was highly dependent on accurate modeling of neighbours' locations. A behavior, labelled as *chattering*, emerged as an oscillation of the robot back and forth, as it tried to remain in a neutral position, free of overlapping social potential fields. This wastes the time and energy of the robots. It is avoided by thresholding the density of the robots so as to not cause such situations. Other results discovered during the simulation phase included recursive sensing, where robots tended to recurse over already-sensed areas, and omissive sensing, where robots tended to stay away from certain areas that did not match pre-defined environmental conditions.

A physical testbed environment was built and deployed utilizing *Growbot* robots by Parallax [27]. Each robot was controlled by a pure reactive agent architecture: the subsumption architecture. As previously indicated, a solution to the chattering emergent behavior was implemented in simulation using online learning of the individual and the collective. It was also found, through real-world exploration of laboratory space as well as a warehouse floor, that it is possible to effectively cover and monitor an area of interest by a swarm of small and inexpensive robots, coordinating their actions and feeding back to a single operator their incomplete exploratory results.

2.1.3.1 Discussion

The INEEL project is a very ambitious and resourceful undertaking. Since it targets military applications, it has to deal with factors that typical indoor environment monitoring systems do not bother with. That said, the project does suffer from a few drawbacks, that we have identified, mainly the actual positional information of the robots. As proposed by the authors in [9], the robot positioning is derived by attaching a different-coloured marker to each robot and a human-eye tracking method is used to determine the robot's path and area coverage. This rather crude robot localization is a cause for many systematic errors and deficiencies, not the least of which being the inflexibility of the system when a larger number of robots need to be deployed. Another major drawback involves the lack of suitable solutions to the problems of recursive and omissive sensing. This reduces the coverage area of the robotic society as a whole, and wastes much-needed time and energy in the sensing of *already-discovered areas*. Finally, the chosen agent architecture (i.e., purely reactive with no state knowledge) does not particularly suit the requirement of socially interacting agents. The proof lies in the *emergent behaviors* of the society: no positive behavior emerged; on the contrary, negative behaviors, such as chattering, were the only emergent ones. However, it is worth mentioning that extensive work has been done on the simulation phase of the project, where most of the beneficial results and conclusions can be extracted and learnt from.

2.1.4 Other References

There are a few other works that are worth mentioning as they concentrate on a specific issue within the vast field of MASses for indoor environment monitoring. This could be anything from mapping and exploration, to perceptual efficiency, to agent localization, to futuristic applications of the field. Each one of these projects provides insight into the field, with the actual listing of the open research problems in the field having been researched and presented in [28]. They are summarized hereby:

- What is the best mapping representation of the spatio-temporal range of interest ?
- What is the best global map synthesizing technique derived from local incomplete representations ?
- How do you efficiently and accurately localize the individual agents ?
- How do you effectively share knowledge amongst the society members ?
- How to determine the plan of action of the society to minimize room entropy and/or maximize novel perception ?
- How to distribute the perception and deliberation tasks amongst the agents ?
- How to provide an effective interface to the user ?

Some of these open questions are partially answered by the projects summarized in the following sections.

2.1.4.1 Dissanayake et al.

This work on robotic localization and environment mapping presents a mathematically-elegant formalization to the problem [29]. The so-called *Simultaneous Localization and Map (SLAM)* Building problem is addressed by modelling the position of the robots as well as environmental attributed as a state vector, while modelling the uncertainty of the estimates as a covariance matrix. The motion of a robot updates the state vector accordingly and adds noise to the uncertainty vector, while sensory input updates the state vector accordingly and increases the confidence by reducing the uncertainty vector magnitude. The uncertainty vector is attributed to the uncertainties surrounding the motion of a robot (e.g., dead reckoning errors), while the confidence is attributed to the certainties obtained from new environmental data. [28].

2.1.4.2 Burgard et al.

This work on multi-robot exploration maximizes the information sensed from the environment by the colony through novel coordination techniques [30]. Each robot is assigned a set of *target points* to be explored in the unknown environment. However, unlike previous approaches, the robots calculate the aggregate cost of reaching the target points, as well as the utility of those target points. The utility of a target point is given as the size that the robot can sensorially cover when at that target point. When a target point is actually collectively assigned to a robot, the utility of that target point is mitigated for the other robots. In this way, target points are assigned to the robot with the best tradeoff between reaching the target point in the shortest time and utilizing the least energy, and gathering as much information as possible when there. This approach was tested in a simulation environment, as well as on a robotic testbed that included two physical robots. For a $15m$ x $8m$ environment, exploration time was reduced from 49 seconds when the robots were uncoordinated, to 35 seconds when they were coordinated. In simulation, this time using a $27m$ x $20m$ environment, it was concluded that two coordinated robots explore the region of interest in approximately the same time as three uncoordinated robots. Burgard et al.'s main contribution is the calculated maximization of the number of empty intersecting sets composed of target points, in other words, the minimization of the number of target points assigned to two or more robots.

2.1.4.3 Rekleitis et al.

This work on multi-robotic exploration also attempts to maximize the information gathered from the environment [31]. The approach taken here consists of a line-of-sight exploration of an unknown environment by two robots. The robots maintain a line-of-sight constraint while exploring the *large scale space*, where localization and obstacle detection is difficult using conventional methods (e.g. landmark detection or occupancy grids). Two strategies are discussed and undertaken in both simulation and experimental scenarios: the triangulation method is used

when the environment is small enough so as to allow the robots to perceive each other at all times, whilst a trapezoidal method is used when the environment is large enough to present situations where the distance between the two robots is greater than the individual sensory range of one of the robots. This work concentrated more on the mapping of the unknown environment, and we describe the authors' main contribution as the novel coordination technique between the mapping robots, that reduces accumulated dead reckoning errors and map inaccuracies.

2.1.4.4 Ponsa and Vitria

This work on a mobile monitoring system attempts to utilize an agent-oriented approach to the problem of tracking dynamic mobile robots without prior knowledge of their trajectories or appearances [32]. A monitoring station, endowed with a multi-agent system, formed the testbed of the vision-based monitoring system. The agents involved include *tracking agents* used to follow the mobile robots, with each agent embedded with a different tracking algorithm for robustness, *acquisition agents* used to acquire images from a camera at a rate proportional with the complexity of the movement, and *vigilant agents* used to distinguish the static state from the current state of the environment, hence detecting the introduction of mobile objects into the scene. Communication occurs between the agents through a shared-memory scheme, and cooperation is undertaken between tracking agents only. We describe the authors' main contribution as the multi-agent monitoring station capable of observing mobile objects in an environment without the necessity of previous knowledge of their dynamnics or appearances.

2.1.4.5 Halme et al.

This work on monitoring process plant internals attempts to utilize a robotic society of autonomous underwater robots to act as a distributed mobile sensing instrument [33]. The authors discuss a Bacterium Robotic Society, that includes autonomous underwater robots such as SUBMAR I and SUBMAR II. A simulator was created to test out the application before all the robots were built. The simulator was designed using Open GL and is a bit slow due to its

3D rendering requirements. Communication between the society members is difficult due to the nature of the medium, although infrared light has shown promise in clear liquid environments. The authors target map building as an application and utilize a directed graph approach, where nodes represent a spatially changing event, and links represent transitions from one event to another. Pose changes are only detected through onboard pressure sensors, while recharging and sample unloading were implemented to guarantee fully autonomous operation. The authors do not demonstrate how such a society can be used for environment monitoring, rather than just map building. Also, neither sensor fusion nor robotic coordination is performed. We describe the authors' main contribution as the validity of real-world runs with a OpenGL simulator that comes close to predicting the actual paths and behaviors of the robotic members.

2.1.5 Comparisons and Discussion

Shown in Table 2.1 is a comparison of the relevant indoor environment monitoring MASses prosed in this section, including each's agent architecture (AA) (i.e., PR is a purely reactive one, PD is a purely deliberative one, and H is a hybrid one), and the system's most prevalent pros and cons. In comparison with the systems presented above, DISS possesses a hybrid agent architecture, as both reactive and retroactive agent architectures are utilized, that provides system flexibility in choosing the best architecture for particular applications. DISS clearly outlines a communications protocol that allows for the agents to communicate with system access points, as well as with each other. The communication of incomplete results is performed through a blackboard architecture, the latter being a major component in synthesizing the global map from local incomplete ones. The agents are hardware independent, as the entire system was specified using a hybrid formal specification language, allowing for a level of abstraction that does not constrain the designer to a particular architecture. This has been demonstrated by the realization of three different design implementations of the physical agents: a purely software spin running on a microcontroller, a hardware/software spin running on a programmable device (e.g. FPGA) and an off-chip microcontroller, and finally, a hardware/software co-design

Authors	AA	Advantages	Disadvantages
Lesser and Erman	H	foundational work in DAI two-tiered blackboard architecture robust and fault-tolerant system	pre-designed (static) agent organizations impracticality of communicating incomplete results homogeneous AAs a hindrance
Pauly and Kraiss	H	successful Quadtree and A* combination EKF aids in reducing odometric errors hybrid and flexible AA	overburdening customization in the design of agents lack of explicit communications details hardware-dependent agents
Dudenhoeffer and Bruemmer	PR	robust simulation engine in SimAgent solid social potential fields method online learning solved	crude robotic POSE no solution to recursive or omissive sensing purely reactive AA a detriment

Table 2.1: Comparison of MASses for environment monitoring

running on a programmable device and an on-chip, soft-core microprocessor. Finally, DISS presents separate simulation and virtualization engines. The simulation engine allows the user to simulate MAS-based scenarios to optimize design factors while extracting environmental parameters. Once the system has been optimized and realized, a separate virtualization engine is used to seamlessly immerse the remote user into the MAS environment, even across the Internet, if required.

Let us briefly review the questions that were posed in section 2.1.4, and relate them to our work. First, there is no consensus on an **ideal mapping representation** for the spatio-temporal range of interest; we have chosen a quadtree-based representation, as it allows for both a multi-resolution and multi-dimensional representation of the environment. Second, the data structure is compact enough to be realized on an embedded system; it feeds a networked virtual environment, that allows for its offline storage for future temporal analysis of environmental events. More details about the data structure will be presented in section 3.2.

As will be seen later in this thesis, our **global map synthesis** is performed, centrally, by feeding a centralized processor that holds the global map, and distributively, by feeding a mapping agent, endowed with enough storage capacity to hold the global map. Each individual agent holds a local incomplete representation; it queries either the centralized processor or the mapping agent for an updated view when traversing outside of its spatial range. Details about the global map synthesis are presented in section 3.1.3.2.

In our system, the agents are **localized** by fusing incremental wheel encoder information

with external landmarks for a better pose estimate. A magnetic digital compass [34] is used for heading correction, and an infrared beacon [35] is used for position correction. It is worth noting that the beacon, initially used to indicate the Cartesian origin (0,0), was often occluded by obstacles and was eventually not used. The eventual solution consisted of dropping the agents one by one into the environment starting at the origin, since the odometric scheme provided good positioning estimates. Agent navigation is performed using reactive control schemes (i.e., the subsumption architecture) and feedback systems (i.e., proportional-integrator (P-I) controllers). Details of the localization and navigation schemes are presented in section 6.6.

Knowledge sharing is performed using a simplified blackboard system, since an agent's knowledge is integrated into the global map, residing on the mapping agent or the centralized processor, that can then be queried by the same agent that produced the original sensorial event, or by other agents requiring synchronization. This scheme is not further discussed in this thesis, as it is inherent by the presense of the global map synthesis technique mentioned above.

To determine the **plan of action of the society**, each agent is provided with the goal of environmental entropy minimization. In the centralized scenario, a processor determines the sequence of actions for each agent; however, in the distributed case, each agent performs its own goal and path planning to reach its world objectives. Details of action planning are presented in sections 3.1.3.1 and 3.1.3.2.

In the distributed scenario, each agent decides whether it is more cost effective to navigate and perceive world events, or communicate and deliberate with other agents to receive the same information from another source. This **perception vs. deliberation** decision is performed by estimating the costs of energy consumption in both cases. A simplified energy consumption equation is used, that compares the cost of travelling to a location and sensing an environmental parameter vs. the cost of wireless transmission and reception of that same value from another agent. The distributed case is discussed in section 3.1.3.2.

Finally, an effective **user interface** is used to immerse the human into the agent society's world. The concept of a *virtualized reality model of the environment*, where a virtual model is

fed with physical data, will be elaborated upon, and represents a seamless and interactive user interface to the entire system. Details of the immersive virtualized reality model are presented in section 7.1.

2.2 Summary

Multi-agent systems provide an effective solution to distributed sensing applications. This chapter presents and discusses advantages and disadvantages of previous and related solutions. It was concluded that success was mainly achieved due to blackboard-based knowledge sharing, odometric error correction, simulation engine presence and real-time mapping representation and searching techniques. Conversely, it was found that problems with scalability, robustness and emergent behaviors resulted mainly from static agent organizations, customized agent designs, homogeneous agent architectures and incomplete results communications.

There are seven themes that classify each multi-agent system, namely the mapping representation, the global map synthesis, the agent localization scheme, the knowledge sharing method, the plan of action of the society, the perception/deliberation dilemma, and the user interface. This chapter has described a multi-agent system, called DISS, that possesses a quadtree-based *mapping representation* allowing for both a multi-resolution and multi-dimension depiction of the environment. DISS either synthesizes the *global map* through a centralized processor, or through a distributed mapping agent. DISS utilizes wheel encoder information and a magnetic digital compass to derive *agent localization*, while *sharing knowledge* through a blackboard system. The *plan of action* is directed by each agent's goal of entropy minimization. In the distributed case, each agent decides whether it is more efficient to *perceive*, which includes the navigation and sensing actions, or to *deliberate*, which includes the communication and deliberation actions, to minimize energy consumption. Finally, a virtualized reality model of the environment interface is presented and discussed as the *user interface* solution of this system.

Chapter 3

Entropy Minmization Model

"The only people for me are the mad ones, the ones who are mad to live, mad to talk, mad to be saved, desirous of everything at the same time, the ones who never yawn or say a commonplace thing, but burn, burn, burn, like fabulous yellow roman candles exploding like spiders across the stars and in the middle you see the blue centerlight pop and everybody goes "Awww!" "

(Jack Kerouac)

In this chapter, we demonstrate our concepts for controlling and using multiple collaborating robotic sensor agents, each of which has two independently-driven wheels, two static gliders used for stability, and a range sensor for object detection. The agent motion and range sensor models are presented for the differentially-driven robots. We derive the Cartesian coordinates of the agent, the sensor and any object that is detected by the range sensor. We assume that the object's motion is fixed, and that the driven wheels and static gliders exhibit minimal friction. The agent's posture is represented by the position vector $[x_t^e, y_t^e, \sigma_t^e]^T$, where t is the time, x_t^e and y_t^e represent the agent's location in Cartesian coordinates within the experimental frame, and σ_t^e represents the agent's heading direction taken as the orientation angle of the forward-pointing perpendicular bisector of the drive axle. The posture update takes into account the agents' axle distance and turn radius, and is defined by Equation 3.2 for the straight-line update, and Equation 3.3 for the non straight-line update.

Entropy, as it relates to the thesis' main application, is defined as the information entropy

$H(X) = -\sum_{i=1}^{n} p(i)log_2 p(i)$, where X is a discrete random event with possible states $1..n$. It is used as the minimization metric in the developed environment mapping method, in combination with a quadtree-based data structure representation of the environment. Mutual information is used as the maximization metric and is defined as the amount of information shared by 2 or more random variables. In the case of 2 variables, mutual information is defined as $I(X;\ Y) = H(X)+H(Y)-H(X,Y)$, where $H(X)$ and $H(Y)$ are the entropy measures of the random variables X and Y, respectively, and $H(X,Y)$ is the joint entropy of both variables. Mutual information is used to integrate sensor landscape maps together. More details are presented below in this chapter.

3.1 Data Models

To integrate our sensor agents into our environment, it is important to generate three data models: a motion model, a sensor model and a multi-sensor fusion (MSF) model. Our **robotic intelligent sensor agents** (R-ISAs or ISAs for short) are differential-drive robots, whose motion has been analyzed and modeled in [19]. Using the results of that paper, we will introduce our data models for our sensor agent.

3.1.1 ISA Motion Model

Forward kinematics of wheeled mobile robots have been well-developed in [36] and [37] for numerous types of mobile robots. In [37], the authors not only realize a kinematic model for five types of *wheeled mobile robots (WMRs)* (called the *posture kinematic model*), but also develop a *configuration dynamical model* that takes into account the dynamics of the robot, including the torques provided by the motors. In this thesis, we will concentrate on Type (2,0) robots, defined as a "robot with two conventional fixed wheels and one conventional off-centered orientable wheel" [37]. Figure 3.1 (extracted from [37]) shows a Type (2,0) WMR (differential-drive), with P being our source 2D position, and L being constant. These types of robots will be referred to as ISAs from now on, to keep our nomenclature uniform.

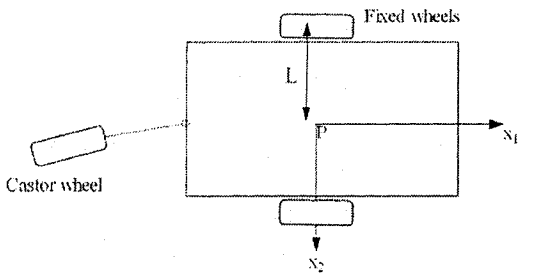


Figure 3.1: Type (2,0) WMR

Let us define ξ_t as the posture of the sensor agent at time t, represented by the position vector $[x_t^e, y_t^e, \sigma_t^e]^T$, where x_t^e and y_t^e represent the agent's location in Cartesian coordinates within the inertial basis, i.e., the global coordinate frame, and σ_t^e represents the agent's heading direction, taken as the orientation angle of the basis $\{\vec{x_1}, \vec{x_2}\}$ with respect to the inertial basis $\{\vec{I_1}, \vec{I_2}\}$ (left graph in Figure 3.3). Analyzing the robot frame R (right graph in Figure 3.3), it can be noticed that the presence of two off-centered wheels at either side of P would characterize the ISA position (P) with respect to frame E by the three constants m, α, and ϕ. In our case, m = L, α = 0, and ϕ = 0. The ISA motion with respect to frame E is characterized by two time varying angular velocities, namely the left (ω_t^l) and right (ω_t^r) wheel angular velocities. The robot's trajectory is finally defined as the sequence of all its postures $\xi_1, \xi_2, ..., \xi_T$, and note that every agent motion involves three operations in a *point-and-shoot* style of navigation:

- Rotate towards desired position
- Travel in a straight-line motion towards the desired position
- Rotate again to match the desired orientation

This process repeats for every consequent position, that each agent performs along its trajectory.

Considering the actual realization of the ISA motion models and our decision to use the last design of the agents, consisting of differential-drive locomotion, it is worth exploring the implementation constraints on the theoretical models. The differential-drive agents use two motors: a left and a right one (e.g. see subsequent Figure 6.4). Acceleration is neglected,

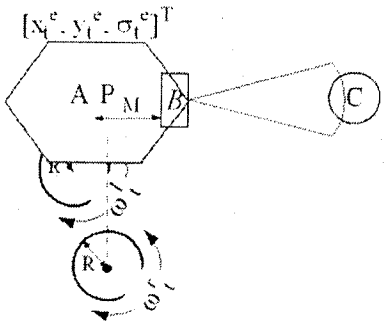


Figure 3.2: ISA component structure

that implies that changes in speed are nearly instantaneous, an assumption that is valid for lightweight vehicles. The localization scheme combines dead-reckoning odometry, along with a magnetic compass for heading self-correction. If the agent's wheels are moving at the same speed, then the motion is a straight-line one and is easily modeled by Equation 3.2; however, if the wheels are moving at different velocities, then we treat the agent as a rigid body and use equations similarly to those used in [38] and [39]. We first calculate the agent's turn radius, r, to model the circular trajectory of the agent's center, given the main axis distance, b, measured from center-to-center, and the left (S_L) and right (S_R) displacements (depicted in Figure 3.4):

$$r = \frac{b(S_R + S_L)}{2(S_R - S_L)} \tag{3.1}$$

The posture update for the experimental differential-drive agent follows a generic equation; however, it takes into account the physical features of the agent, mainly the axis distance and the turn radius. The straight-line update thus becomes:

$$\hat{\xi} = \begin{pmatrix} x_t^e + S_R cos(\sigma_t^e) \\ y_t^e + S_L sin(\sigma_t^e) \\ \sigma_{t-1}^e \end{pmatrix} \tag{3.2}$$

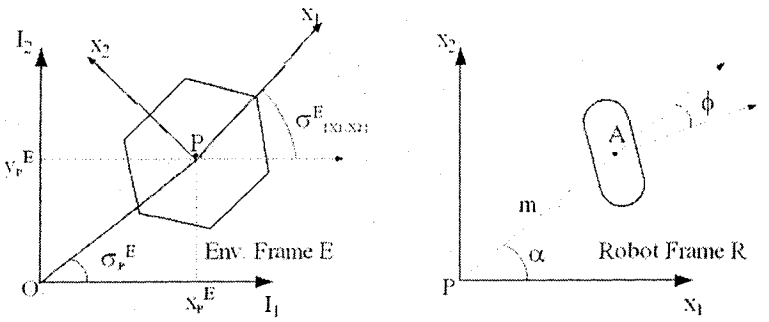


Figure 3.3: ISA motion model

while the non-straight-line update becomes:

$$\hat{\xi} = \begin{pmatrix} x_t^e + r[sin(\sigma_t^e) - sin(\sigma_{t-1}^e)] \\ y_t^e - r[cos(\sigma_t^e) - cos(\sigma_{t-1}^e)] \\ \frac{(S_R - S_L)}{b} + \sigma_{t-1}^e \end{pmatrix} \tag{3.3}$$

3.1.2 Range Sensor Model

The range sensor position (B) with respect to frame R is governed by three constants, namely l, δ and λ (refer to left graph of Figure 3.5). In our case, as can be seen in Figure 3.2, l = M, δ = 0, and λ = 0. The sensor motion is stationary with respect to frame R; however, if we were to allow it to rotate, then it would be characterized by the time-varying rotation angle $\lambda(t)$.

Since the basis $\{\vec{x_1}, \vec{x_2}\}$ is rotated $\sigma^E_{\{x_1,x_2\}}$ from the basis $\{\vec{I_1}, \vec{I_2}\}$, the sensor position will have the following Cartesian coordinates with respect to frame E:

$$\begin{pmatrix} x_B^E \\ y_B^E \end{pmatrix} = \begin{pmatrix} x_P^E + lcos(\sigma^E_{\{x_1,x_2\}}) \\ y_P^E + lsin(\sigma^E_{\{x_1,x_2\}}) \end{pmatrix} \tag{3.4}$$

where,

x_B^E is the x-coordinate of sensor B at instant time t;

y_B^E is the y-coordinate of sensor B at instant time t;

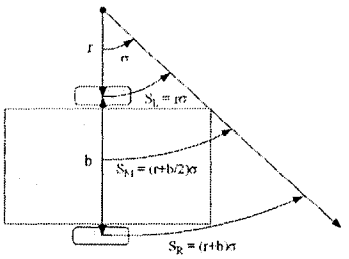


Figure 3.4: Differential-drive dead-reckoning kinematics

$\sigma^E_{\{x_1,x_2\}}$ is the angle that the basis $\{\vec{x_1}, \vec{x_2}\}$ makes with $\{\vec{I_1}, \vec{I_2}\}$.

An obstacle can be detected by the range sensor if the former falls within the latter's dispersion angle and directivity range. Assuming that the obstacle does, its position (C) with respect to the sensor frame S is characterized by two variables, namely $u(t)$ and $\sigma(t)$ (refer to the right graph of Figure 3.5). We also assume that the obstacle's motion is stationary, and hence its velocity is 0.

The obstacle position is represented by the following vector $[ucos(\sigma), usin(\sigma)]^T$, where $r^S_C = ucos(\sigma)$ represents the sensed range of the obstacle, and $d^S_C = ucos(\sigma)$ represents the sensed distance of the obstacle. The subscript of the previous variables indicates the entity being measured (i.e. P $\rightarrow$ agent, A $\rightarrow$ centered wheel, B $\rightarrow$ sensor and C $\rightarrow$ obstacle). Meanwhile, the superscript of the previous variables indicates the reference frame (i.e. E $\rightarrow$ environment, R $\rightarrow$ robot or agent and S $\rightarrow$ sensor). Since the basis $\{\vec{x_1}, \vec{x_2}\}$ is coincident with the basis $\{\vec{x_3}, \vec{x_4}\}$, we reproduce the sensor position conditions applied to Figure 3.5:

$$l = M, \delta = 0, \lambda = 0 \tag{3.5}$$

Given the sensing range (r^S_C) and the sensing distance (d^S_C) of the obstacle C with respect to frame S, we can deduce the detection angle of the obstacle (σ^S_C) with respect to frame S as being:

$$\frac{r^S_C}{d^S_C} = \frac{usin(\sigma)}{ucos(\sigma)} = tan(\sigma^S_C) \tag{3.6}$$

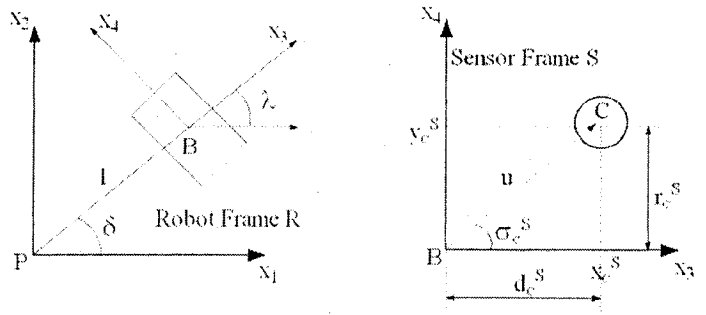


Figure 3.5: Range sensor model

$$\sigma_C^S = arctan(\frac{r_C^S}{d_C^S}) \tag{3.7}$$

Using (3.7), the obstacle C will have the following Cartesian coordinates with respect to frame S:

$$\begin{pmatrix} x_C^S \\ y_C^S \end{pmatrix} = \begin{pmatrix} u_C^S cos(\sigma_C^S) \\ u_C^S sin(\sigma_C^S) \end{pmatrix} \tag{3.8}$$

where,

x_C^S is the x-coordinate of obstacle C at instant time t;

y_C^S is the y-coordinate of obstacle C at instant time t;

σ_C^S is the detection angle of obstacle C at instant time t;

u_C^S is the detection distance of obstacle C at instant time t.

And since the bases of frames R and S are coincident and governed by the conditions listed in (3.5), we can deduce the obstacle's Cartesian coordinates with respect to frame R as:

$$\begin{pmatrix} x_C^R \\ y_C^R \end{pmatrix} = \begin{pmatrix} x_C^S + l \\ y_C^S \end{pmatrix} \tag{3.9}$$

where,

l is the distance from point P to point B.

Transforming back to the global coordinate frame, we can deduce the obstacle's Cartesian coordinates with respect to frame E as:

$$\begin{pmatrix} x_C^E \\ y_C^E \end{pmatrix} = \begin{pmatrix} x_B^E + u_C^S cos(\sigma_C^S + \sigma_{\{x_1,x_2\}}^E) \\ y_B^E + u_C^S sin(\sigma_C^S + \sigma_{\{x_1,x_2\}}^E) \end{pmatrix} \tag{3.10}$$

3.1.3 MSF Models

A multi-sensor data fusion mechanism is used for the integration of heterogeneous sensory data into composite models of 3D object shape, surface and material properties, heat transfer and radiation (EM, thermal, radioactive, optical, etc.) characteristics. The multi-sensor fusion framework [40] deals in a consistent way with a diversity of measurement data produced by the ISAs. Such a multi-sensor fusion system has to:

(1) Organize the data into collections and process signals from different types of sensors;

(2) Produce local and global world models using the multi-sensor information about the environment; and

(3) Integrate the information from the different sensors into a continuously updated model of the system.

There are four main sensor network configurations: complementary, competitive, cooperative and independent [41]. Refer to Table 3.1 for a more detailed description of all four configurations. In this thesis, we will be considering both complementary and competitive sensor network configurations. Note that our sensor network is composed of ISAs that either work together or against each other in resolving a common goal.

To combat the computationally intensive real-time operations that are required in intelligent sensor processing, one needs to carefully choose the data structure that will represent and store the raw data. The requirements are that the data has to be efficiently stored, manipulated and transferred. Trees, graphs, sparse arrays and wavelets are some of the data structures that

Sensor Network Configuration	Description of Sensors and Storage
Complementary	Sensors do not directly depend on each other Combined by appending to provide a more complete picture of the environment
Competitive	Provide independent information about the same phenomena Combined by direct competition
Cooperative	Provide information that cannot be deduced from only one sensor Combined in the preprocessing phase
Independent	Provide unrelated information Can be stored in the same data structure

Table 3.1: Sensor network configurations

meet the previous requirements. Each has its advantages and disadvantages; however, tree-based data structures promote very efficient data processing and storage, and 2^N-trees in particular, provide an organized view of the environment at multiple resolutions. As one travels from the root down to the leaves, one can view the environment at higher resolutions. In choosing 2^N-tree-based structures, we decided to use the *forest of quadtrees (FoQ)* data structure [42], since it greatly reduces the number of nodes required to store a quadtree and can easily be extended to the multidimensional 2^N-tree.

Another factor that has to be explored when dealing with multi-sensor fusion is whether the processing is centralized or decentralized. **Centralized MSF (C-MSF)** requires all sensors to transmit local information to the processor, that in turn performs the fusion, as well as the global estimation. **Decentralized MSF (D-MSF)**, on the other hand, requires the sensors to transmit local information to each other, process that information locally, and together synthesize a global estimate out of all the local estimates provisioned by each sensor in the network.

3.1.3.1 Centralized MSF

Figure 3.6 represents the flow of centralized multi-sensor fusion. In our case, the sensor agents are mobile. Each agent provides the C-MSF processor with local sensory information as described by Figure 3.7, and the C-MSF processor fuses all the information and sends positional information back to the sensor agents. The latter proceed to move towards their new positions for further sensing. Figure 3.8 shows the processor flow upon reception of new sensory infor-

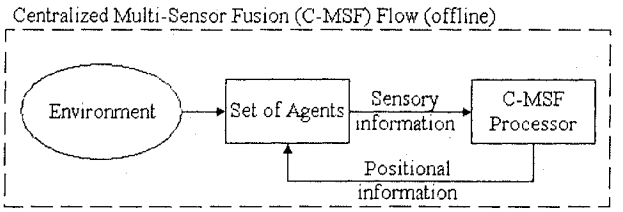


Figure 3.6: Centralized multi-sensor fusion flow

mation. First, a relationship function decides whether the new information is complementary or competitive in nature. The information synthesis module decides the next wheel angular velocities (ω), while the information integration module places the new sensor values at the appropriate nodes of the FoQ data structure. The processor then decides, for each agent, which high-entropy region to explore next, and proceeds to inform all the agents of their new posture goals by transmitting to them their respective next angular velocities. Upon reception of the latter, the agents proceed in those new directions and senses the environment there, whence the entire flow repeats.

3.1.3.2 Decentralized MSF

Figure 3.9 represents the flow of decentralized multi-sensor fusion. In this scheme, each agent sends sensory, as well as positional, information to other agents, and decides, autonomously, where it will explore the environment next. Figure 3.10 shows the agent flow in a D-MSF scheme. As in C-MSF, the ISA can detect parameters from the environment or receive information from other agents. If detecting, the agent first pre-processes the data to compute the local estimate in the agent (robot) frame. The local pre-processing module also filters the raw sensory data to account for sensor-specific anomalies. The local estimate is then spatially aligned within the global coordinate (environment) frame. The information synthesis and integration modules' functionalities are similar to those described in the C-MSF section above, with a minor difference: once the next wheel angular velocities are calculated, the sensory and

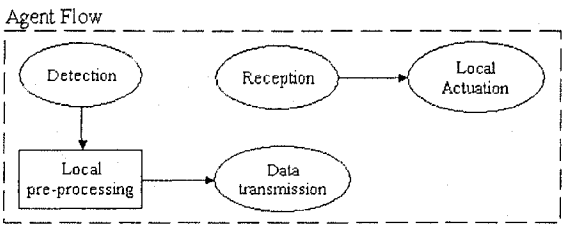


Figure 3.7: C-MSF agent flow

positional information is broadcast to the other agents, informing the latter of the current state of the environment at the agent's present location. This is needed to build a global map of the environment by synthesizing the local information held by each of the ISAs.

If the agent is receiving data from another ISA, then a different flow is taken. First, a decision is made whether the new message is a request from another agent, an acknowledge for an earlier transmission or just a broadcast announcement. If the message is a request, then a deliberation module decides whether the request should be fulfilled by the receiving agent or not. If so, then the agent acknowledges the request and then heads to its new goal, otherwise discards the message. On the other hand, if the message is an acknowledge for a previous request, then it is first decided whether the acknowledge is for a map or information request, by decoding the incoming packets. In the case of the former, the FoQ data structure is updated with a new local map; and in the case of the latter, the agent's plan is updated to include the contract between this agent and the agent that agreed to the request. Finally, if the message is an announcement, then a relationship synthesis module determines whether the new relative position-dependent sensory information is complementary or competitive. The data are fused accordingly into the agent's local model of the environment, and sent to the information synthesis and integration modules for system broadcasts and map integration, respectively.

Note that in Figure 3.10, the modules drawn with dotted lines after the reception module designate an extra functionality of a special kind of ISA, called the *mapping agent*, endowed with extra storage capacity so as to fit the global FoQ data structure, and hence the entire en-

vironment map. The mapping agent can be queried by the other ISAs, whenever any of the latter ventures beyond the boundaries of their local map, and need updated views of their new environment.

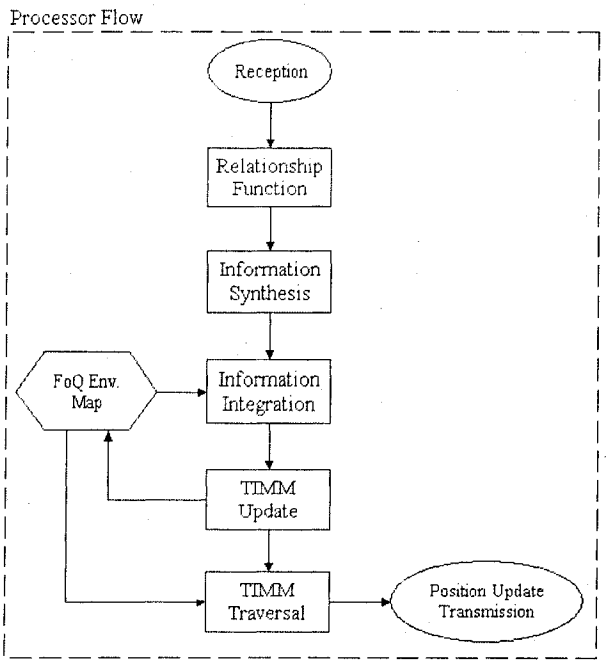


Figure 3.8: C-MSF processor flow

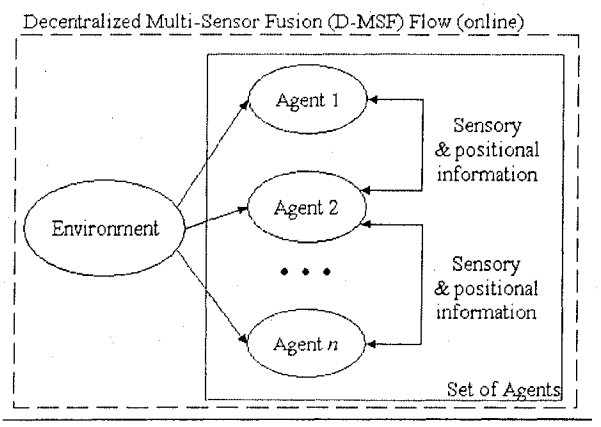


Figure 3.9: Decentralized multi-sensor fusion flow

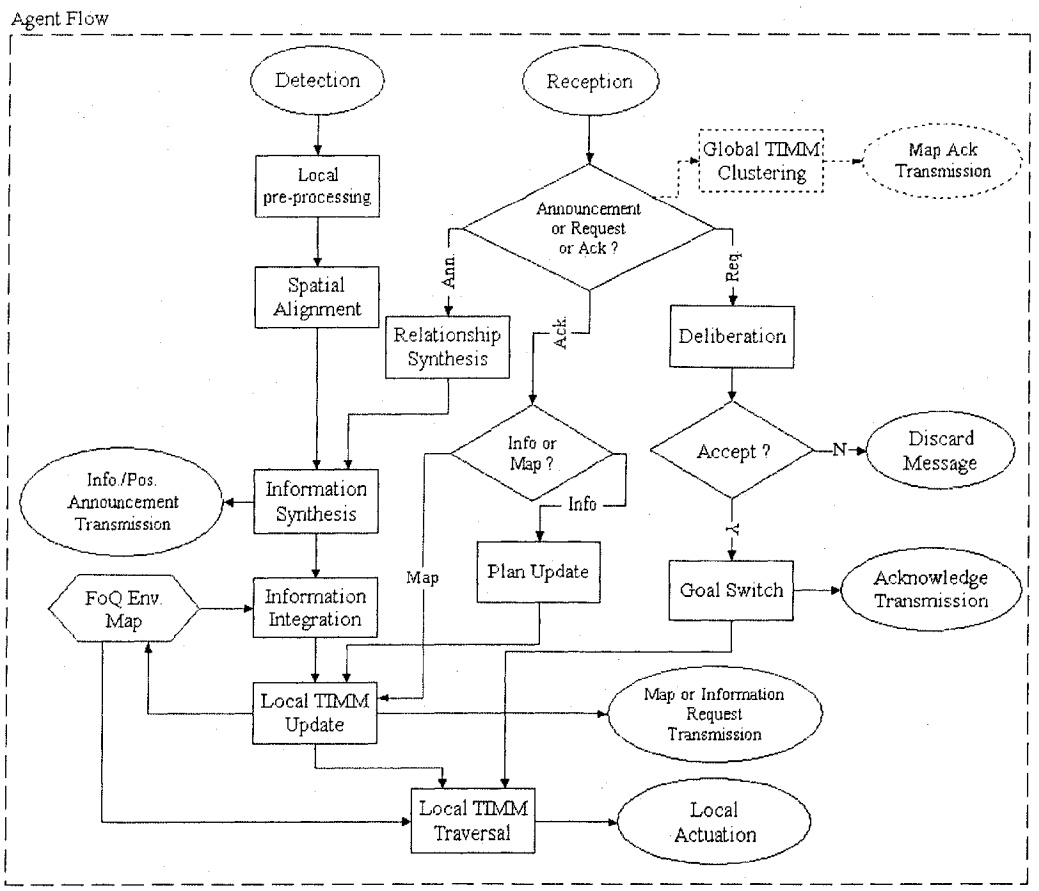


Figure 3.10: D-MSF agent flow

3.2 Tree-In-Motion-Mapping

Due to Elfes [43], an environment can be tessellated and each cell assigned a probability value that the corresponding location in the environment is occupied (1) or empty (0). Initially, all tessellations are assigned a 0.5 probability value, implying that each cell is equally likely to be occupied or empty. This scheme is termed as the *occupancy grid framework (OGF)*, and has been successfully used for the mapping of an unknown environment [14], [44]. The cell probabilities are updated by using Bayes' theorem, which is not a very computationally-intensive operation; however, all those cells affected by a range reading are updated, and that imposes a time delay that is not appropriate for an embedded system processing data in real-time. For this reason, *histogram in-motion mapping (HIMM)* is often used instead of the occupancy grid technique, by associating a histogram with each tessellation of the environment. All cells start with a zero value, that is incremented or decremented, depending on each new sensor reading. The result is an environment map that is similar to that generated using the occupancy grid framework; however, it is simpler and faster to compute, as each sensor reading only affects the cell that lies a measured distance away from the sensor, but on the latter's acoustic axis (in the case of ultrasonic sensors) [45]. Note that both the OGF and HIMM techniques make use of the *empty sector* between the sensor and the object, with the OGF method computing and projecting a negative probability function, and the HIMM method simply decrementing the certainty value, for all cells in that sector.

The HIMM technique works well for single sensor readings, but what if multiple readings, mostly of different physical phenomena, have to be combined together into one data structure, or certainty value ? The HIMM and OGF techniques are well-established mapping methods for the unified representation of data from different but similar sensors, such as ultrasonic and proximity sensors. As well, the HIMM technique contains only one resolution of the environment. This does not allow for efficient storage and transfer of the environment map when considering that sensor agents are limited in storage and communications bandwidth. A solu-

tion better tailored towards computationally-limited devices is the **Tree In-Motion Mapping (TIMM)** technique, that integrates the simplicity and uniformity of the occupancy grid framework and the low storage and high speed of computation advantages of the HIMM method. By tessellating the environment, storing the probabilities of occupancy for each location, and incrementing/decrementing those probabilities similarly to the HIMM method, one can map an unknown environment very quickly and efficiently. However, the TIMM technique associates an FoQ data structure with the tessellated environment, allowing for a multi-resolution and multi-dimensional view of the environment. Note that in this thesis, at the lowest level resolution, an ISA only occupies one cell.

Referring to Figure 3.11, one can see an example 4x4 tessellated environment, along with the associated quadtree representing each cell of the environment. The first node, the root node, is designated by an R, and contains 4 pointers to the 4 cardinal regions of the environment, namely northwest (NW), southwest (SW), southeast (SE) and northeast (NE). In turn, each one of these nodes contains 4 pointers, adding up to the 16 cells of the environment. Note the indexing scheme of the cells, as one traverses the tree a level at a time, one passes through the nodes in the following order: NW $\rightarrow$ SW $\rightarrow$ SE $\rightarrow$ NE.

The line of numbers just under the quadtree represents each cell's probability of occurrence (i.e. that the cell is occupied) at a particular time instant. The next line represents the **entropy** of that cell (i.e. the uncertainty of measurement). Entropy is the measure of disorder or randomness present in a signal or random event. However, as per Claude E. Shannon's now famous 1949 paper "A Mathematical Theory of Communication" [46], it can also be viewed as the measure of information carried by that same signal or event. Shannon's information entropy is now taken as the measure of new information contained in a measurement event, as opposed to the deterministic portion of the message, that contains no new information. Shannon defines entropy, as measured in bits, in terms of a discrete random event X with possible states $1..n$ as:

$$H(X) = -\sum_{i=1}^{n} p(i) log_2 p(i) \tag{3.11}$$

In an environment map building case, $n = 2$, since there are two possible states: either empty or occupied, $p(i)$ is the i^{th} cell's probability of occurrence, and $H(X)$ is the entropy of the measurement event X. The probabilities $p(i)$ are initialized to $1/n$, where n is the total number of states for each sensor, and represent the probability of that i^{th} state occurring. For example, for obstacle sensing, the two states are occupied or empty, whereas for temperature sensing, the three states are hot, normal and cold. The third line in Figure 3.11 represents the entropy measure of each of the 4 cardinal regions of the environment, and is a summation of the child nodes' entropies. Similarly, the final line in Figure 3.11 represents the entropy measure of the entire environment, and is calculated by adding up its child nodes' entropies. However, in our environment monitoring case, the number of states, n, increases in a combinatorial fashion, since combinations such as "the cell is probably empty and most likely contains a hazard that triggered the temperature alarm" become possible! A robust fusion algorithm is required to aggregate all the detailed sensory data, but more importantly is the interpretation of this data. As we shall later see (section 3.3), we have integrated a technique that controls the non-scalability of entropy by attempting to relate measured events and detected objects, and extract information about one by observing the other, and vice versa.

Let us further explore the environment in Figure 3.11 and note that at the highest level (i.e. 3), the maximum entropy value is 1 bit/cell. Hence, if a cell's probability of occurrence is high or low, then its entropy measure is quite low, whilst if a cell's probability of occurrence is neither high nor low, then its entropy measure is quite high, indicating that one still requires close to 1 bit to represent the state of the cell. Moving one level back (i.e. 2), the maximum entropy value is 4, since in a completely unknown cardinal region, one would need at most 4 bits to represent the states of the cells within that region. A similar analysis can be held for the root node, that has a maximum entropy value of 16. Since the root entropy value is 8.77 for this example, one can say that we need 9, instead of 16, bits to encode the states of the environment in Figure 3.11. It is easy to see that the maximum entropy value is 16 bits of information (one bit for each cell), indicating that the map is totally unknown, and the minimum entropy value

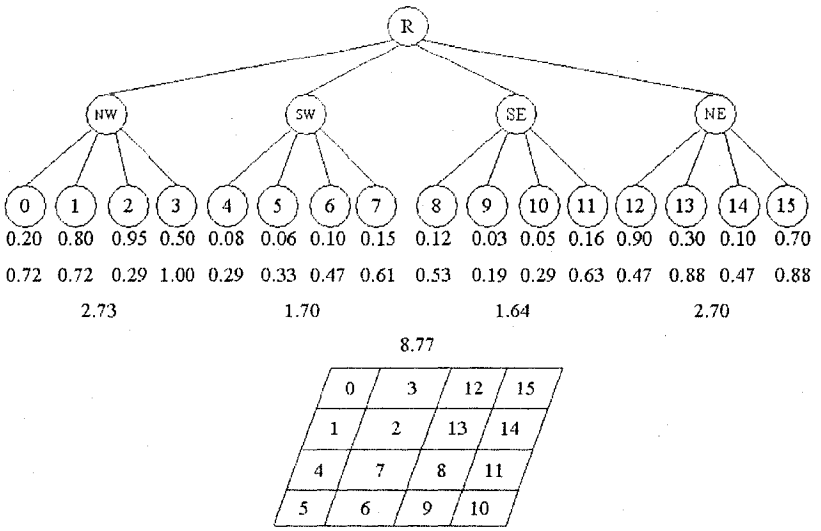


Figure 3.11: TIMM for a 4x4 environment

is 0 bits of information (i.e. the environment is entirely occupied or empty), indicating that the map is entirely known. Finally, for our application, one can also notice that the goal of entropy reduction is simplified to maximizing or minimizing the cells' probabilities of occurrence, that is, to maximize one of the probabilities, and hence to minimize all the others, to settle on a state for that particular cell. Mathematically, we can define the environment's entropy cost function as being:

$$J = \alpha H \tag{3.12}$$

where,

α is the entropy rate; and

H is the entropy cost function defined in (3.11).

Deriving (3.12) with respect to the probability that a cell is occupied (P_f), we obtain the entropy reduction rule:

$$\frac{\partial J}{\partial P_f} = -\frac{\alpha log_2(P_f)}{ln(2)} \tag{3.13}$$

Hence, to minimize entropy ($H \rightarrow 0$), one has to either maximize or minimize each cell's probability of occurrence ($Pf \rightarrow 0$ or $Pf \rightarrow 1$). This becomes the goal of each sensor agent, as well as the C-MSF processor, in attempting to uncover the unknown environment through a series of sensor readings. Actually, we can be show that in cases where there are n possible states to consider, under the constraint that all probabilities must sum to 1, the generalized entropy reduction rule can be defined as:

$$\forall i : \mathbb{I} \mid i \in 1..n \bullet \frac{\partial J}{\partial P_i} = -\frac{\alpha log_2(P_i)}{ln(2)}, \text{where} \quad \sum_{i=1}^{n} P_i = 1 \tag{3.14}$$

Hence, it suffices to drive one of the probabilities to either 0 or 1 for the goal to be achieved, that concurs with our analysis, since we would like to classify each tessellation as being in a unique state, and not in a combination of partial states (e.g. either empty or occupied, rather than a degree of emptiness or occupation). It will also be shown that the TIMM technique contains an inherent and novel method for obstacle avoidance, in that the regions of high entropy will be designated as those that are most likely to contain obstacles, and thus should be avoided.

3.2.1 TIMM Details

There are currently two popular techniques for map building an unknown environment: *OGF* and *HIMM*. We will describe each method's *mean bias*, indicating its initial average CV, *value range*, indicating its range of allowed CVs, *exploration strategy*, its weight category that will be explained later, and *update method*, its approach for updating the CVs. The OGF technique has a 0.5 mean bias, a [0, 1] value range, a *middleweight* exploration strategy, and a Bayes' *probabilistic* update that follows the relation given by Bayes' rule:

$$CV_{ij}(occ \mid r) = \frac{p(r \mid occ) \cdot CV_{ij}(occ)}{p(r)} \tag{3.15}$$

where,

$CV_{ij}(occ \mid r)$ is the updated (posterior) certainty value of the occupancy of cell (i,j) given reading r;

$p(r \mid occ)$ is the likelihood of reading r being true given the occupancy state;

$CV_{ij}(occ)$ is the prior certainty value of the occupancy of cell (i,j);

$p(r)$ is the constant normalizing factor.

The HIMM technique has a 0 mean bias, a $[0, 15]$ value range, a *lightweight* exploration strategy, and a *histrogrammic* update that follows the relation below [45]:

$$CV'_{ij} = CV_{ij} + I^{+} + \sum_{\substack{p,q=-1 \\ p,q=1}} (w_{p,q} \cdot CV_{i+p,j+q}) \tag{3.16}$$

where,

CV_{ij} is the previous certainty value of cell (i,j);

CV'_{ij} is the updated certainty value of cell (i,j);

I^{+} is the constant increment (e.g. I^{+} = 3);

w is the weighing factor based on the growth rate operator (GRO), with the following value:

$$w_{p,q} = \begin{cases} 0.5, & \text{if } p = \pm 1 \ \& \ q = \pm 1; \\ 1, & \text{if } p = q = 0. \end{cases} \tag{3.17}$$

The TIMM technique has a 1 mean bias, a $[0, 1]$ value range, a *heavyweight* exploration strategy and an *entropic* update that follows the relation given in equation (3.11). The latter is recalculated once the individual probabilities change, and the previous entropy value is discarded in the agent's memory. Let us explain the various weight categories when it comes to exploration strategies:

(1) **Lightweight** (e.g. HIMM approach): The goal of this strategy is to attack cells with CVs that are closer to the minimum of the value range (e.g. 0) and drive their CVs towards the maximum of the value range (e.g. 15), hence requiring a low store for CVs, a fast and simple CV update (addition of I^{+} or subtraction of I^{-}), and a rather

complex decision making process of which area to explore next given a flat landscape of CVs: i.e., choose regions with low and medium CVs;

(2) **Middleweight** (e.g. OGF approach): The goal of this strategy is to attack cells with CVs closer to the median of the value range (e.g. 0.5) and drive their CVs towards the extremes of the value range (e.g. 0 or 1), hence requiring a medium store for CVs, a slow and complex CV update (according to Bayes' rule), and a simple decision making process of which area to explore next given a flat landscape of CVs: i.e., choose regions with medium CVs;

(3) **Heavyweight** (e.g. TIMM approach): The goal of this strategy is to attack cells with CVs closer to the maximum of the value range (e.g. 1) and drive their CVs towards the minimum of the value range (e.g. 0), hence requiring a medium store for CVs, a fast and simple CV update (summation of products according to an entropic update), and a simple decision making process of which area to explore next given a flat landscape of CVs: i.e., choose regions with high CVs.

Note that unlike OGF and HIMM, the TIMM method does not update cells in the *empty sector* (i.e. cells in between the sensor and the perceived object), rather it uses a sensor threshold value to register whether a particular cell is likely to be occupied or not. This decision was made to demonstrate the utility of mapping with sensors possessing relatively short sensing distances (e.g. IR sensors), as opposed to the sensors utilized in the HIMM and OGF experiments, that were mostly ultrasonic sensors. As well, this demonstrates that the rapid sampling of the sensor, and the quick update of the cell's certainty value combine to provide an in-motion mapping functionality similar to the previously used techniques.

As well, the TIMM technique contains a statistical component to each cell's certainty value, demonstrated if we take a look at the individual state probabilities, that over time, are polarized by the agents' entropy reduction rules to one of the extremes, through a series of simple increments and decrements, as in the HIMM method. However, for storage purposes, the

individual state probabilities are not kept in the agent's memory space; however, they are instantaneously stored in the access point's/mapping agent's memory, and histogrammically stored in the virtualized reality model of the environment application, for future playback and analysis. Note that the entropic update occurs after the increment or decrement of the $p(i)$s. A negative probability decrement is assumed for each step, unless there is an anomaly (e.g. an obstacle, a high temperature, an elevated UV), and it is shown that no conflict exists (by the relationship function), where a positive increment is assumed. The value for the increment/decrement was decided upon through the experimental runs.

Finally, TIMM inherits the noise immunity feature from the HIMM method, in that it keeps the certainty values low for cells that were erroneously updated due to noise, crosstalk, and moving objects or other agents. However, this is performed through reccurrent sensing by the multiple agents present in the environment, as opposed to the empty space decrements offered up by both the HIMM and OGF methods. The empty information is accumulated by the aforementioned negative probability assumption. It was decided to use fast short-range infrared measurements, rather than slower, long-range ultrasonic sensor measurements. The noise immunity is due to the multiple agents sensing the cells of the environment, reinforcing the cells' certainty values. At the same time, the certainty values of cells that actually correspond to obstacles and hazards are also kept low, since they have already been explored and detected as having an anomaly. Since the overall goal is to minimize cellular entropy, the method inherently guards against measurement or sensor inaccuracies, as well as noise, crosstalk and other misreadings.

3.2.2 Search Strategy

To direct the search within the environment, the following methods were initially researched as search strategies: simulated annealing and genetic algorithms. These methods were deemed to be too computationally intensive for the undertaken system. Instead, it was decided to use a more embedded system suitable solution, that involved two key points:

High-entropy driven by starting at the current quadrant, and verifying if there is a difference (by a set threshold) between that entropy value and all the other quadrants. If a decision cannot be made (all entropy values are within the threshold), move down the quadtree iteratively, until we have reached the cellular level,

Time constraint driven so as to guard against an agent staying too long in one specific region of the environment.

The agents are thus directed to go the regions with high entropy, where a region is, as described above, a quadrant at any hierarchy of the quadtree.

3.3 Information Fusion

To measure the information content extracted by our sensor fusion technique, it is necessary to measure the mutual dependence of the environmental variables being considered. For this end, **mutual information** [47], or trans-information, is a quantity that is used to identify the amount of information that is shared by 2 or more random variables. Mutual information is mostly used in information theory, where the mutual information of variables X and Y represents the information about X that is shared by Y. The unit of measurement of mutual information is the bit. Mutual information is typically designated by the variable I and is defined as:

$$I(X;\ Y) = H(X) + H(Y) - H(X, Y) \tag{3.18}$$

where,

$H(X)$ is the entropy measure of random variable X;

$H(Y)$ is the entropy measure of random variable Y; and

$H(X, Y)$ is the joint entropy measure of variables X and Y.

Joint entropy measures how much entropy is contained in a joint system of two or more random variables, and like entropy and mutual information, is measured in bits. If each pair of

possible outcomes $(X = x)$ and $(Y = y)$, designated as (x, y), occur with probability $p(x, y)$, then the joint entropy is defined as:

$$H(X,Y) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}(X,Y) \cdot log_2 p_{ij}(X,Y) \tag{3.19}$$

where, $pij(X, Y)$ indicates the joint probability of random variable X being classified in category i and random variable Y being classified in category j, and is defined as:

$$p(x,y) = p(x \cap y) = p(y \mid x) \cdot p(x) \tag{3.20}$$

Information fusion is thus possible by integrating the various entropy landscape maps together to form a consistent and shared view of the environment that can be fed into the virtual world inhabited by the human tele-operators. This is done by considering equation (3.18) and attempting to optimize it through a *maximal mutual information (MMI)* goal-directed approach. The latter implies maximizing the marginal entropies (i.e. *H(X)* and *H(Y)*), which is already performed for each entropy; and the former implies minimizing the joint entropies (i.e. *H(X,Y)*). Using a similar analysis to that performed in section 3.2, and considering cases where there are m possible states to consider for variable X and n possible states to consider for variable Y, while under the constraint that all probabilities must sum to 1, the generalized entropy reduction rule can be defined as:

$$\forall i,j : \mathbb{I} \mid i \in 1..m, j \in 1..n \bullet \frac{\partial H}{\partial P_{ij}} = -\frac{\alpha log_2(P_{ij})}{ln(2)}, \text{where} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij} = 1 \tag{3.21}$$

where, H, here, represents the joint entropy of the two random variables. Hence, it suffices to drive one of the probabilities to either 0 or 1 for the goal to be achieved, that again implies that each tessellation is classified as being in a unique state combination (e.g. the cell is empty and cold). Let us analyze the scenario where the obstacle environment landscape (Ω) is fused with the temperature environment landscape (Ψ). The mutual information of those two variables becomes:

$$I^{\Omega,\Psi}(X;\ Y) = H^{\Omega}(X) + H^{\Psi}(Y) - H^{\Omega,\Psi}(X,Y) \tag{3.22}$$

The goal remains the same for two random variables, and in fact, the generalized rule extends to three random variables, where the mutual information is defined as [48]:

$$I(X;\ Y;\ Z) = H(X)+H(Y)+H(Z)-H(X,Y)-H(X,Z)-H(Y,Z)+H(X,Y,Z) \quad (3.23)$$

where, the joint entropy of 3 random variables is defined as:

$$H(X,Y,Z) = H(X)+H(Y)-H(Z \mid X,Y) = H(X \cup Y \cup Z) \quad (3.24)$$

and more generally, for n random variables, the joint entropy is defined as [49]:

$$H(X_1, X_2, ..., X_n) = \sum_{k=1}^{n} H(X_k \mid X_{k-1}, ..., X_1) \quad (3.25)$$

Our method thus involves multi-sensor information gathering resource management, that handles sensing process-specific and environment-specific parameters. We finally obtain a geometric-indexed layered set of information grids, one layer per environment parameter (such as the ones demonstrated in Figures **??** and later in 6.34), forming a multi-dimensional layered environment map.

3.4 Summary

This chapter presents the agent motion model, the range sensor model and two multi-sensor fusion models. Also presented is a sensing methodology based on minimizing the sensed environment entropy, which is compared to two prevalent methods. The use of a quadtree-based data structure representation of the environment allows for a multi-resolution and multi-dimension view of said environment. The chapter concludes with the presentation of an integration technique based on maximizing the environment's information content, when integrating multiple sensor modalities together.

Chapter 4

Multi-Agent Platform Specifications and Simulation Environment

"When I examine myself and my methods of thought, I come close to the conclusion that the gift of fantasy has meant more to me than my talent for absorbing positive knowledge"
(Albert Einstein)

Agent architectures define the approach that one takes in the building of intelligent systems, including the internal structure and operation of the agent. They are typically divided into logic-based, reactive, belief-desire-intention and layered architectures. However, a more concrete division outlines proactive, reactive and hybrid agent architectures. Proactive agent architectures are intuitive and easily decompose into subsystems; however, they suffer from the problem of calculative rationality and are usually difficult to realize. Reactive agent architectures are simple, economical and computationally tractable; however they suffer from the problem of a short-term view and are usually difficult to implement if they contain many layers. Hybrid agent architectures are either horizontally- or vertically-layered, and combine advantages from both proactive and reactive architectures; however, they suffer from other problems, such as the absence of a semantic or conceptual clarity and the difficulty of inter-layer interactions.

Specification languages are used to provide meanings for common concepts and terms, in a sufficiently well-structured fashion, and must enable alternative designs of models and systems to be presented explicitly, compared and evaluated. This chapter presents and specifies a novel architecture, termed the retroactive agent architecture, that combines the advantages of

both reactive and proactive architectures, but that also resolves their drawbacks. The architecture is specified by the CSP-OZ specification language, after a deep analysis and discussion of candidate specification languages.

4.1 Agent Architecture

During this project, we have been involved in numerous robotics-oriented projects, spanning the perception to actuation spectrum. As can be seen from Figure 4.1, that describes the typical **proactive agent architecture** used for most of the robotic architectures synthesized in the 1980s, and still in use today, there are at least four major functional units that a robotic agent has to execute to respond to environmental events through the manipulation of one or more of its actuators. This model of robotic control has been experimented with, at length, through numerous research projects and experiments at the *Sensing and Modelling Research Lab (SMR-Lab)* of the University of Ottawa. It has shown conditional success in structured environments; however, it is not very reliable in hostile environments. To achieve a level of autonomy capable of existing in an unstructured environment, a different agent architecture has to be utilized.

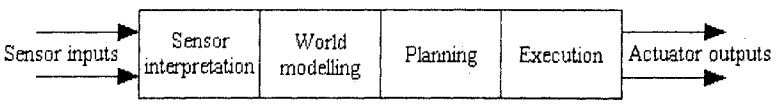


Figure 4.1: Proactive agent architecture

Starting with Brooks' now famous paper [18], it has been shown that the problem of *calculative rationality*, present in the proactive architectures (also known as deliberative or logic-based) described above, where the decision making apparatus will suggest an action that was optimal when the decision making process began, is a major drawback to proactive agent architectures. This problem, along with many others (e.g. knowledge representation complexity, slow search and planning phases, and inherently sequential operation), forced robotics researchers into devising faster, simpler and more economical solutions for autonomous robotic

control. Behavior-based architectures, better known as **reactive agent architectures,** were the result of such endeavors, and have led to a revolution in the design of robotic controllers. Reactive architectures are elegant, robust against failure and computationally tractable. They are based on three key Brooksian topics, mainly *situatedness*, that allows for a timely response to world events since the model consists of a direct connection from perception to actuation, *embodiment*, which is the physical grounding of the robot in the real world giving meaning to the internal symbolic system, and *emergence*, that allows for insect-like intelligence through the interactions of simpler components of the system.

Over time though, it started becoming apparent that reactive architectures were not the generic solution to autonomous robotic control: there were just too many unforeseen drawbacks. First of all, robots needed more information to make a decision due to the lack of internal work models. Second, it was not very clear how these entities can learn from experience, to improve their performance over time. Third, and most important, robots built using this architecture suffered from a short-term view, in that each decision was based on the current state only. Due to these hindrances, researchers began combining the two extremes into 3-layer systems (reactive near the bottom, proactive near the top and an intermediate layer connecting the two), that have been dubbed as hybrid agent architectures.

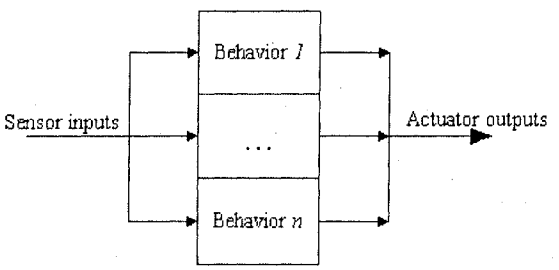


Figure 4.2: Reactive agent architecture

Today hybrid architectures are the most popular class of robotic agent architectures, due to their natural decomposition of functionality into the individual reactive, proactive or abstract layers. Two types of hybrid architectures exist, one with horizontal layering and another with

vertical layering. The former has each layer connected to the sensors and actuators, with a mediator deciding which layer has control over the body at any time, while the latter has a one- or two-pass organizational scheme where information and command both flow upwards (one-pass) or information flows upwards and commands flow downwards (two-pass). Vertically-layered hybrid agent architectures suffer from a loss of flexibility and weak fault-tolerance, whilst horizontally-layered hybrid agent architectures suffer from complex mediation modules in the case of real-world robots.

Let us now take a look at a novel agent architecture that combines both proactive and reactive characteristics. This architecture has been presented in [14], and is called **retroactive**. In a retroactive architecture, an event in the environment occurs, causing momentum in a behavior to increase, eventually causing that behavior to fire and react to the event by executing the behavior's plan. The events of importance are changes in the perceived world attributes that the mind is concerned with, i.e. ones that conflict with its world goals. The reactive characteristic of this architecture involves precisely knowing which behavior to fire through *momentum resolution*, while the proactive characteristic of this architecture involves precisely knowing how to behave through *plan execution*. Figure 4.3, reproduced from [14], depicts the retroactive agent architecture, that augments both reactive and proactive architectures, through its validation block by feeding back into the mind and predicting the next event that will occur, based on previous experiences.

The retroactive agent architecture builds on the importance of synaptic feedback, whence, for example, every sensed event illicits a *reaction* by the agent onto the world to handle the event, a *retroassertion* (i.e. future validation) onto the agent's mind to compartementalize the new or old data, and subsequently a possible *proaction* onto the world again to seek more information about the event. The author has identified six **synaptic acts**, namely reactions, proactions, retroactions, reassertions, proassertions and retroassertions, that if observed from the outside, ascribe intelligent behavior to machines. to better visualize the synaptic acts, the reader is pointed towards Figure 4.4. A retroactive architecture allows an autonomous robot to

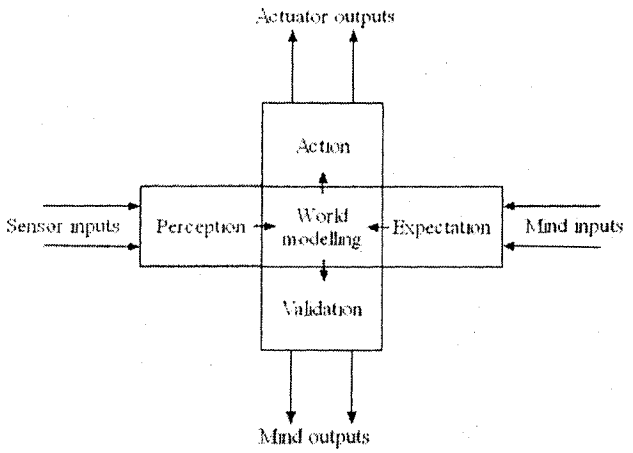


Figure 4.3: Retroactive agent architecture

learn over time, solving the short-term view of reactive systems, and to respond in real-time to world events, solving the calculative rationality of proactive systems.

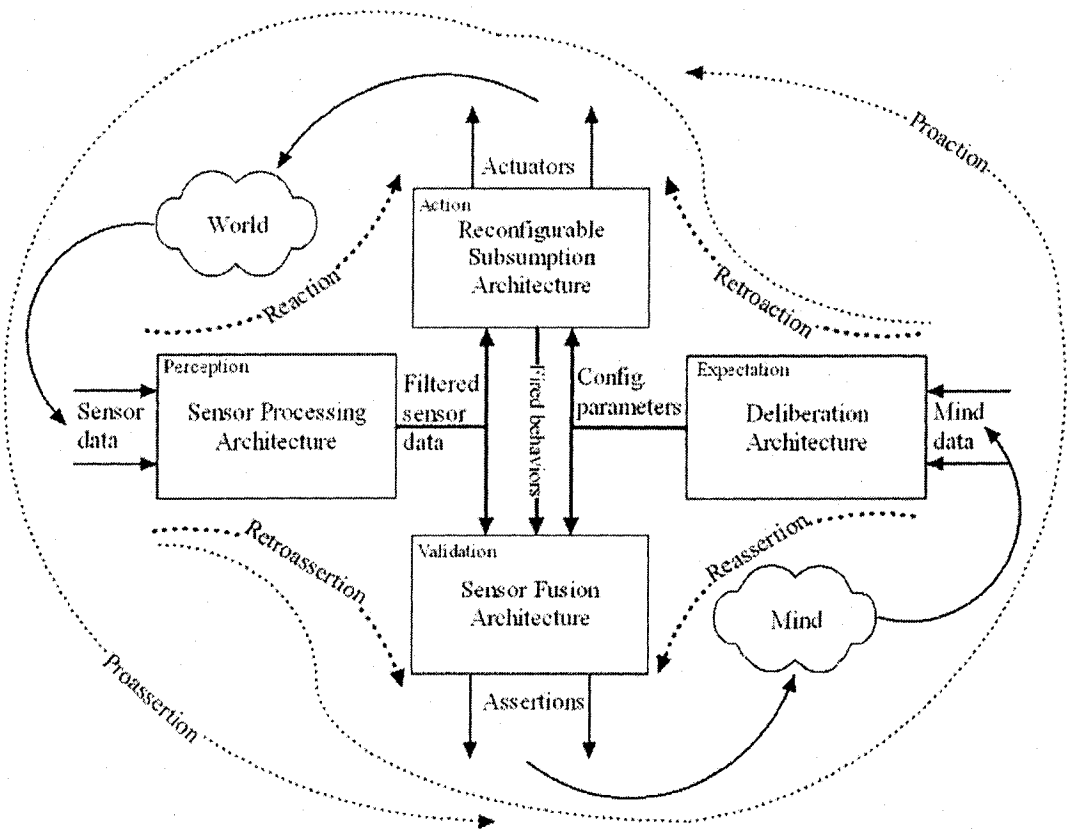


Figure 4.4: Retroactive agent architecture - synaptic acts

4.2 Specification Languages

To better design robust and scaleable multi-agent systems, one needs to formally specify an agent, its attributes, interactions and associations. A *specification language* is defined as a "formal language used in computer science [..] [that is] used during system analysis and design" [50]. It directly contrasts a *programming language*, which is a directly executable formal language used during system realization. Specification languages can be classified as **state-based** (e.g. VDM [51], B [52], Z [53]), **process algebras** (e.g. CCS [54], CSP [55], LOTOS [56]), **temporal logic-based** (e.g. temporal logic [57], active logic [58]), **modal logic-based** (e.g. modal logic [59]), and **statechart-based** (e.g. statecharts [60]). In this thesis, we will concentrate on the Z and CSP formal languages, and their derivatives.

4.2.1 Z

The Z specification language is a state-based formal language mainly used for the specification of data, involving state spaces, state transformations and schema calculus. It is mainly due to Abrial [61], Sufrin [62] and Spivey, and finds its origins in the Programming Research Group at Oxford University. Z is not an executable notation and can actually describe systems for which no execution is possible. It can be used to: (i) specify functionality by modelling the state of a system; (ii) build a model or theory of a subject area; and (iii) specify constraints on certain aspects of the system [63]. Although Z can be used to specify state and behavior, it does not support encapsulation or inheritance of information. Hence, given the definitions of object-based and object-oriented in [64], it is important to realize that Z is *object-based*, as in it supports objects, but *not object-oriented*, since it does not support classes or inheritance.

4.2.1.1 Advantages

The Z language has a few advantages that allow it to be a target language for the design and analysis of multiagent systems, including [65]:

- a vast literary support;
- a vast refinement method support allowing for a systematic flow from specification to implementation;
- a vast animation tool support;
- an increasing utilization in both industry and academia; and
- an extremely expressive language.

4.2.1.2 Disadvantages

However Z is not without disadvantages that may hinder its use as a specification language in the design and analysis of a multiagent system. They include the following:

- No notion of time exists in Z, hence it is difficult to express liveness properties that can be easily be specified in temporal logic;
- Few high-level concepts relevant to agents, such as mentalistic notions (e.g. beliefs, goals, desires), exist in Z, hence forces users to define such notions as part of the specification; and
- No notion of grouping operations on a particular state exists in Z, hence it must be combined with a process to provide the structure for Z specifications.

4.2.1.3 Example: 1-Place Buffer

To illustrate the concepts presented in Z, let us try to model a 1-place buffer capable of holding one natural number.

Buffer

$max : \mathbb{Z}$

$places : \operatorname{seq} \mathbb{N}$

$max = 1$

$\#places \leq max$

Put

$\Delta Buffer$

$x? : \mathbb{N}$

$\#places < max$

$places' = \langle x? \rangle \frown places$

Get

$\Delta Buffer$

$x! : \mathbb{N}$

$\#places > 0$

$x! = head(places)$

$places' = \langle \rangle$

We can see that three schemas are involved, one to define the actual buffer, *Buffer*: it contains a powerset of natural numbers whose length can be at most 1 (indicated by the predicate). The second schema is an operation schema, *Put*, that indicates that the declarations made in the *Buffer* schema will change. A natural number is input and stored in the buffer via the predicate. The final operation schema, *Get*, indicates that the declarations made in the *Buffer* schema will also change: the natural number in the buffer is output and the buffer is cleared.

4.2.2 Object-Z

The Object-Z specification language is an object-oriented extension of the Z specification language mainly used for the specification of complex data systems, involving encapsulation and inheritance of information. It is mainly due to Duke, Rose and Smith [66], and finds its origins in the Software Verification Centre at the University of Queensland, Australia. Object-Z introduces the class schema [67], that encapsulates a single state schema and all operations that may affect its instance variables. The complete Object-Z schema calculus allows for the object-oriented notions of inheritance, object instantiation, object aggregation, object containment and polymorphism; however, as we shall see, only a subset of these operations are required in our framework.

4.2.2.1 Advantages

The language incorporates all the advantages of the Z specification language, and enhances them by adding:

- an object-oriented formalism that includes class, inheritance and polymorphism concepts;
- an inheritance of the refinement methods of the Z specification language allowing for a systematic flow from specification to implementation; and
- a yet to be seen seamless integration with process algebras.

4.2.2.2 Disadvantages

Object-Z suffers from its relative innovation, as a vast body of literature has not been compiled yet, and industrial endeavours using the specification language are still in their infancy. However, a substantial number of academic projects have successfully used Object-Z as a formalism to design and analyze real-time as well as concurrent systems (e.g. [68, 69]). Tool

support for the language is also on the rise as indicated by the various projects undertaken ranging from an Object-Z browser [70], to Object-Z model checking using FDR [71], to Object-Z encoding using Isabelle/HOL [72].

4.2.2.3 Example: 1-Place Buffer

The 1-place buffer example introduced in section 4.2.1.3 is modelled again this time using Object-Z.

Buffer_data

$max : \mathbb{Z}$

$places : \operatorname{seq} \mathbb{N}$

$max = 1$

$\#places \leq max$

Init

$places = \langle\rangle$

Put

$\Delta(places)$

$x? : \mathbb{N}$

$\#places < max$

$places' = \langle x? \rangle \frown places$

Get

$\Delta(places)$

$x! : \mathbb{N}$

$\#places > 0$

$x! = head(places)$

$places' = \langle\rangle$

We see that the schemas have been encapsulated into a class schema, *Buffer_data*, that includes a state schema, that is actually nameless, an initialization schema (*Init*) and two operations schemas (*Put* and *Get*). This allows us to inherit this class to form more complex data structures,

such as n-place buffers.

4.2.3 CSP

The *CSP (Communicating Sequential Processes)* specification language is a process algebra mainly used for the specification of a system as a collection of concurrent processes that communicate over unbuffered channels by synchronizing over events. It is mainly due to Hoare, and also finds its origins in the Programming Research Group at Oxford University. CSP considers processes as self-contained components with an interface through which they interact with the environment. The processes are *compositional* in that if they were combined, they would also exhibit self-containment, but at a group level. CSP also provides the primitives to describe the temporal ordering of these processes, as well as the synchronizing communication between them. We shall see later on the utility of such a feature.

4.2.3.1 Advantages

The language's advantages lie in its conceptual simplicity and syntactic flexibility. Given the latter, CSP nevertheless provides a robust solution to many of the common synchronization problems found in all shapes of parallel systems. Due to its temporal ordering specification, CSP also aids in the description of the dynamic event exchange and process execution, hence the modelling of the *behavior of systems*.

4.2.3.2 Disadvantages

CSP suffers from three major drawbacks:

- Explicit naming of the communicating processes is required;
- No message buffering is included in the language; and
- The lack of a rich language hinders CSP in the description of the data aspects of a system.

The first drawback does not allow for broadcast communications or process discovery, while the second drawback places a burden on the designer to build I/O buffering between communicating processes, as the latter always block after sending messages. The final drawback is resolved by either extending CSP to enrich the language or by combining CSP with an existing formalism that provides such semantic riches. The latter solution is taken in the next section.

4.2.3.3 Example: 1-Place Buffer

The 1-place buffer example introduced in section 4.2.1.3 is modelled again this time using CSP.

$$\alpha(Buffer) = \{Get.n \mid n \in \mathbb{N}\} \cup \{Put.n \mid n \in \mathbb{N}\}$$

$$Buffer \stackrel{c}{=} Put?x : \mathbb{N} \rightarrow Get!x : \mathbb{N} \rightarrow Buffer$$

CSP's syntax is quite trivial and easily understood. The first sentence defines the alphabet to be used in the buffer's declaration, which, in this case, is the set of all the events that interact with the buffer. The second sentence defines the *Buffer* process by saying that an event has to be received on the *Put* channel with a natural number input, before an event can be received on the *Get* channel to extract that natural number: temporal ordering of the buffer is inherently maintained in CSP. Finally, since recursion is allowed in CSP syntax, the process recurses to wait for another *Put* channel input.

4.2.4 CSP-OZ

The CSP-OZ specification language is a combination of CSP and Object-Z, mainly used for the modelling of both state and behavior in concurrent systems. It is mainly due to Fischer and finds its origins in the Correct System Design Group at the University of Oldenburg, Germany. CSP-OZ extends the Object-Z class schema to include an interface definition and a CSP process. As previously mentioned, process algebra languages, such as CSP, are ideal for modelling the interactions between concurrent processes; however, they are not suitable for

modelling complex data structures, that could aid in the specification of the processes themselves [73]. Such a rich data aspect formalism is furnished by state-based languages such as Z and Object-Z. The combination of the two formalisms (i.e. process algebra with state-based languages) cannot always be smoothly accomplished, but in the particular case of CSP and Z/Object-Z, a common semantic model exists between the two, namely the failures-divergences theorem [74], that provides the basis of the semantic integration. Historically, combinations of CSP and Z have been attempted [75, 76], but these efforts have been superceded by the recent combinations of CSP and Object-Z, due to the latter language's implicit semantic integration with CSP: Object-Z contains constructs, namely classes, that are given a failures-divergences semantics identical to that of CSP processes, effectively mapping Object-Z classes to CSP processes and Object-Z operations to CSP events! The reader is referred to [77] for a detailed proof of the failure-divergence semantics of CSP-OZ.

4.2.4.1 Advantages

CSP-OZ is a complex, but very expressive and powerful language. It inherits the advantages of flexibility and behavior specification from CSP, along with the advantages of complex data containment and state management from Object-Z. This combination leads to the true power of the CSP-OZ language: its *ability to model most aspects of a system*, neither relying on a mixture of independent formalisms each meant to handle a specific aspect, nor on a poor formalism of all aspects of a system by a specification language intended for only one such aspect. CSP-OZ is also endowed with an established theory of refinement, an applicable library of tool support and a growing database of literary references.

4.2.4.2 Disadvantages

The main drawback of CSP-OZ is its complexity. As we shall see in the next section, it is neither intuitive nor trivial to specify a system using CSP-OZ, that may have led to the reluctance of designers to apply the language to the specification of systems as evidenced by

the modest number of practical concurrent systems specified using CSP-OZ.

4.2.4.3 Example: 1-Place Buffer

The 1-place buffer example introduced in section 4.2.1.3 is modelled again this time using CSP-OZ.

Buffer_cntl

$Put[x? : \mathbb{N}]$

$Get[x! : \mathbb{N}]$

$main \stackrel{c}{=} Put \rightarrow Get \rightarrow main$

`inherit` *Buffer_data*

`enable_`*Put*

$\#places < max$

`enable_`*Get*

$\#places > 0$

`effect_`$Put = Put$

`effect_`$Get = Get$

The above CSP-OZ class consists of three main parts: the interface, the CSP-part and the Z-part. The interface defines the channels that communicate with this class/process, mainly the *Put* and *Get* channels. As we can see, the *Buffer_cntl* class is a specialization of the *Buffer_data* class, inheriting all of the latter's attributes and methods. This is indicated by the '*inherit Buffer_data*' specification. The CSP-part defines the temporal ordering of the process and thus restricts the behavior of the class (a put must be performed before a get!). Finally, the Z-part provides the guards of operations (through the *enable* keyword) and the corresponding state transitions (through the *effect* keyword). This will be discussed in more detail when the language syntax is

described in more detail.

To complete the specification, and following the integration paradigm laid out in [78], we have to model the entire architecture of the 1-place buffer, which is usually done by combining CSP-OZ classes using CSP operators. The buffer is defined as the *Buffer_cntl* CSP-OZ class, while hiding the set of events that correspond to the *Put* and *Get* channels.

$$Buffer = Buffer\_cntl \setminus \{| Put, Get |\}$$

4.2.5 TCOZ

The *TCOZ (Timed Communicating Object-Z)* specification language is a combination of CSP and Timed CSP [79] mainly used for modeling systems with complex components that may have their own thread of control. Timed CSP is an extension of Hoare's CSP, that introduces the concept of modeling real-time to build on CSP's strengths of process control and communication modeling by adding the ability to represent timeouts, delays and interrupts as well. TCOZ is mainly due to Mahony and Dong [80], and finds its origins in the Defence Science and Technology Organisation and the Commonwealth Science and Industrial Research Organisation both of Australia. TCOZ differs from CSP-OZ and Smith's integration of CSP and Object-Z, in that it identifies Object-Z operations with CSP processes. As well, all the communications must go through the explicitly declared channels. Hence, TCOZ is not as trivial an integration of CSP and Object-Z as CSP-OZ is, extending both base notations with new language constructs. However, TCOZ does have two dinstint advantages over Smith's Object-Z/CSP integration and CSP-OZ, in that the semantic link between Object-Z operations and CSP processes is bidirectional, whilst the semantic link between Object-Z classes and CSP processes in the other two methods is unidirectional [81]. This aids in the modeling of true multi-threaded concurrency. The second distinct advantage is that TCOZ provides language semantics that allow for multi-threading and real-time applications, as they forego the constraint

of operations being atomic, which the other two methods keep. The designers of TCOZ argue that restricting operations to atomic events forces actions to happen at one single point in time.

Due to the added complexity of TCOZ and the domain at hand (multiagent systems), it is better to model the system using CSP-OZ. Multiagent systems are basically systems with a group of simple components with a complex set of concurrent interactions between them, hence a language such as CSP-OZ is better equipped at dealing with these system-level connections than TCOZ.

4.2.6 Other Languages

There have been over the years, numerous attempts to connect state-based languages with their process algebra counterparts to provide designers with a more complete landscape of system specifications. In this section, we will review some of the most prominent alternative methods that have been successfully realized.

4.2.6.1 CSP-OZ-DC

The CSP-OZ-DC specification language is a combination of three well-established and researched methodologies: CSP, Object-Z and Duration Calculus [82]. The language provides the designer with techniques to model process, data and time simultaneously, where a class is defined as $C = (I, P, D, T)$ with interface I, process (CSP) part P, data (Object-Z) part D a timing (DC) part T [83]. The language is thus a very expressive and powerful, but quite complex language. It is due to Olderog and Hoenicke and finds its origins in the Correct System Design Group at the University of Oldenburg, Germany. The language is touted to be able to handle the modelling of complex systems with dynamic behaviors such as "'communication between components, state transformation inside components, and real-time constraints on the communications and state changes"' [83]. Model checking support using FDR for the CSP part, and UPPAAL for the DC part is well-supported. Although a relatively new language, CSP-OZ-DC introduces a complete solution to the dynamics of complex systems, where state

and behavior have to be specified.

4.2.6.2 B, VDM and CSP

The *VDM (Vienna Development Method)* and *B* specification languages are similar state-based languages, mainly used to define systems with complex data aspects. Both methodologies (rather than notations such as Z) possess well-defined refinement notions; however, they lack temporal ordering concepts required to describe dynamic behavior. VDM, an ISO standard [84], has its origins at the IBM Vienna Laboratory in Austria, and contains explicit modularisation facilities, as well as object-oriented extensions (e.g. VDM^{++} [85] and Fresco [86]). B, on the other hand, is due to Abrial [87] and has its origins in the Programming Research Group at the Oxford University (England) Computing Laboratory. B supports data encapsulation and posseses a vast library of tools and support packages. Attempts have been made to combine B and CSP [88, 89] to control the temporal ordering of B operations, as previously discussed in this chapter.

4.2.6.3 Z and LOTOS

The *LOTOS (Language Of Temporal Ordering Specification)* specification language is mainly used in the design of sequential, concurrent and reactive systems. On conception though, LOTOS was intended for the design of OSI (Open Systems Interconnection) systems, that led to its standardization in 1989. It is mainly due to the international standardization effort that required an implementation bias-free formalization for the OSI model. As a result, two international standards were produced: *ESTELLE (Extended Finite State Machine Language)* [90] and LOTOS. The former targetted distributed systems while the latter targetted reactive systems. LOTOS includes a process algebra section based on CSP and CCS. The latter is a process algebra that is very similar to CSP, except that it was designed as a minimal and elegant language, hence with a minimal set of process operators. It is mainly due to Milner and finds its origins at the University of Edinburgh, England.

LOTOS suffers from three major drawbacks: its inability to support distributed systems, its weak expressions and handling of time and its indirect support for object-orientation. These drawbacks have no viable solutions, have figured into our decision of choosing a suitable specification language. It is worth mentioning that to target LOTOS to the design of distributed systems, a standardization initiative has been proposed and accepted where Z and LOTOS are combined for specifying particular views. For more information, the reader is referred to the ODP (Open Distributed Processing) ISO standard [91].

4.2.6.4 Z and Refinement Calculus

The final discussed combination of formal specifications involves the aforementioned Z notation and the refinement calculus [92], which is also a notation that allows incremental refinement of specification to implemenation. The advantage of refinement calculus is that the computational detail that allows for the refinement is built right into the specification. However, the combination of Z and the refinement calculus suffers from major drawbacks including poor timing constructs, operations structures and object-oriented extensions. Moreover, the combination was not intended for the design of distributed or concurrent systems, hence reducing its usability. The reader is referred to [93, 94] if interested in works that have described such a combination.

4.2.7 Comparisons and Discussion

Shown in Table 4.1 is a comparison of all the specification languages prosed in this section, including each's type (i.e. PA if a process algebra, SB if state-based, COMBO if a combination). From this extensive research, we have chosen CSP-OZ as the most suitable specification language for our design and analysis. The major reasons for the decision have been distributed throughout the previous sections but consolidate to:

- CSP-OZ is an expressive and powerful language with a solid refinement method and

tool support due to its Z/Object-Z inheritance;

- CSP-OZ can model most aspects of distributed, concurrent and reactive systems due to its inheritance of CSP and Z/Object-Z, as well as its seamless integration of the two;
- CSP-OZ has a foundational integration paradigm (mentioned in section 4.2.4.3) that designers can follow for a systematic flow.

No language entirely fulfills the requirements set out at the outset of this research; however, CSP-OZ comes the closest to doing so.

Language	Description	Advantages	Disadvantages
B	SB	vast tool and literary support supports data encapsulation refinement to code built into semantics	keyword-based notation proof required to preserve the invariants poor temporal ordering constructs
CCS	PA	minimal and elegant language minimal set of operators process communication easily handled	observation equivalence and congruence [95] lack of a rich set of data structures minimal set of operators (no typo!)
CSP	PA	robust solution to synchronisation problems rich temporal ordering specification modelling of behavior of systems	explicit naming of the communications lack of a message buffering lack of a rich data language
CSP-Z	COMBO	intuitive adaptation from CSP existing model checker can model most aspects of a system	few literary references lack of a tool support poor operations structure
CSP-OZ	COMBO	expressive and powerful language can model most aspects of a system has an applicable tool support	competing techniques of integration complex language syntax and semantics small number of practical references
CSP-OZ-DC	COMBO	combination of well-established specifications real-time constraint modelling well founded Duration Calculus theory	relatively new methodology poor literary library complex syntax and semantics
LOTOS	COMBO	standardized under ISO vast amount of tool and literary references includes a process algebra based on CCS	does not support distributed systems weak expressions of time no direct support for object-orientation
Object-Z	SB	provides object-oriented formalism to Z inherits refinement methods seamless integration with process algebras	relatively new formal language lack of a temporal ordering few mentalistic notions
TCOZ	COMBO	can model most aspects of a system semantic link is bidirectional allows for multi-threading applications	targetted for complex components difficult Object-Z/CSP integration competing techniques of integration
VDM	SB	standardized under ISO supports modularisation facilities object-oriented extensions exist	poor timing primitives lack of support for concurrency poor operations structure
Z	SB	rich data structures vast literary and tool support gaining acceptance in AI community (ISO std.)	poor timing primitives few mentalistic notions poor operations structure
Z + Refinement Calculus	COMBO	computational detail within the specification allows refinement from specification to code well founded Refinement Calculus theory	poor timing primitives poor literary library poor operations structure

Table 4.1: Comparison of specification languages

4.3 Multi-Agent Platform Specifications

This section presents the *multi-agent system (MAS)* specifications formulated using the CSP-OZ specification language. The reader is asked to note the differences between this and Z's formalism to realize the advantages offered by CSP-OZ over Z. Especially of note is the inclusion of temporal behaviour specifications within each defined entity, as well as the separation between data and behaviour aspects, so as to maximize flexibility, modularity and reuse of the specifications. The formalism is called **Multi-Agent Platform Specifications (MAPS)** for two reasons: the name is a decently encompassing one, and the name indicates that the formalism provides a set of maps to guide us through the design of multi-agent systems.

4.3.1 World View

Before we actually begin specifying the system, we have to first introduce our world view. Our world consists of an environment that needs to be monitored by a MAS. The MAS itself is composed of multiple agents that cooperate together towards the achievement of the task at hand. We will first present an example of a design that can be accomplished using our framework. Figure 4.5 demonstrates such a design. Let us not worry about the various components of the example, as these will be formally explained in the successive sections; however, let us note the most important aspect of this figure, and that is the feedback path that occurs between the layers. Notice that the arrows are bidirectional; as will be later explained, a momentum-resolution function is introduced in the feedback path, that causes the agent to switch the state of its mind to handle an important situation that has arisen in the external environment or internal mind.

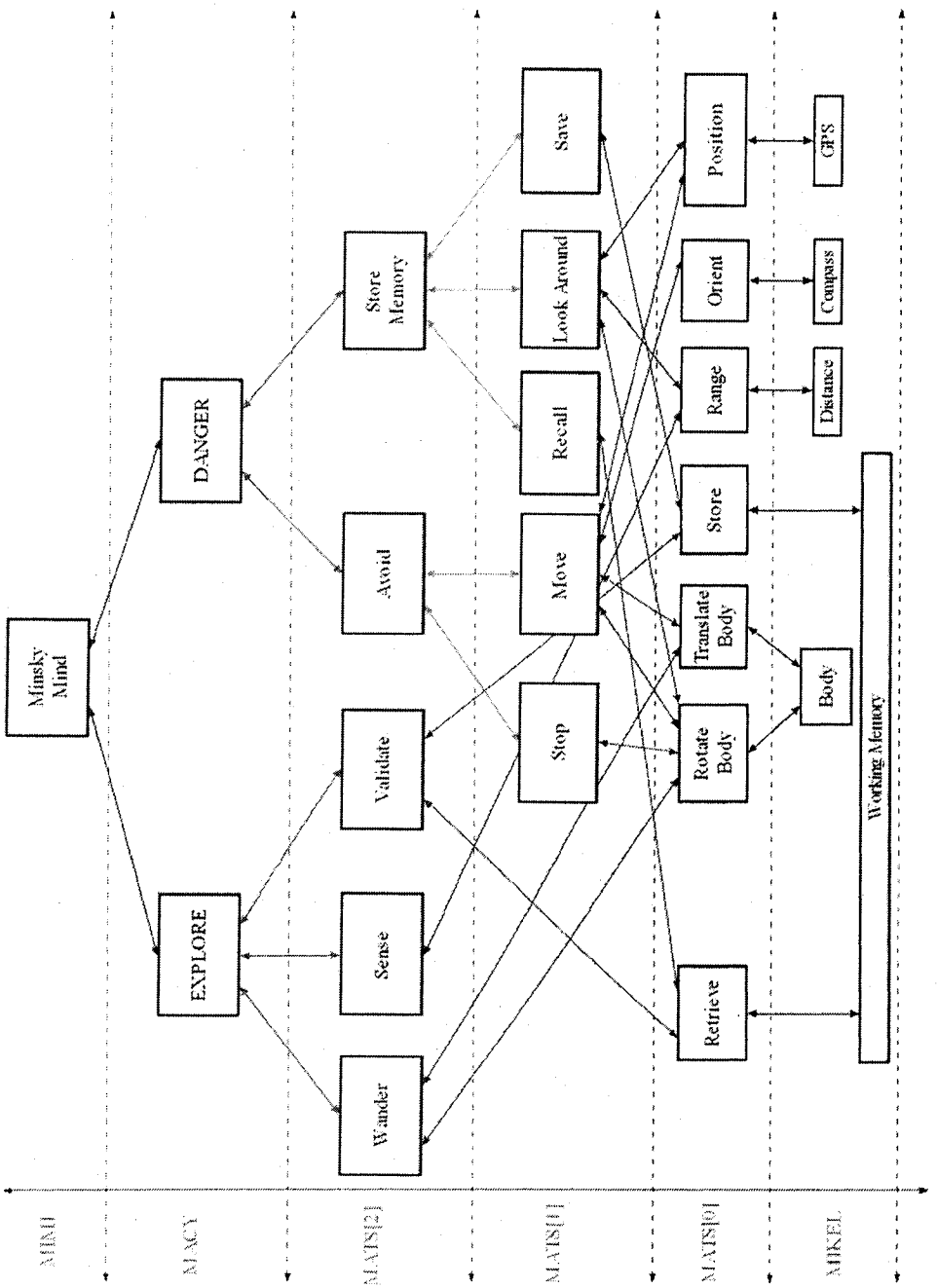


Figure 4.5: MAPS system architecture

4.3.2 Global Specification

To drive the *world view* and synchronize all its associated events and messages, we have chosen to define a free running clock that any agent or time-based computational entity can interrogate for various purposes (e.g. event synchronizations, time stamps and execution delays). This free running clock simulates the continuously increasing world time. The world time is thus specified by the following axiomatic description.

$$worldTime : \mathbb{N}$$

$$worldTime = freeRunningClock$$

All entites within the world are composed of given **attribute triplets**. Each attribute triplet consists of an attribute name, an attribute value (e.g. the tree's "colour is green", the bin's "shape is circular") and an attribute probability (e.g. the probability that the tree's colour is green is 0.75). Attribute triplets could also describe the state of an entity (e.g. the robot's "position is (x,y)" with a probability of occurrence of 0.85). Entities are thus simply a collection of attribute triplets, differing from the SMART framework [65] that disregards the probabilities of occurrence for each attribute.

Definition 4.3.1 An attribute triplet is a sensory world feature composed of an attribute name and a corresponding attribute value, and a corresponding attribute value probability of occurrence.

The set of all such attribute triplets is thus specified.

$$[AttributeName, AttributeValue, AttributeProbability]$$

World attributes are thus ordered triplets that contain an attribute name, an attribute value and an attribute probability. Since all three are given sets as previously specified, a world attribute becomes a relation between the three sets.

Definition 4.3.2 A world attribute is an attribute triplet composed of an attribute name, and corresponding attribute values and probabilities.

The world attribute is thus specified as an infix relation in Z.

$$\_WAttribute\_ : AttributeName \leftrightarrow \text{seq}(AttributeValue \times AttributeProbability)$$

To interact with the rest of the world, each agent will be required to perform a function upon the environment. These functions could be sensory or actuatory in nature, hence perceiving or acting on the environment. The functions are behavioural ones and are characterized by associated figures of merit, such as entropy reduction and response time optimization. Using these figures of merit, it is possible to evaluate the progress of each agent and its overall contribution to resolving the problem at hand.

Definition 4.3.3 A function is an action that can either sense the state of the surrounding environment attributes or act to change that state.

The set of all such functions is thus specified.

$[Function]$

As just mentioned, a *figure of merit* is a measure of success when performing a function. Three major figures of merit are defined: response time, accuracy and entropy. The first measure, response time, describes for how long the function was executing in the context of the current goals. The second measure, accuracy, describes how well the function performed locally in the context of the current goals. Finally, the third measure, entropy, describes how well the function globally performed in the context of the current goals. Response time and entropy tend to be minimized, whilst accuracy tends to be maximized. The three defined figures of merit will be formally specified later on in this section. On that note, this formalism allows for the addition of figures of merit through (i) the extension of the FoM's Z schema (presented below) and (ii) the specification of the new FoM's generic definition.

Definition 4.3.4 A figure of merit is a quantitative measure of the dynamics of a behavioural function.

The figure of merit is thus specified as a Z schema whose state is composed of a non-empty sequence of natural numbers, which currently number three and that must always have a minimum of three. The state-invariants guard against the latter, as well as extracting the figures of merit using the generic definitions of each.

FigureOfMerit

$$figures : \mathrm{seq}_1\ \mathbb{N}$$

$$\#figures \geq 3$$

$$figures\langle 0 \rangle = responseTime\ function$$

$$figures\langle 1 \rangle = accuracy\ function$$

$$figures\langle 2 \rangle = entropy\ attributes$$

Having introduced the figure of merit definition, let us discuss in more detail, the three initial figures of merit: response time, accuracy and entropy. First, let us introduce the probability distribution function related to the attribute value set. It represents the probability of occurrence of the random event associated with the behavioural function that changes that particular attribute value. This function is extracted from the attribute by building a sequence of probabilities, one for each possible state of the attribute. The extraction function is thus specified by the following generic definition.

$$
\begin{array}{|l}
\hline
[attrib] \\
\hline
extractpdf : AttributeName \rightarrow \mathrm{seq}\,\mathbb{N} \\
pdf : \mathrm{seq}\,\mathbb{N} \\
pdfpairs : \mathrm{seq}(AttributeValue \times AttributeProbability) \\
extractprob : AttributeValue \rightarrow AttributeProbability \\
\hline
attrib \subseteq AttributeName \\
pdfpairs = \mathrm{ran}\, attrib \\
\forall i : \mathbb{Z} \mid i = 1..\#pdfpairs \bullet \\
\quad pdf\langle i \rangle = extractprob\ pdfpairs\langle i \rangle \\
extractpdf(attrib) = pdf \\
\hline
\end{array}
$$

Another important function is the attribute differentiation function. It takes a set of ideal and actual attributes and produces a natural number representing the quantified difference of both sets. This number is used in the tabulation of the accuracy of each behavioural function.

$$
\begin{array}{|l}
[idealAttrib, actualAttrib] \\
\hline
attributesNotEqual : \mathbb{Z} \\
runningError : \mathbb{N} \\
extractprob : AttributeValue \rightarrow AttributeProbability \\
differentiateattrib : \mathbb{P}\, WAttribute \rightarrow \mathbb{P}\, WAttribute \rightarrow \mathbb{N} \\
\hline
\#idealAttrib = \#actualAttrib \\
attributesNotEqual = 0 \land runningError = 0.0 \\
\forall a : WAttribute \mid a \in idealAttrib;\ b : WAttribute \mid b \in actualAttrib; \\
valueA : AttributeValue \mid valueA \in \operatorname{ran} a; \\
valueB : AttributeValue \mid valueB \in \operatorname{ran} b \bullet \\
\quad valueA = valueB \Rightarrow \\
\quad\quad runningError = extractprob\ valueA - extractprob\ valueB \\
\quad valueA \neq valueB \Rightarrow \\
\quad\quad attributesNotEqual + one \\
differentiateattrib(idealAttrib, actualAttrib) = attributesNotEqual + runningError \\
\hline
\end{array}
$$

Response time is the measure of computational time taken by a behavioural function in bringing about its planned actions. It is measured time units and indicates the speed of performing functions in the environment. This measurement allows us to fine tune the problem-solving mind agents, by examining the ones that are being overused and attempting to load balance the functionality onto other mind agents. Function response time in terms of a discrete behavioural function X, with start time s being the time that the function was first invoked by the agent, and end time e being the time that the desired view of the environment matches that of the current view, is defined as:

$$R(X) = e - s \tag{4.1}$$

Definition 4.3.5 Response time is the time a behavioural function takes to bring about its

planned behaviour.

The response time figure of merit is thus specified by the following generic description.

$$
\begin{array}{|l}
\hline
[function, funcID] \\
\hline
startTime : \mathbb{N} \\
endTime : \mathbb{N} \\
responseTime : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \\
\hline
startTime = worldTime \\
MIKEL\_Data.write \Rightarrow \\
\quad MIKEL\_Data.write.desiredView? = MIKEL\_Data.localView \Rightarrow \\
\quad\quad endTime = worldTime \wedge \\
\quad\quad responseTime = endTime - startTime \\
MIKEL\_Data.read \Rightarrow \\
\quad MIKEL\_Data.read.currentView! = MIKEL\_Data.localView \Rightarrow \\
\quad\quad endTime = worldTime \wedge \\
\quad\quad responseTime = endTime - startTime \\
\hline
\end{array}
$$

Accuracy is the measure of success present in a behavioural function. Compared to function entropy, which is introduced next, function accuracy possesses mind agent scope, extracted from the mind agent's local views of the environment. Function entropy, on the other hand, possesses environment scope, extracted from the world's attributes. Accuracy will be the measure of how close a function arrives to the desired view of the environment, in the case of an actuatory function, or how close a function arrives to the previous view of the environment, in the case of a sensory function. Function accuracy, as measured in percentage, in terms of a discrete behavioural function X, with m unequal and n equal number of attributes between the desired and actual views, desired discrete probability distribution $P_{desired} = pd_1, ..., pd_n$ and actual discrete probability distribution $P_{actual} = pa_1, ..., pa_n$ (where pd_i is the probability of the

ith desired attribute and pa_j is the probability of the jth actual attribute) is defined as:

$$A(X) = m + \sum_{i=1}^{n} \mid pd(i) - pa(i) \mid \tag{4.2}$$

Definition 4.3.6 Accuracy is the measure of progress in each behavioural function obtained by aggregating the differences between the desired and actual views of the environment in the case of an actuatory function, or the differences between the current and sensed views of the environment in the case of a sensory function.

The accuracy figure of merit is thus specified by the following generic description.

[function]

accuracy : $\mathbb{N}$

MIKEL_Data.capability.function = *function*

MIKEL_Data.write $\Rightarrow$

accuracy = *differentiateattrib MIKEL_Data.write.desiredView?*

MIKEL_Data.localView

MIKEL_Data.read $\Rightarrow$

accuracy = *differentiateattrib MIKEL_Data.read.currentView!*

MIKEL_Data.localView

Entropy, as previously described, is the measure of disorder or randomness present in a signal or random event. In general,

$$H(X) = -\sum_{\underline{x}} p(x_1, x_2, ..., x_n) log_2 p(x_1, x_2, ..., x_n) \tag{4.3}$$

where $\underline{x}$ is a vector representing all of the possible outcomes of the random event X.

Let us now attempt to apply the above to our framework. Table 4.2 shows the mapping that correctly allows us to measure the entropy of our world's random events, that is its behavioural functions. Hence, function entropy becomes:

$$H(F) = -\sum_{\underline{x}} p(\underline{x}) log_2 p(\underline{x}) \tag{4.4}$$

Component	Theory (Shannon)	Practice (MAPS)
Data source	χ	*Environment*
Source alphabet	S	*WAttribute*
Discrete prob. distribution	P	*AttributeProbability*
Number of symbols	2	*#AttributeName*
Random event	X	*Function*

Table 4.2: Mapping of entropy to our framework

where $\underline{x}$ is a vector representing all of the affected attributes of the behavioural function F, $p(\underline{x})$ is the probability of that particular discrete instance of $\underline{x}$, and $H(F)$ is the entropy of the behavioural function F measured in bits.

Definition 4.3.7 Entropy is the measure of information contained in a random event inititated by a behavioural function and perceived by the environment as changes in world attributes.

The entropy time figure of merit is thus specified by the following generic description.

$$
\begin{array}{|l}
[attributes] \\
\hline
b : \mathbb{Z} \\
numAttribs : \mathbb{Z} \\
numStates : \operatorname{seq} \mathbb{Z} \\
pdf : \operatorname{seq} \operatorname{seq} \mathbb{N} \\
entropy : \operatorname{seq} \operatorname{seq} \mathbb{N} \rightarrow \mathbb{Z} \rightarrow \mathbb{N} \\
alphAccuracy : \operatorname{seq} \operatorname{seq} \mathbb{N} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{N} \\
\hline
b = \#AttributeName \\
numAttribs = \#attributes \\
\forall attrib : attributes;\ n : \mathbb{Z} \mid n = 1..numAttribs \bullet \\
\quad numStates\langle n \rangle = \# \operatorname{ran} attrib \\
\quad pdf\langle n \rangle = extractpdf\ attrib \\
entropy = -\sum_{attributes}^{numStates} pdf(attributes) * log_2\ pdf(attributes) \\
alphAccuracy = -\sum_{attributes}^{numStates} pdf(attributes) * log_b\ pdf(attributes) \\
\hline
\end{array}
$$

World functions are thus ordered triplets that contain a function, a powerset of affected attributes and a non-empty sequence of figures of merit, and that can be represented by relations among the three sets. It is important to note that, in the rest of this text, a function is referred to as a world function only when it is intended to point out to the reader the dynamics, that is its attributes or figures of merit, of that particular function. It is therefore implied that both the attributes that a world function affects and the figures of merit it outputs can change over the lifetime of the function.

Definition 4.3.8 A world function is a behavioural function composed of a base function and its associated affected attributes and resulting figures of merit.

The world function is thus specified as a Z schema whose state is composed of the behavioural function, its affected attributes, its resulting figures of merit and a relation that extracts the affected attributes of each function. The state-invariants include the static sizing pre-condition and the attribute extraction.

WFunction

$$
\begin{array}{|l}
\hline
FigureOfMerit \\
function : Function \\
attributes : \mathbb{P}\, WAttribute \\
extractattrib : Function \rightarrow \mathbb{P}\, WAttribute \\
\hline
attributes = extractattrib\, function \\
\hline
\end{array}
$$

The environment itself encompasses the entire set of world attributes. These are in contrast with *mind attributes*, in that the former define externally perceivable attributes, whilst the latter define internally (to the agent) perceivable attributes.

Definition 4.3.9 An environment is the non-empty set of all world attributes.

The environment is thus specified.

$$Environment == \mathbb{P}_1\, WAttribute$$

Every agent in the environment is goal-oriented. Two types of goals are defined: world goals and mind goals (described in the next specification). World goals describe a specific state of affairs [65] that is to be achieved in the world by the agent who has been ascribed this goal or has chosen its undertaking autonomously. The process by which these world goals are to be achieved will be explored later on through the specification of the agents themselves.

Definition 4.3.10 A world goal is a target state of affairs in the world described as a non-empty set of world attributes.

The world goal is thus specified.

$$WGoal == \mathbb{P}_1\ WAttribute$$

Mind goals are internal states of affairs that the agent contains or acquires to organize its mind and problem-resolution faculties. Each mind goal is composed of a sequence of high-level *mind agents*, the latter form the learning and adaptive mechanisms of the agents and will be explored in more detail later. Since learning and adaptation are not features of the basic agent, mind goals thus become well-known sequences of problem-solving tasks.

Definition 4.3.11 A mind goal is a target state of affairs within the agent's mind described as a non-empty sequence of mind agents.

The mind goal is thus specified.

$$MGoal == \mathrm{seq}_1\ MATS\_[L]$$

4.3.3 Agent DNA

For the cohesive development of mind agents, it is necessary to lay a common foundation that can synthesize all the agents. To that effect, it becomes advantageous for us to learn from how living matter functions in the real world. Every living entity possesses a unique signature that defines that entity's communal adaptations and ongoing evolutionary changes to the

all-encompassing world. The signature defines the best-chance of survival for that particular species in that specific world. Such a signature is known as the entity's **deoxyribonucleic acid (DNA)**.

As is well-known about DNA strands, they are composed of pairs of chromosomes, which themselves are composed of **alleles**. Alleles are encoded traits that are either turned on or off according to the evolutionary process. For example, if a bird species consists of birds with blue eyes and others with brown eyes, and the blue-eyed birds are being blinded by the current environmental conditions, then through the survival-of-the-fittest, mostly brown-eyed birds will remain, as the blue-eyed kind will die off (probably due to the loss of the imperative sense of sight). That said, the trait that characterizes brown eyes in that bird species will be more and more defined, whilst the blue eyes trait will be more and more suppressed. The previous discussion tends to demonstrate that alleles could be represented by a rational number between 0 and 1, the value indicating the degree of promotion or suppression that trait is experiencing in that instantaneous moment in evolutionary time. Due to its simplicity in digital representation, manipulation and understanding, we have decided to represent alleles as digital on-off switches, implying that traits are either present or not.

Definition 4.3.12 An allele is a digital trait of a mind agent that is either present in the agent or not.

The allele is thus specified.

$$\begin{aligned} \mathit{Allelle} ::= & \ 0 \\ | & \ 1 \end{aligned}$$

As aforementioned, mind attributes are defined as being internal to the agent, describing perceivable features of this agent (e.g. the agent's "eye colour is green" or the agent's "number of ears is three" as well as the agent's "working memory is large" and the agent's "internal temperature is low"). Henceforth, these attributes are specified by the actual alleles in the DNA strands, and consist of an augmentation of the sets specified by the *WAttribute* relation.

Definition 4.3.13 A mind attribute is an attribute triplet composed of an attribute name and a corresponding attribute value and probability; however, the latter two are extracted from the allele encoding defining that particular attribute, rather than a given set of attribute values.

The mind attribute is thus specified as an axiomatic description extending the *WAttribute* relation by including the attribute triplets defined by the actual alleles within the agent.

$$
\begin{array}{|l}
alleles : \text{seq}_1\ Allele \\
extractname : \text{seq}_1\ Allele \rightarrow AttributeName \\
extractvalue : \text{seq}_1\ Allele \rightarrow AttributeValue \\
extractprob : \text{seq}_1\ Allele \rightarrow AttributeProbability \\
MAttribute == WAttribute \cup \{(extractname\ alleles) \mapsto \\
\quad ((extractvalue\ alleles) \times (extractprob\ alleles))\}
\end{array}
$$

We now try to define some global variables that can be included in any of the lower-level specifications. These currently include the maximum number of alleles that a chromosome may contain (*maxAllele*), the maximum number of chromosomes that a DNA strand may contain (*maxChromosome*) and the maximum number of internal mind agent layers that a mind agent can contain (*maxLayer*). As an example, the globals are specified as follows.

$$
\begin{array}{|l}
maxAllele : \mathbb{N} \\
maxChromosome : \mathbb{N} \\
maxLayer : \mathbb{N} \\
\hline
maxAllele = 20 \\
maxChromosome = 7 \\
maxLayer = 3
\end{array}
$$

We shall soon see that our mind agents are cellular organisms that behave at the micro-level (i.e. the cells) as well as a macro-level (i.e. the agent) through the hierarchical interconnection of these foundational structures. The cells may have different types, as in living organisms:

sight, hearing, touch, taste, skin, heart and so on. For our practical and computational purposes, we will restrict the types of these cells to the ones specified hereafter.

$$
\begin{aligned}
\mathit{CellType} ::= \; & \mathit{Vision} \\
| \; & \mathit{Hearing} \\
| \; & \mathit{Smell} \\
| \; & \mathit{Body} \\
| \; & \mathit{WorkingMemory} \\
| \; & \mathit{HippocampalMemory}
\end{aligned}
$$

Having introduced the traits, we now turn our attention to the chromosomes that embody those alleles. The chromosomes can define the vision, hearing, smell, body, mind features of an agent, as well as the physical characteristics that the agent is possesses. The chromosomes are divided into groups of encoded alleles, with each group describing a character or a feature of the agent, and each sequence of encoded alleles defining the value of that physical or mental feature. The number of alleles per chromosome is variable, but is upper-bounded by the global variable *maxAllele*, as aforementioned.

Definition 4.3.14 A chromosome is a character- or feature-defining structure described as a non-empty sequence of alleles.

The chromosome is thus specified as a Z schema whose state is composed of the non-empty sequence of alleles and whose state-invariants include the non-emptiness and boundary sizing pre-conditions.

Chromosome

$$
\begin{aligned}
& \mathit{alleles} : \mathrm{seq}_1 \mathit{Allele} \\
\hline
& \mathit{alleles} \neq \langle\rangle \\
& \#\mathit{alleles} \leq \mathit{maxAllele}
\end{aligned}
$$

We have already introduced the concept of DNA strands, so we will restrict our discussion here to its specification. The DNA strand, that is at the heart of every cell of the computational organism, is composed of a fixed number of chromosomes. Every DNA strand contains the same chromosomes as every other DNA strand. This allows for a few advantages: it is easier to model another agent's mind by knowing/guessing its DNA strand, it is simpler to reproduce or repair an agent due to the uniformity of its DNA strands and it is easier to manage an organism whose cells all inherit the same features.

Definition 4.3.15 A DNA strand is the complete signature defining the best-chance of survival for a particular species in a specific world.

The DNA strand is thus specified as a Z schema whose state is composed of the non-empty set of chromosomes and whose state-invariants include the non-emptiness and static sizing pre-conditions.

MYDNA

$$chromosomes : \mathbb{P}_1\ Chromosome$$

$$chromosomes \neq \{\}$$

$$\#chromosomes = maxChromosome$$

The final specification of this section is that of a cell. We have already defined the cell's type, but not the cell itself, which is composed of a cell type as well as a world function. For example, a distance cell's function is to provide range information to the organism, while a vision cell's functions include image capture and primitive recognition. The cell is specified as a Z schema with a cell type and a non-empty set of world functions, the latter extracted from the original given set of functions.

Cell

$$type : CellType$$

$$fcn : \mathbb{P}_1\ WFunction$$

4.3.4 Mind Cell

The mind agent is composed of interconnected **mind cells**. These mind cells provide the agent's temporal continuity, in that they are always sensing if they are captor (sensor) cells. On the other hand, the cells provide the agent's effector capabilities if they are actuator cells. Common-type cells group together to form a better sensor (e.g. more distance cells gives the agent a more accurate ranging sensor), and each cell contains a local view of the environment. As small as they may be, these local views are integral for the consolidation of information synthesized by the agent about its surrounding (and perceivable) environment.

4.3.4.1 CSP-OZ Specification

The specification of the mind cell will be done according to the integration paradigm proposed in [78]. It specifies four layers that completely define the system:

Layer 1 : Standard Z semantics definig the data types. This layer contains all the Z axiomatic descriptions, data type definitions and the sort.

Layer 2 : Failure-divergence semantics defining the data states and transformation. This layer contains pure Object-Z classes describing the state schema, attributes of classes and the method schemas.

Layer 3 : Standard CSP-OZ semantics defining the processes. This layer contains the CSP behavior definitions integrated within an Object-Z class.

Layer 4 : Standard CSP semantics defining the architecture. This layer contains the final instantiations of classes into objects and their composition via CSP operators.

It also follows from the previous discussion that layer 2 corresponds to the data aspects of the entity being specified, whilst layer 3 corresponds to the entity's behavior aspects. This separation is quite advantageous to enhance the modularity and reuse of the specification [96].

Layer 1 has already been laid out in the previous sections. Here, we shall call layer 2 specifications *xyz_Data*, and layer 3 specifications *xyz_Control*, where *xyz* is the entity's name.

The mind cell's data specification thus consists of the generic DNA strand contained in every cell, and a specific DNA strand that describes the cell's capabilities and attributes that is extracted by inhibiting and exciting appropriate alleles. Each cell has a function described by its *capability* and initialized in the behaviour specification by a *capability resolution function (crf)*. The latter takes a particular world function and a non-empty set of chromosomes as inputs and outputs whether that function is present in this specific cell (e.g. does the cell possess a hearing function ?). Each cell also has a static location in the body of the agent, described by the 2-dimensional *location* sequence of world attributes. The cell's state consists of the aforementioned local view that is specified as a powerset of world attributes.

The mind cell contains four method schemas: *sense, read, act* and *write*. The *sense* and *act* schemas capture information and effect the environment, respectively, while the *read* and *write* schemas return readings or receive desired actions from the higher-level mind entities (in this case mind agents of type 0).

MIKEL_Data

$dna : MYDNA$

$celldna : \mathbb{P}_1\ Chromosome$

$attribs : \mathbb{P}\ MAttribute$

$capability : WFunction$

$crf : WFunction \rightarrow \mathbb{P}_1\ Chromosome \rightarrow WFunction$

$location : \text{seq}_1\ WAttribute$

$\#location = 3$

$localView : \mathbb{P}\ WAttribute$

sense

$\Delta(localView)$

$localView' = capability.function$

read

$currentView! : \mathbb{P}\ WAttribute$

$currentView! = localView$

act

$\varnothing$

$capability.function\ localView$

write

$\Delta(localView)$

$desiredView? : \mathbb{P}\ WAttribute$

$localView' = desiredView?$

The mind cell's behaviour specification consists of the interface of the four method schemas, the CSP behaviour of the class, the initialization schema, and the enable/effect operations. The temporal behaviour of a mind cell is quite simple: for captor cells, interleave the read and sense operations; for actuator cells, interleave the write and act operations. The process is continuously repeated. The control class also inherits the data specification to have access to all the class attributes and schemas. Appropriate enable (i.e. guard) and effect (i.e. state transition) operations ensure the correct behaviour of the class. The reader may also notice that the specification receives a *Cell* parameter as an input that defines the type of cell this is.

MIKEL[*Cell*]_*Control*

sense[]

read[*currentView*! : $\mathbb{P}$ *WAttribute*]

act[]

write[*desiredView*? : $\mathbb{P}$ *WAttribute*]

$main \stackrel{c}{=} ((read|||sense) \,\Box\, (write|||act)) \rightarrow main)$

inherit *MIKEL_Data*

Init

$celldna = (inhibit) \wedge (excite)$

$attribs = \langle\,\rangle$

$capability = crf\ Cell.fcn\ celldna$

$location = \langle xyz \rangle$

$localView = \langle\,\rangle$

enable_*sense*

$Cell.type = (Vision \vee Distance \vee WorkingMemory \vee HippocampalMemory)$

enable_*read*

$Cell.type = (Vision \vee Distance \vee WorkingMemory \vee HippocampalMemory)$

enable_*act*

$Cell.type = (Touch \vee WorkingMemory \vee HippocampalMemory)$

enable_*write*

$Cell.type = (Touch \vee WorkingMemory \vee HippocampalMemory)$

effect_*sense* = *sense*

effect_*read* = *read*

effect_*act* = *act*

effect_*write* = *write*

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the mind cell. The former is the empty set as the sensing and acting operations are bound to the specific function of the cell. The mind cell, itself, is specified as its control synchronized with the environment on the *sense* and *act* channels.

$$\alpha(MIKEL[Cell]) = \{\}$$

$$MIKEL[Cell] = ((\mathbb{P}\, Environment) \underset{sense,act}{\parallel} MIKEL[Cell]\_Control) \setminus \{| sense, act |\}$$

4.3.5 Mind Agent (Type 0)

The next hierarchical mind entity is the **mind agent of type 0 (MATS_0)**. This structure represents the consolidated capabilities of the complete mind agent. They are the lowest-level mind agents, hence do not possess goals or plans, but do form the faculties of the agent, including access to the body of the agent. We can think of these type 0 mind agents as the interfaces to the groups of mind cells. It is thus implied that this interface must somehow fuse the information that is begotten from all the various mind cells, to present the higher level mind agents with a consistent view of the world. Type 1 mind agents will query these type 0 mind agents for access to the sensor or actuator sites of the agent.

4.3.5.1 CSP-OZ Specification

Type 0 mind agent's data specification starts with upper-bounding the number of cells that could be grouped together to form this interface. The aforementioned fusion function is also described as taking a sequence of a powerset of world attributes and returning the fused powerset of world attributes. The state of the class is composed of a sequence of local views, one for each attached mind cell, and a direction (0 if this is a sensor and 1 if it is an actuator).

The type 0 mind agent contains four method schemas: *read, write, act* and *retroact*. The *read* and *write* schemas receive a parameter (m) as an input defining the interacting cell. Thus,

if reading from the third mind cell, an event occurs on the *read[3]* channel, while if writing to the fifth mind cell, an event occurs on the *write[5]* channel. The *act* schema provides type 1 mind agents with access to this mind agent through the sensory fusion of all mind cell data or through the uniform desired views of all actuating mind cells. Finally, the *retroact* schema provides higher level mind agents with a consolidated view of the immediate environment.

MATS_0_Data

$maxMIKELS : \mathbb{N}$

$fuse : \text{seq}(\mathbb{P}\, WAttribute) \rightarrow \mathbb{P}\, WAttribute$

$localView : \text{seq}(\mathbb{P}\, WAttribute)$

$dir : 0..1$

---

read[m]

$\Delta(localView)$

$currentView? : \mathbb{P}\, WAttribute$

$localView\langle m\rangle' = currentView?$

---

write[m]

$desiredView! : \mathbb{P}\, WAttribute$

$desiredView! = localView\langle m\rangle$

---

act

$\Delta(localView)$

$desiredView? : \mathbb{P}\, WAttribute$

$actualView! : \mathbb{P}\, WAttribute$

$dir = zero \Rightarrow desiredView? = \{\} \land \forall i : \mathbb{N} \mid i \in \{1..maxMIKELS\} \bullet$
$read[i] \land actualView! = fuse\ localView'$

$dir = one \Rightarrow actualView! = \{\} \land \forall i : \mathbb{N} \mid i \in \{1..maxMIKELS\} \bullet$
$localView\langle i\rangle = desiredView? \land write[i]$

---

retroact

$currentView! : \mathbb{P}\, WAttribute$

$currentView! = fuse\ localView$

The type 0 mind agent's behaviour specification is similar to that of the mind cell's. The four interface channels are thus defined, along with the temporal behaviour of the class that reads as follows: a type 1 mind agent will ask this type 0 mind agent to *act* accordingly, hence the latter will either *read* all channels and retroact (sensor capability) or *write* to all channels and retroact (actuator capability). The process is continuously repeated. It is worth noting here that initializing the direction of the type 0 mind agents is a responsibility of the complete mind agent when instantiating these lower level entities. Finally, the reader is to note that the *read* channel is only enabled for sensors, while the *write* channel is only enabled for actuators.

MATS_0_Control

$read[m][currentView? : \mathbb{P}\ WAttribute]$

$write[m][desiredView! : \mathbb{P}\ WAttribute]$

$act[desiredView? : \mathbb{P}\ WAttribute, actualView! : \mathbb{P}\ WAttribute]$

$retroact[currentView! : \mathbb{P}\ WAttribute]$

$$main \stackrel{c}{=} act \rightarrow (read[m] \rightarrow retroact \rightarrow main)$$

$$\Box$$

$$(write[m] \rightarrow retroact \rightarrow main)$$

inherit *MATS_0_Data*

Init

$maxMIKELS = 50$

$localView = \langle\,\rangle$

enable_*read*[*m*]

$dir = zero$

enable_*write*[*m*]

$dir = one \wedge localView \neq \langle\,\rangle$

effect_*read*[*m*] = *read*[*m*]

effect_*write*[*m*] = *write*[*m*]

effect_*act* = *act*

effect_*retroact* = *retroact*

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the type 0 mind agent. The former is the union of the sets composed of the *read* and *write* channel events. The type 0 mind agent, itself, is specified as its control

class synchronized with all the mind cells that it is attached to on the *read* and *write* channels.

$$\alpha(MATS\_0) = \{read.cv \mid cv \in \mathbb{P}\, WAttribute\} \cup \{write.cv \mid cv \in \mathbb{P}\, WAttribute\}$$

$$MATS\_0 = ((\mathbb{P}\, MIKEL[Cell]) \underset{read,write}{\parallel} MATS\_0\_Control) \setminus \{\mid read, write \mid\}$$

4.3.6 Mind Agent (Type 1)

The next hierarchical mind entity is the **mind agent of type 1 (MATS_1)**. This structure represents the first look at the agents of the mind that are well described in Minksy's "Society of Mind" [97]. *MATS_1* modules provide an increasing abstraction of the mind by provisioning the conduit between type 0 mind agents and higher level mind agents. As we shall see, this is quite an important feature of learning and adaptive agents, as layers upon layers of mind agents are built on top of each other, and when one layer becomes hardened, it takes on the form of common sense, whence the agent does not proceed through a time-consuming problem-resolving trace through the mind, but instead calls upon the time-tested and proven common sense layers to automatically respond or react to known or predicted changes in the environment.

4.3.6.1 CSP-OZ Specification

Type 1 mind agent's data specification starts with ceiling the number of *MATS_0* modules that could be interconnected to form this abstract interface. A *conflict resolution function (crf)* is also defined as receiving the response time and the goal satisfaction result as inputs and producing a numerical value called *conflicteffect* that represents the conflicting effect of not satisfying the goal at hand as time passes by (better described as type 1 mind agent's frustration level). The goal that this mind agent has to satisfy is described as part of the state of this class. The goal is a mind goal, hence is a sequence of lower level mind agents, that if executed properly produce the overall effect of the current mind agent. Other state variables include the current executing *MATS_0* and its corresponding sequence number, as well as the current and

goal views of the environment.

The type 1 mind agent contains three method schemas: *act, engage* and *model.* The *act* schema receives two parameter (*m* and *D*) as inputs defining the MATS_0 that is being communicated with and the direction of communication, respectively. The *engage* schema describes events that allow higher level mind agents to engage the current mind agent into action, while passing it a goal view to be achieved in the environment. Finally, the *model* schema updates the mind agent's running goal, as well as its conflict effect.

MATS_1_Data

$maxMATS : \mathbb{N}$

$crf : \mathbb{N} \rightarrow \mathbb{B} \rightarrow \mathbb{Z}$

$goal : MGoal, conflicteffect : \mathbb{Z}$

$curMATS : MATS\_0, numMATS : \mathbb{N}$

$currentView : \mathbb{P}\, WAttribute, goalView : \mathbb{P}\, WAttribute$

---

act$[m, D]$

$\Delta(currentView)$

$desiredView! : \mathbb{P}\, WAttribute$

$actualView? : \mathbb{P}\, WAttribute$

$D = zero \Rightarrow desiredView! = \{\} \land currentView = actualView?$

$D = one \Rightarrow actualView? = \{\} \land desiredView! = goalView$

---

engage

$\Delta(goalView)$

$goalView? : \mathbb{P}\, WAttribute$

$goalView' = goalView?$

---

model

$\Delta(numMATS, curMATS, conflicteffect)$

$numMATS = \#goal \Rightarrow numMATS' = one$

$numMATS \neq \#goal \Rightarrow numMATS' = numMATS + one$

$curMATS' = goal(numMATS)$

$conflicteffect' = crf\ exectime\ (satisfy\ goal)$

The type 1 mind agent's behaviour specification is similar to that of the mind cell's. The three interface channels are thus defined, along with the temporal behaviour of the class that reads as follows: a type 2 mind agent will *engage* this type 1 mind agent to achieve its inherent mind goal, by repeatedly acting and modeling the environment, until there is a minimal conflict effect. The process is continuously repeated. It is also important to realize that initializing the mind goal of this mind agent is the responsibilty of the entire mind agent. Another important observation is the *difference-engine* realization of the modeling behaviour of this mind agent. A difference-engine is a process inside a machine that gives the impression of goal-directedness or persistence, that was devised in the 1950s by Allen Newell, C.J. Shaw and Herbert A. Simon. Originally called *general problem solvers*, these processes possess the following characteristics [97]:

- A difference-engine must contain a description of a "desired" situation.
- It must have subagents that fire based on perturbations between the desired and actual situations.
- Each subagent must act in a fashion so as to minimize its firing parameters.

Given the previous description, it becomes easy to see that the "desired" situation is actually the *goalView* state variable, the firing subagents are the *MATS_0* modules that fire (i.e. are enabled) when differences exist between the desired (*goalView*) and actual (*currentView*) situations, and the actions of these subagents are defined in such a way to attempt to minimize the differences between the firing parameters (i.e. *act* to achieve the *goalView*). That said, each type 1 mind agent thus contains a difference-engine.

MATS_1_Control

$act[desiredView! : \mathbb{P}\, WAttribute, actualView! : \mathbb{P}\, WAttribute]$

$engage[goalView? : \mathbb{P}\, WAttribute]$

$main \stackrel{c}{=} engage \rightarrow (act[numMATS, curMATS.dir] \rightarrow model) \rightarrow main$

`inherit` *MATS_1_Data*

`Init`

$maxMATS = 100$

$conflicteffect = zero$

$curMATS = head\ goal$

$numMATS = one$

$currentView = \{\}$

$goalView = \{\}$

`enable_`*act*$[m, D]$

`enable_`*engage*

`enable_`*model*

$goalView = \{\} \lor currentView \neq goalView$

`effect_`*act*$[m, D] = act[m, D]$

`effect_`*engage* $= engage$

`effect_`*model* $= model$

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the type 1 mind agent. The former is the set composed of the *act* channel events. The type 1 mind agent, itself, is specified as its control class synchronized with

all the type 0 mind agents that it is attached to on the *act* channel.

$$\alpha(MATS\_1) = \{act.cv \mid cv \in \mathbb{P}\, WAttribute\}$$

$$MATS\_1 = ((\mathbb{P}\, MATS\_0) \underset{act}{\parallel} MATS\_1\_Control) \setminus \{\mid act \mid\}$$

4.3.7 Mind Agent (Type L)

The next hierarchical mind entity is the **mind agent of type L (MATS_[L])**. This structure represents the majority of Minksy's agents of the mind, that together form the society of the mind ©. *MATS_[L]* modules are similar to their lower level counterparts, with one major difference: instead of being connected to mind agent primitives (of type 0), these mind agents are connected below and above their layer to other mind agents, hence it becomes important to discuss the number of layers of mind agents within the agent itself. That number is upper-bounded by the aforespecified *maxLayer*. An important characteristic of a learning agent is the ability of creating and organizing new layers of mind agents that can build on the lower level mind agent layers. Now, let us assume the layers are static and do not grow or diminish.

4.3.7.1 CSP-OZ Specification

Type L mind agent's data specification starts with ceiling the number of *MATS_1* modules that could be interconnected to form this abstract interface. As in the type 1 mind agents, a *conflict resolution function (crf)* is also defined as receiving the response time and the goal satisfaction result **as well as the conflict effects of all the lower level connected type 1 mind agents** as inputs and producing a numerical value called *conflicteffect* that represents the conflicting effect of not satisfying the goal at hand as time passes by with the presence of conflicts in the layers below (better described as type L mind agent's frustration level). The goal that this mind agent has to satisfy is described as part of the state of this class. The goal is a mind goal, hence is a sequence of lower level mind agents, that if executed properly produce the overall

effect of the current mind agent. Other state variables include the current executing *MATS_1* and its corresponding sequence number, as well as the current and goal views of the environment.

The type L mind agent contains three method schemas: *engage[m], engage* and *model*. The *engage[m]* schema receives a parameter (*m*) as an input defining the MATS_1 that is being communicated with. The *engage* schema describes events that allow higher level mind agents to engage the current mind agent into action, while passing it a goal view to be achieved in the environment. Finally, the *model* schema updates the mind agent's running goal, as well as its conflict effect.

MATS_L_Data

$maxMATS : \mathbb{N}$

$crf : \mathbb{N} \rightarrow \mathbb{B} \rightarrow \text{seq}\,\mathbb{Z} \rightarrow \mathbb{Z}$

$goal : MGoal$

$conflicteffect : \mathbb{Z}$

$curMATS : MATS\_[L]$

$numMATS : \mathbb{N}$

$currentView : \mathbb{P}\, WAttribute$

$goalView : \mathbb{P}\, WAttribute$

---

engage[*m*]

$goalView! : \mathbb{P}\, WAttribute$

$goalView! = goalView$

---

engage

$\Delta(goalView)$

$goalView? : \mathbb{P}\, WAttribute$

$goalView' = goalView?$

---

model

$\Delta(numMATS, curMATS, conflicteffect)$

$numMATS = \#goal \Rightarrow numMATS' = one$

$numMATS \neq \#goal \Rightarrow numMATS' = numMATS + one$

$curMATS' = goal\langle numMATS\rangle$

$conflicteffect' = crf\ exectime\ (satisfy\ goal)$

$(seq(\forall\, mats : MATS\_[L-1] \mid mats \in goal \bullet mats.conflicteffect))$

The type L mind agent's behaviour specification is similar to that of the mind cell's. The three interface channels are thus defined, along with the temporal behaviour of the class that reads as follows: a type $L+1$ mind agent will *engage* this type L mind agent to achieve its inherent mind goal, by repeatedly engaging and modeling lower level mind agents, until there is a minimal conflict effect. The process is continuously repeated. It is also important to realize, as in all type L mind agents that initializing the mind goal of this mind agent is the responsibilty of the entire mind agent. The boundary conditions on the layer number are specified in the guard operations of the schemas (i.e. the enable schemas).

MATS_[L]_Control

$engage[m][goalView! : \mathbb{P}\ WAttribute]$

$engage[goalView? : \mathbb{P}\ WAttribute]$

$main \stackrel{c}{=} engage \rightarrow (engage[numMATS] \rightarrow model) \rightarrow main$

`inherit` *MATS_L_Data*

`Init`

$maxMATS = 100$

$conflicteffect = zero$

$curMATS = head\ goal$

$numMATS = one$

$currentView = \{\}$

$goalView = \{\}$

`enable_`*engage*$[m]$

$L > one \wedge L < maxLayer$

`enable_`*engage*

$L > one \wedge L < maxLayer$

`enable_`*model*

$L > one \wedge L < maxLayer$

`effect_`*engage*$[m] = engage[m]$

`effect_`*engage* $= engage$

`effect_`*model* $= model$

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the type L mind agent. The former is the set composed of the *engage*

channel events. The type L mind agent, itself, is specified as its control class synchronized with all the type L-1 mind agents that it is attached to on the various *engage* channels.

$$\alpha(MATS\_[L]) = \{engage.cv \mid cv \in \mathbb{P}\, WAttribute\}$$

$$MATS\_[L] = ((\mathbb{P}\, MATS\_[L-1] ||| \mathbb{P}\, MATS\_1) \underset{engage}{\|} MATS\_[L]\_Control) \setminus \{| \, engage \, |\}$$

4.3.8 Mind Agency

The next hierarchical mind entity is the **mind agency (MACY)**. This structure represents the behaviours that the entire mind agent can be in. *MACY* modules are different than lower level mind agents, as they contain a mind goal, as well as a world goal. These world goals are states of affairs of world attributes that the agent is expected to bring about through effective interaction with the environment. This is accomplished by the mind goal, that interleaves the operations of the lower level type L mind agents. Mind agencies vie for the attention of the entire mind agent, thus directly compete with each other for the resources available to them, including the faculties of the agent. It is this level of abstraction that provides the pro-activeness and reactivity of the entire mind agent. Pro-activeness is provided by the pursuit of the mind to achieve the world goal through achievement of the mind goal, while reactivity is provided by the direct access of mind agencies to the lowest level primitives of the mind (described herein as retroaction). In an autonomous agent, mind agencies are formed by motivations that an agent possesses. However, for the basic mind agent, motivations are ascribed by external sources, and hence mind agencies are static and well-formed abstract interfaces of the mind.

4.3.8.1 CSP-OZ Specification

Mind agency's data specification starts with ceiling the number of *MATS_[L]* modules that could be interconnected to form this abstract interface. In the same context as the type L mind agents, we define a *momentum resolution function (mrf)* as receiving the frame differences,

the goal satisfaction result and the conflict effects of all the lower level connected type L mind agents as inputs and producing a numerical value called *momentumeffect* that represents the momentum buildup of increasing differences between the local frame and the goal frame, as well as not satisfying the goal at hand with the presence of conflicts in the layers below. The goal that this mind agency has to satisfy is described as part of the state of this class. The goal is a mind goal, hence is a sequence of lower level mind agents, that if executed properly produce the overall effect of the current mind agency. Other state variables include the currently executing *MATS_[L]* and its corresponding sequence number, as well as the current view of the environment.

The mind agency contains four method schemas: *engage[m], switch, model* and *retroact.* The *engage[m]* schema receives a parameter (*m*) as an input defining the MATS_[L] that is being engaged for action. The *switch* schema modifies the world goal being attained. The *model* schema updates the mind agency's running goal, as well as its momentum. Finally, the *retroact* schema receives a parameter (*m*) as an input defining the *MATS_0* primitive being communicated with. It is quite important to stress the functionality of the momentum of a mind agency: our mind is always responding to demanding urges, such as sleeping, eating or thinking. As soon as one of those urges builds enough momentum to wrest the mind's control away, it will have the mind's resources to perform its required actions, until another urge builds enough momentum to wrest the mind away again, and the cycle continues.

MACY_Data

$maxMATS : \mathbb{N}, mrf : \mathbb{Z} \rightarrow \mathbb{B} \rightarrow \text{seq}\,\mathbb{Z} \rightarrow \mathbb{Z}, fuse : \text{seq}(\mathbb{P}\, WAttribute) \rightarrow WGoal \rightarrow \mathbb{Z}$

$mgoal : MGoal, wgoal : WGoal, momentumeffect : \mathbb{Z}$

$curMATS : MATS\_[L], numMATS : \mathbb{N}, localView : seq(\mathbb{P}\, WAttribute)$

engage[*m*]

$goalView! : \mathbb{P}\, WAttribute$

$goalView! = wgoal$

switch

$\Delta(wgoal)$

$goalView? : \mathbb{P}\, WAttribute$

$wgoal' = goalView?$

model

$\Delta(numMATS, curMATS, momentumeffect)$

$numMATS = \#goal \Rightarrow numMATS' = one$

$numMATS \neq \#goal \Rightarrow numMATS' = numMATS + one$

$curMATS' = goal\langle numMATS\rangle$

$momentumeffect' = mrf\,(fuse\ localView\ wgoal)(satisfy\ goal)$

$(seq(\forall\, mats : MATS\_[L] \mid mats \in mgoal \bullet mats.conflicteffect))$

$satisfy\ wgoal$

retroact[*m*]

$\Delta(localView)$

$currentView? : \mathbb{P}\, WAttribute$

$localView\langle m\rangle' = currentView?$

The mind agency's behaviour specification is similar to that of the mind cell's. The four interface channels are thus defined, along with the temporal behaviour of the class that reads as follows: the mind agent performs a switch operation giving this mind agency control of the mind with a world goal to achieve. The latter is accomplished through interleaved iterations of type L mind agent engagement and modeling, whilst keeping an eye out on the primitive faculties through their retroactions. The process is continuously repeated. It is again important to realize that initializing the mind goal of this mind agency is the responsibilty of the entire mind agent.

MACY_Control

$engage[m][goalView! : \mathbb{P}\, WAttribute]$

$switch[goalView? : \mathbb{P}\, WAttribute]$

$retroact[m][currentView? : \mathbb{P}\, WAttribute]$

$main \stackrel{c}{=} switch \rightarrow (engage[numMATS] ||| retroact[m]) \rightarrow model \rightarrow main$

`inherit` *MACY_Data*

`Init`

$maxMATS = 100$

$momentumeffect = zero$

$curMATS = head\ goal$

$numMATS = one$

`enable_`*engage*$[m]$

`enable_`*switch*

`enable_`*retroact*$[m]$

`effect_`*engage*$[m] = engage[m]$

`effect_`*switch* $= switch$

`effect_`*retroact*$[m] = retroact[m]$

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the mind agency. The former is the union of the sets composed of the *engage* and *retroact* channel events. The mind agency, itself, is specified as its control class synchronized with the interleaved operation of all the type L and type 0 mind agents that it is attached to on the various *engage* and *retroact* channels.

$$\alpha(MACY) = \{engage.cv \mid cv \in \mathbb{P}\, WAttribute\} \cup \{retroact.cv \mid cv \in \mathbb{P}\, WAttribute\}$$

$$MACY \;=\; ((\mathbb{P}\,MATS\_[L] \,|||\, \mathbb{P}\,MATS\_0) \underset{engage,retroact}{\|} MACY\_Control) \setminus \{|\, engage, retroact \,|\}$$

4.3.9 Minsky Mind

The final hierarchical mind entity is the **Minsky mind (MIMI)**. This structure represents the internal view of the entire mind agent, hence consolidates every layer of the mind that we have previously presented into what seems to be one controlling *internal self.* Analyzing this self, we might assume that only one activity is occuring in the mind at one time, namely the currently executing mind agency; however, as we have seen previously, numerous mind agents are executing simultaneously, all trying to attract the mind's attention away from the currently executing agency. The *MIMI* contains a set of world attributes, as well as another of mind attributes, that allows us to view it from an internal and external (next section) perspective.

4.3.9.1 CSP-OZ Specification

The mind's data specification starts with upper-bounding the number of *MACY* modules that could be interconnected to form this abstract interface. A *fitness resolution function (frf)* receives the response time, the goal satisfiability, and the sequence of underlying momenta as inputs, and produces a corresponding normalized fitness of this mind as to how well it is dealing with the changes in the environment and its internal world and mind goals. The mind also possesses a set of capabilities that emerge from its lower level primitives and a sequence of behaviours, representing the mind agencies that it could be in. Important to a MIMI is its ability to tabulate the results of the primitive capabilities of the mind, and to relay these tabulations to higher-level entities for analysis. The state schema defines the currently executing mind agency, the running fitness effect, the powerset of world goals to be achieved by this mind, as well as the state of the MIMI. The latter is quite important as it differs from the static set of world attributes that the MIMI holds. The state is a powerset of dynamic attributes that are also make up the mind, as well as the environment. An important condition is that an attribute cannot be part

of the mind agent's world attributes and state. Also, it is important to note that operations that affect the attributes of a mind agent only affect its state.

The mind contains four method schemas: *switch[m], model, ascribe* and *analyze*. The *switch[m]* schema receives a parameter (*m*) as an input defining the MACY that is being switched to execution. The *model* schema updates the mind's running mind agency (the one with the highest momentum always wins the attention of the mind), as well as its fitness effect. The *ascribe* schema defines how the mind adopts world goals, voluntarily or not. Finally, the *analyze* schema relays the system measurement results, that is behavioural function response time, accuracy and entropy, for further upstream analysis.

MIMI_Data

$maxMACYS : \mathbb{N}$

$frf : \mathbb{Z} \rightarrow \mathbb{B} \rightarrow \text{seq}\,\mathbb{Z} \rightarrow \mathbb{R}$

$behaviors : \text{seq}\, MACY$

$capabilities : \mathbb{P}\, WFunction, wattributes : \mathbb{P}\, WAttribute, mattributes : \mathbb{P}\, MAttribute$

$wgoals : \mathbb{P}\, WGoal, fitnesseffect : \mathbb{R}$

$curMACY : MACY, state : \mathbb{P}\, WAttribute$

---

switch[*m*]

$goalView! : \mathbb{P}\, WAttribute$

$goalView! = \exists\, g : WGoal \mid g \in wgoals$

---

model

$\Delta(curMACY, fitnesseffect)$

$curMACY' = max(behaviors.momentumeffect)$

$fitnesseffect' = frf$

---

ascribe

$\Delta(wgoals)$

$wgoals? : \mathbb{P}\, WGoal$

$wgoals' = wgoals?$

---

analyze

$results! : \text{seq}\,\text{seq}\,\mathbb{N}$

$results! : FigureOfMerit$

$\forall\, cap : WFunction \mid cap \in capabilities;\ index : \mathbb{Z} \mid index = 1..\#capabilities \bullet$

$\quad results\langle index\rangle! = cap.figures$

The mind's behaviour specification is similar to that of the mind cell's. The three interface channels are thus defined, along with the temporal behaviour of the class that reads as follows: the mind is ascribed a set of world goals to be achieved. The latter is accomplished by iterative switches back and forth between the various mind agencies, while attributing a world goal to each mind agency. The process is continuously repeated.

MIMI_Control

$switch[m][goalView! : \mathbb{P}\ WAttribute]$

$ascribe[wgoals? : \mathbb{P}\ WGoal]$

$main \stackrel{c}{=} ascribe \rightarrow model \rightarrow switch[curMACY] \rightarrow analyze \rightarrow main$

`inherit` *MACY_Data*

`Init`

$maxMACYS = 10$

$behaviors = \langle\,\rangle$

$capabilities = \{\,\}$

$wattributes = \{\,\}$

$mattributes = \{\,\}$

$wgoals = \{\,\}$

$fitnesseffect = zero$

$curMACY = head\ behaviors$

$state \neq$

$state \bigcap wattributes =$

$state \bigcup wattributes \subset Environment$

`enable_`*switch*$[m]$

`enable_`*model*

`enable_`*ascribe*

`enable_`*analyze*

`effect_`*switch*$[m] = switch[m]$

`effect_`*model* $= model$

`effect_`*ascribe* $= ascribe$

`effect_`*analyze* $= analyze$

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the mind. The former is the set composed of the *switch* channel events. The mind, itself, is specified as its control class synchronized with all the mind agencies that it is attached to on the various *switch* channels.

$$\alpha(MIMI) = \{switch.cv \mid cv \in \mathbb{P}\, WAttribute\}$$

$$MIMI = ((\mathbb{P}\, MACY) \underset{switch}{\parallel} MIMI\_Control) \setminus \{\mid switch \mid\}$$

4.3.10 Mind Agent

The external view of the entire mind is called the *mind agent*. This entity is responsible for the instantiation of objects and the interconnection between them. It is what is referred to whenever a computational agent, or agent for short, is mentioned in the discourse of the task at hand. It forms a basic type of agent that could be extended in one of eight ways to form more complex agents that are, together, able to form multi-agent systems with dynamics, organizations, attributes and characteristics typical of human groupings. These extensions will be explored in much more detail later in these documents.

4.3.10.1 CSP-OZ Specification

The mind agent's data specification starts with the definition of all state variables, including the powersets of mind agencies, mind agents and mind cells. The internal self is also a state variable, and is viewed as the controller of the mind agent. The same DNA strand that resides in every cell is also viewed at a macro-level, in the mind agent.

The mind agent contains two method schemas: *ascribe* and *survive*. The *ascribe* schema updates the mind agent's powerset of world goals to be achieved, and, the *survive* schema outputs the mind agent's fitness effect.

MindAgent_Data

$mimi : MIMI$

$macys : \mathbb{P}\, MACY$

$mats\_2 : \mathbb{P}\, MATS\_[2]$

$mats\_1 : \mathbb{P}\, MATS\_1$

$mats\_0 : \mathbb{P}\, MATS\_0$

$mikels\_vision : \mathbb{P}\, MIKEL[Vision]$

$mikels\_distance : \mathbb{P}\, MIKEL[Distance]$

$mikels\_wmem : \mathbb{P}\, MIKEL[WorkingMemory]$

$mikels\_hmem : \mathbb{P}\, MIKEL[HippocampalMemory]$

$mikels\_body : \mathbb{P}\, MIKEL[Body]$

$chromosomes : \mathbb{P}\, Chromosome$

$mydna : MYDNA$

survive

$fitnesseffect! : \mathbb{R}$

$fitnesseffect! = mimi.fitnesseffect$

The mind agent's behaviour specification is similar to that of the mind cell's. The two interface channels are thus defined, along with the temporal behaviour of the class that reads as follows: the mind agent ascribes its internal self to a set of world goals to be achieved, and monitors their progress through intrinsic fitness evaluations. The process is continuously repeated. Of importance here is the instantiation of many of the aforementioned structures, including the mind goals of all mind agents, the direction of all the mind agent primitives and the initialization of all lower level entities. A contradicting method is one that inherently makes

all these instantiations and connections, as they are pre-programmed within the DNA strands of every cell of the organism. This method does not require such an extensive mind agent initialization; however, it does call for dynamic layers and dynamic connections, both of which are not properties of the basic mind agent.

MindAgent_Control

$survive[fitnesseffect! : \mathbb{R}]$

$main \stackrel{c}{=} survive \rightarrow main$

`inherit` *MindAgent_Data*

`Init`

$mimi.INIT, macys = \{DANGER, EXPLORE\}$

$DANGER.INIT \wedge EXPLORE.INIT$

$DANGER.mgoal = \langle Avoid, StoreMem \rangle$

$EXPLORE.mgoal = \langle WAnder, Sense, Validate \rangle$

$mats\_2 = \{Avoid, StoreMem\}$

$Avoid.INIT \wedge StoreMem.INIT, Avoid.goal = \langle Stop, Move \rangle$

$StoreMem.goal = \langle Recall, LookAround, Save \rangle$

$mats\_1 = \{Wander, Sense, Validate, Stop, Move, Recall, LoodAround, Sense\}$

$\forall m : mats\_1 \bullet m.INIT$

$Wander.goal = \langle rotateBody, translateBody \rangle, Sense.goal = \langle range \rangle$

$Validate.goal = \langle retrieve, store \rangle, Stop.goal = \langle rotateBody, translateBody \rangle$

$Move.goal = \langle position, orient, rotateBody, translateBody \rangle$

$Recall.goal = \langle retrieve \rangle, Save.goal = \langle store \rangle$

$LookAround.goal = \langle position, orient, range, rotateBody, translateBody \rangle$

$mats\_0 = \{retrieve, rotateBody, translateBody, store, range, orient, position\}$

$\forall m : mats\_0 \bullet m.INIT, chromosomes.INIT \wedge mydna.INIT$

$(retrieve \vee range \vee orient \vee position).dir = zero$

$(rotateBody \vee translateBody \vee store).dir = one$

$mikels\_vision = \{vis1, vis2, vis3\}, \forall mikels : mikels\_vision \bullet mikels.INIT$

$mikels\_distance = \{dis1\}, \forall mikels : mikels\_distance \bullet mikels.INIT$

$mikels\_wmem = \{wmem1, wmem2\}, \forall mikels : mikels\_wmem \bullet mikels.INIT$

$mikels\_hmem = \{\}, \forall mikels : mikels\_hmem \bullet mikels.INIT$

$mikels\_body = \{body1, body2\}, \forall mikels : mikels\_body \bullet mikels.INIT$

The layer 4 (pure CSP) specification consists of defining an alphabet and an object instantiation composition for the mind agent. The former is the set composed of the *ascribe* channel events. The mind agent, itself, is specified as its control class synchronized with the interleaved operation of the internal mind agency and the evolutionary process that it is attached to on the various *ascribe* channels.

$$\alpha(MindAgent) = \{ascribe.wg \mid wg \in \mathbb{P}\, WGoal\}$$

$$MindAgent = ((MIMI|||Evolution) \underset{ascribe}{\|} MindAgent\_Control) \setminus \{|\, ascribe \,|\}$$

4.3.11 MAPS Summary

Figure 4.6 shows the hierarchical structure of the multi-agent framework. Starting with the aforementioned cells, we have these additional specifications:

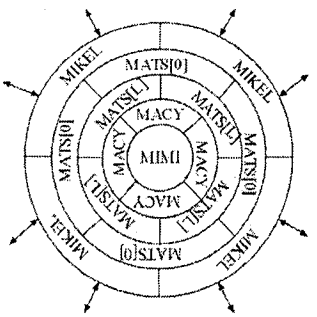


Figure 4.6: Mind agent hierarchy

- **MIKEL**: Entity (cell) from which all *life* is made from (agents, sensors, actuators, minds, etc);
- **MYDNA**: Attributes of a MIKEL (e.g. reactive, self-starting, self-sustaining, etc);
- **MATS**: Unintelligent agent of the mind consisting of a simple process that turns other MATS on or off;

- **MACY**: Agency (group of mind agents) that represents a behavior that the entire mind agent can be occupied with;
- **MIMI**: Brain of the being (e.g. robot), more like the internal view (self) of the entire mind agent; and

Learning in this architecture is performed by the inner mind agent layers, that is, the MATS layers. A *learning mind agent* can dynamically create higher-level MATS to improve its mind agencies (MACYs), and build on the knowledge below by helping it organize its mind hierarchy. The latter is performed by informing the mind agent when and how to use things that it already knows.

4.4 Multi-Agent Platform Simulation Environment

Having formally specified the system at hand, it is now time to turn our attention to the virtual prototyping tool, called the *Multi-Agent Platform Simulation Environment (MAPSE)*, that will allow us to simulate various environment monitoring scenarios. The agent-based scenarios can be viewed as *games* that are played with different scripts defining the rules of each game. Using the response time, accuracy and entropy figures of merit associated with each agent's behavioral function, it is possible to evaluate the progress of the game and to evaluate global figures of merit for a given set of agents and a given plan of behavioral functions.

The environment is to be monitored by a group of agents working togther and controlled by a human agent's motivations. The output of the simulator should be an agent activity trace as well as a GUI that can visually demonstrate the agent behavior and interactions. MAPSE has the following functional requirements:

- The agents have differents shapes, colours, sensors, actuators (i.e. stateless attributes)
- The agents possess situation attributes such as position in the environment, velocity, motion, etc...

- Barriers, obstacles and resources can be modeled in the environment as a powerset of world attributes
- The field of perception for an agent is dependent on its available sensors
- The site of perception and/or actuation is dependent on the location of the sensors/actuators on the agent
- Hazards are modeled as anomalies in the environment that are to be mapped and possibly contained
- Agent communication is modeled through asynchronous message passing
- Agent movement is modeled by the differential-drive (and synchro-drive) motion models

Java, again, was used as the language for the development of the simulation environment, while Sun Netbeans was selected as the integrated development environment. Other technologies that were used for the realization of MAPSE's modules include:

- the actual environment to be monitored using Java Swing
- the agent architecture using J2EE and Enterprise Java Beans (EJB)
- the agent communication using Java event listeners
- the configuration of the environment using XML

Additionally, the tool allows the user to display and edit the agent obtained maps, add/remove an agent, customize an agent, track and plot an agent's path, track and plot an agent's on-board sensors and energy supply, and modify the behavior of an agent. The simulator allows each agent to communicate with other agents, traverse an unstructured environment, collect sensory data, and synthesize a plan and associated goal sequence to achieve the plan.

4.4.1 UML Diagrams

We will now present three UML collaboration diagrams that conform to the MAPS framework. The UML collaboration diagram for the environment monitoring simulation environment is shown in Figure 4.7. A multi-agent system can interact with the environment as well as with the user. The evolution module picks out the best agent chromosomes and feeds the nest module, that directs the spawning of the next generation of mind agents.

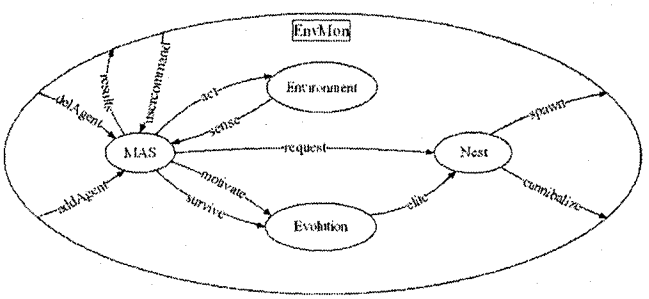


Figure 4.7: Environment monitoring collaboration diagram

The UML collaboration diagram for the multi-agent system is shown in Figure 4.8. The human agent provides an interface to humans so that they may analyze and direct the operation of the MAS. Each mind agent can act and sense the environment, as well as publish its chromosomes and results.

The UML collaboration diagram for the mind agent is shown in Figure 4.9. The mind agent is composed of many layers, as shown in Figure 4.6, most of which have already been discussed in this section. Here, it suffices to note that the mind agent interfaces to other entities through its highest level (i.e. *mimi*) and its lowest level (i.e. *mikels*) entities. The UML class diagrams for the entire MAPSE application are presented in Appendix A.

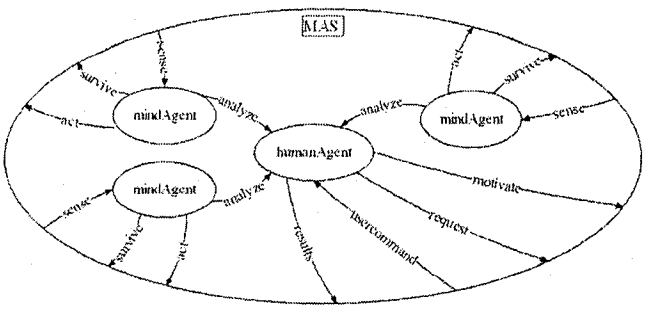


Figure 4.8: Multi-agent system collaboration diagram

4.4.2 Snapshots

Let us conclude this section by viewing a few snapshots of the MAPSE while in action. Table 4.3 briefly describes each snapshot.

Snapshot #	Description
1	Plotting a mind agent's path on the fourth floor of the selected building, with sensor values shown, in real-time, in the agent status data table
2	Performing the OGF method with two agents slaved to each other, with agent events shown to the right
3	Plotting agent resource utilization in real-time, when one agent decides to cannibalize another's resources

Table 4.3: MAPSE snapshot descriptions

4.5 Summary

This chapter has presented a novel agent architecture, that aims to resolve the problems of calculative rationality and short-term views, while providing for a simple mapping of the sub-systems and layers. Also presented in this chapter, are the multi-agent system's specifications that allow for the inclusion of reactive, proactive and retroactive agents within a common framework, and to analyse their interactions. Numerous specification languages were researched and analysed until the CSP-OZ language was selected, due to its expressive and powerful syntax, its ability to model concurrent and reactive systems and its foundational integration paradigm.

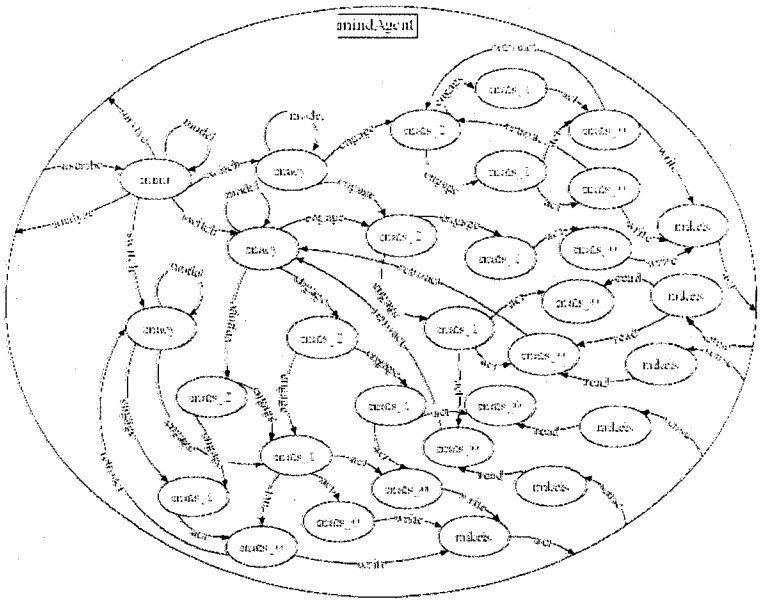


Figure 4.9: Mind agent collaboration diagram

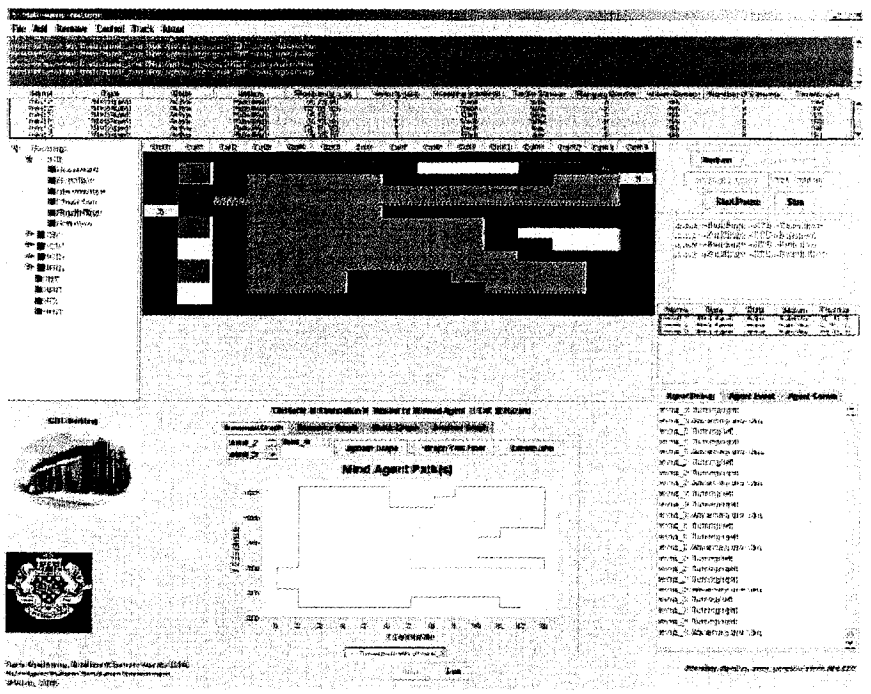


Figure 4.10: MAPSE snapshot 1

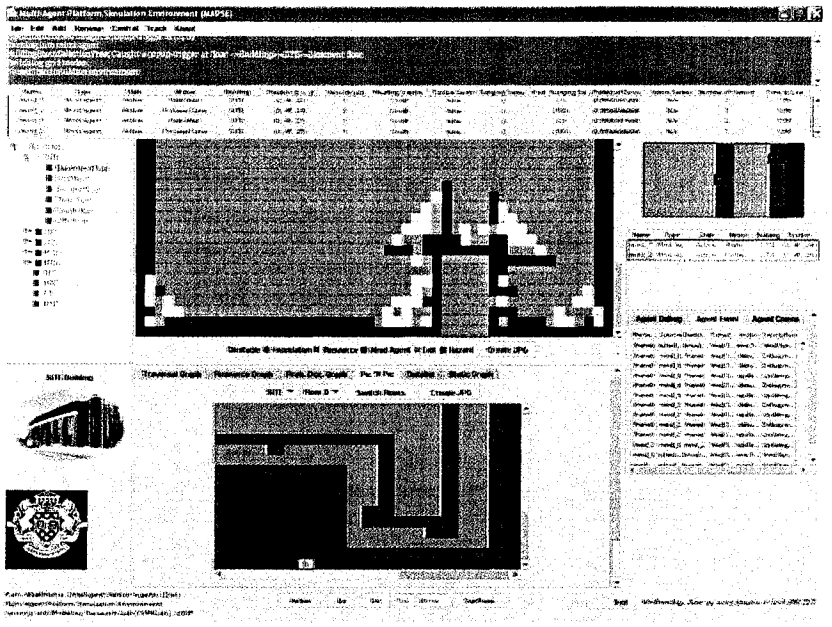


Figure 4.11: MAPSE snapshot 2

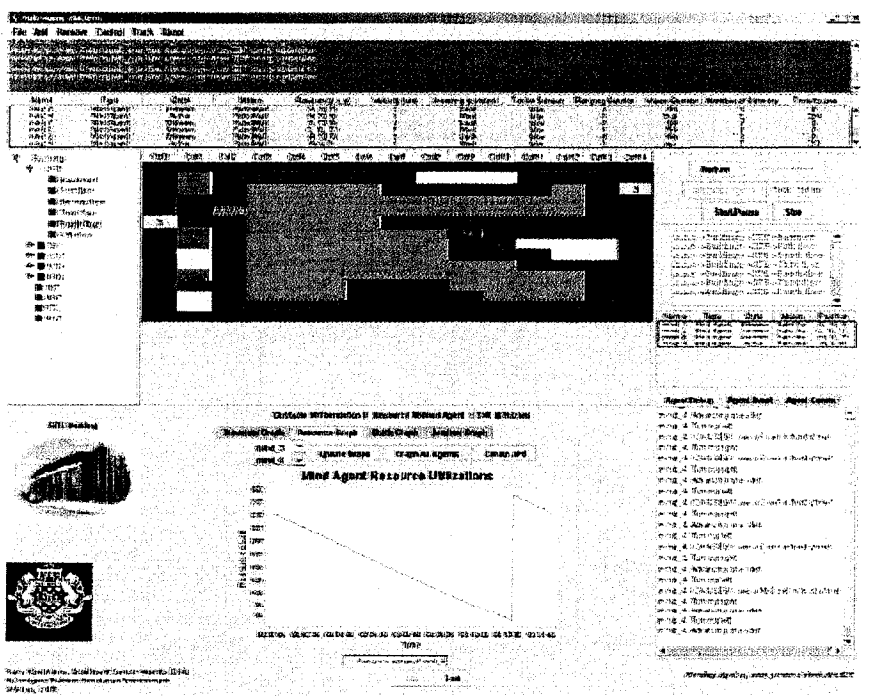


Figure 4.12: MAPSE snapshot 3

Chapter 5

Simulation Results

"The advantage of modern means of communication is they enable you to worry about things in all of the world"

(Laurence Peter)

In this chapter, we develop and present the simulation scenarios and results concerning the map building method and its associated data structure storage and access. We compare the presented method to two prevalent methods in terms of their grid storage size, certainty value update time, exploration strategy decision time step and map extraction time. Both the centralized and decentralized multi-sensor fusion approaches are simulated and presented. Furthermore, the chapter presents the simulation accuracy and noise immunity results, concluding with the simulation results of the mutual information metric. Note that we had developed both differential-drive and synchro-drive simulation models, the latter requiring wheel angular velocities and steering angles; however, we present the former's simulations, since it was decided to use the differential-drive agents in the experimental setup.

5.1 Simulation Results

Two simulation environments have been created, one using the Matlab [98] tool, and another using the Java programming language [14]. This section will concentrate on the former, that involves a simulated environment, filled with walls and obstacles. A set of agents is introduced to perform environment map building, with each sensor agent behaving as modeled by the aforementioned data models. The global goal of each sensor agent is to minimize the total

environment entropy; however, the sub-goals are to maximize or minimize each cell's probability of occurrence, as well as the traversal towards high-entropy regions of the environment. Looking back at Figs. 3.8 and 3.10, the TIMM update module in the C-MSF processor flow reads in the FoQ data structure and updates it to increment/decrement the appropriate cell's probabilities, as well as update the entropies at the various affected nodes. The TIMM traversal module contains the sub-goals of each sensor agent, which currently involve heading towards high-entropy regions of the environment.

It is notable that neither quadtree data structures nor pointers do not inherently exist in Matlab. We have thus used the *Data Structures and Algorithms Toolbox (DSTABX)* [99], as well as the *Pointers Toolbox* [100]. Using those toolboxes, we were able to create quadtree-based data structures for use in our simulation environment.

5.1.1 C-MSF Simulations

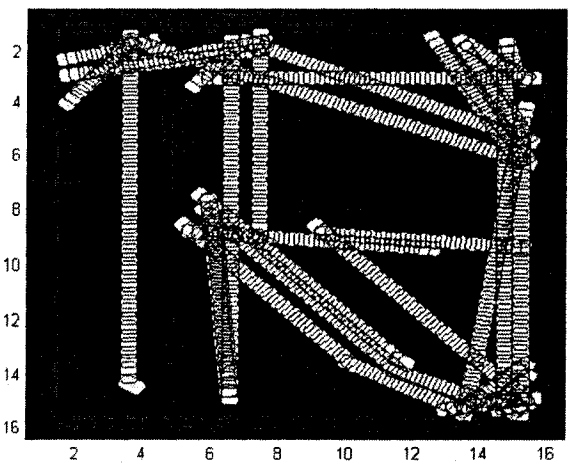


Figure 5.1: Sensor agent environment mapping with 3 ISAs

Taking a look at the path traversed by three agents (Figure 5.1) mapping the environment, one can see that the agents attempt to cover the middle of the environment first, before moving towards the extremes of the environment. There are 256 total tessellations to this particular

Number of agents	EntropyA (bits)	EntropyB (bits)	Savings (bits)	Savings (%)
1	219	151	105	41
2	201	133	123	48
3	180	112	144	56
4	164	96	160	63
5	155	87	169	66
6	140	72	184	72
7	137	69	187	73
8	121	53	203	79
9	129	61	195	76
10	102	34	222	87
15	103	35	221	86
20	86	18	238	93

Table 5.1: Entropy measures for various agent configurations

environment; therefore the highest entropy measure would be 256 bits. Table 5.1 shows the savings, in bits, for the representation of the entropy of the environment as the number of agents mapping the environment is increased. Note that although 256 bits is the maximum entropy, we set the minimum to be 68 bits for this environment (second column). This is attributed to the 60 boundary nodes and 8 obstacle nodes, such that we can visualize them graphically. The third column of Table 5.1 indicates the savings in bits taking into account this minimum, while Figure 5.2 plots the entropy savings, in bits, of the different agent configurations shown in Table 5.1. It is interesting to note that injecting more than 20 agents into this particular environment slowed the entropy reduction, due to the overlapping of many of the sensing regions of the ISAs and the increasing ISA-ISA collisions.

Looking at the entropies, over time, of each cardinal region of the environment (refer to Figure 5.3), one can even notice where the agent is spending its time: a decrease in entropy across a time period indicates that an agent is within that region of the environment, whereas a straight line indicates that the particular cardinal region is not being explored.

Moreover, analyzing the entropy landscapes of the environment at different resolutions provides us with a graphical map of the environment that can be easily studied for environment monitoring in terms of: obstacle detection and avoidance, hazard detection and containment, and anomaly detection and registration. Figs. 5.4 and 5.5 provide the entropy landscapes of the

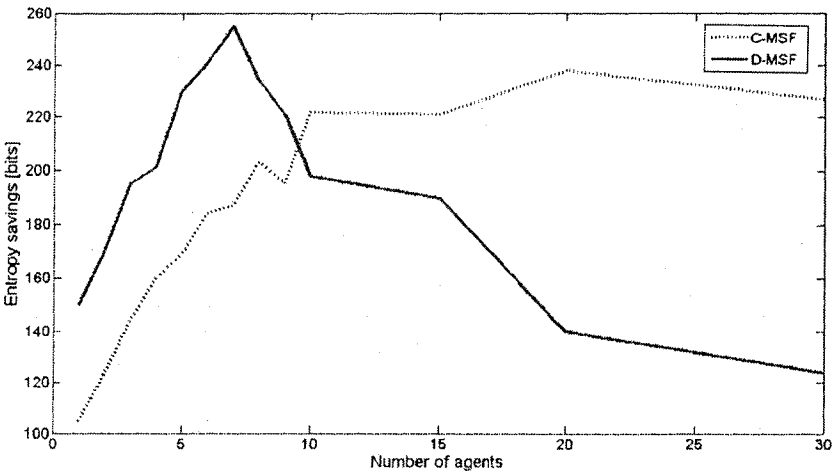


Figure 5.2: Entropy savings for various agent configurations (C-MSF vs. D-MSF)

environment at quadtree depths 0, 1, 2 and 3, for the cases of 3 ISAs and 20 ISAs monitoring, respectively. As can be seen from the entropy landscapes, valleys exist in low-entropy regions, while mountains exist in high-entropy regions. Looking at different resolutions of the map provides a more detailed picture of the environment, and its structure. The global goal becomes the flattening of all regions of the environment, so that no unknowns exist. As is shown in Figure 5.5 (at quadtree depth 3), the environment is nearly entirely mapped within 250 simulation time ticks; however, 18 bits of entropy (uncertainties) are still required to encode the map (as shown in Table 5.1).

Finally, studying the contribution of each agent to the reduction of environmental entropy in Figure 5.6, one can notice that certain agents contribute more to the reduction of entropy than others, as shown by Agent 3's 39% contribution to the overall entropy reduction, compared to Agent 1's 30% contribution, in the case of 3 ISAs in the environment. As aforementioned, each agent's reading can either be complementary or competitive compared to other agents' readings. Figure 5.8 shows the distribution, for each agent, of complementary (black bar) to competitive readings (grey bar). Note that since they are mobile, the sensor agents can be complementary to each other at one time, and competitive at another. If fixed, a sensor agent would either be

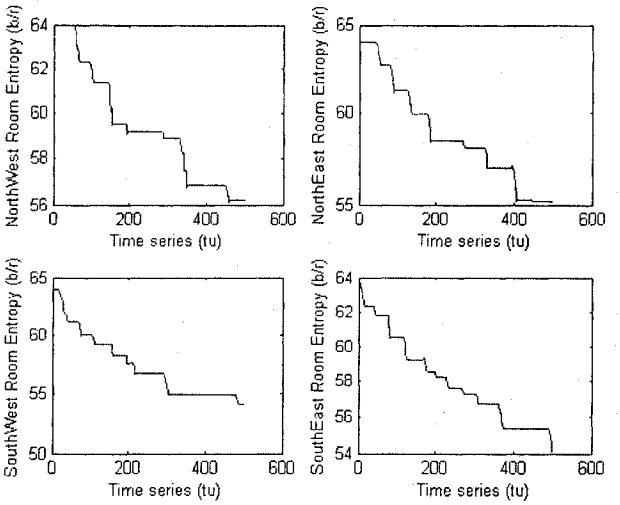


Figure 5.3: Entropy graphs for the 4 cardinal regions for the case of 3 ISAs mapping

complementary or competitive at all times, with respect to other fixed agents. Complementary sensor readings are appended to the environment map, whereas competitive sensor readings are pitted against each other and the *winner* is added to the map. To select a reading, the following procedure is undertaken:

(1) Check if readings are not reflections of the sensor agents themselves;

(2) If the readings agree, then the winner is any of them;

(3) If there is a disagreement between readings, then check the history of the physical location (i.e. probability of occurrence) and then the history of the sensor agents themselves (i.e. their confidence). Choose the reading that best matches both.

As can be noticed in Figure 5.6, in the case of 3 ISAs mapping the environment, agents 1 and 2 were in competition more often than either agents 1 and 3 or agents 2 and 3 were, since agents 1 and 2 were in the vicinity of each other, hence were taking measurements of the same physical areas.

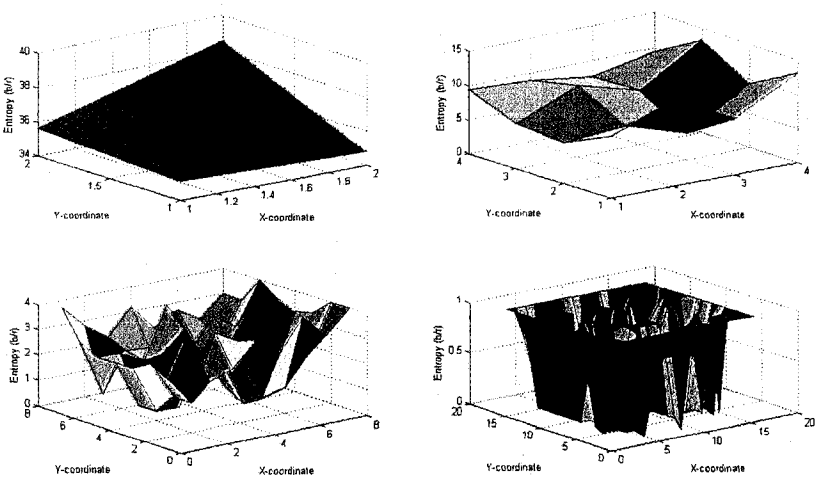


Figure 5.4: Entropy landscape for the case of 3 ISAs mapping

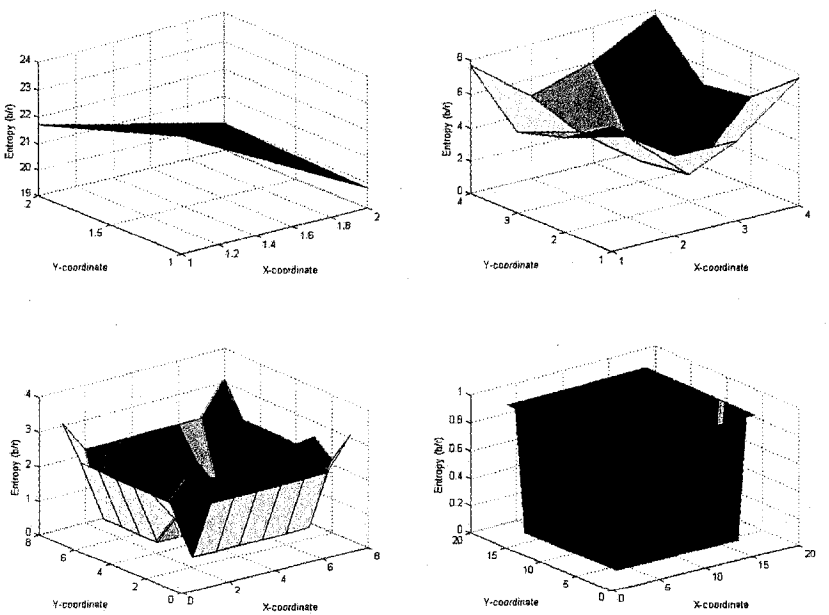


Figure 5.5: Entropy landscape for the case of 20 ISAs mapping

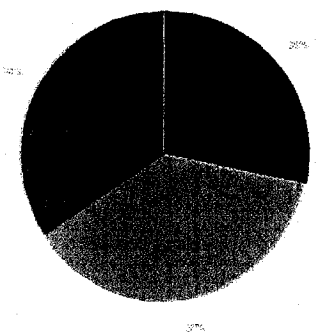


Figure 5.6: Agent contribution to entropy reduction for 3 agents

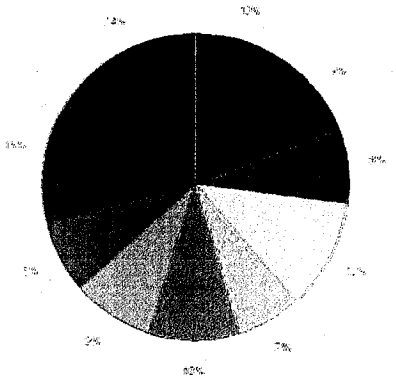


Figure 5.7: Agent contribution to entropy reduction for 10 agents

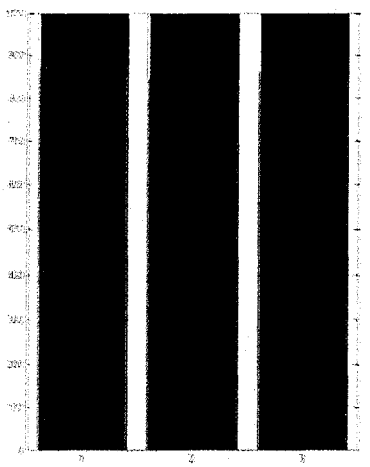


Figure 5.8: Agent distribution of complementary vs. competitive readings for 3 agents

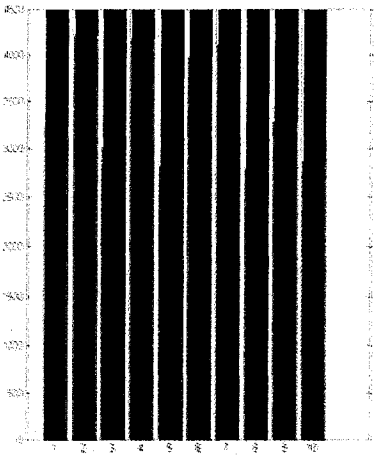


Figure 5.9: Agent distribution of complementary vs. competitive readings for 10 agents

5.1.2 D-MSF Simulations

Shown in Figure 5.2 is a comparison between C-MSF and D-MSF runs through the same environment, but with various agent configurations. It can be seen that the C-MSF scenario reaches its optimum number of agents of 20, at an associated entropy savings of 238 bits. In the case of the D-MSF scenario, the optimum number of agents is 7, with an associated entropy savings of 255 bits. It can thus be deduced that a decentralized approach to multi-sensor fusion is more advantageous in that a smaller number of agents is required, notwithstanding the elimination of the computational bottleneck of a central server coordinating the actions of multiple agents, as in the C-MSF simulations.

However, it can also be seen from the figure that the D-MSF simulations degrade very rapidly soon after reaching the optimum number of agents in the environment, witnessed by the rapid decline in entropy savings for agent configurations larger than 7. Meanwhile, in the C-MSF simulations, it can be seen that even at 30 agents in an environment whose optimum is 20, the performance of the system is degraded by approximately 5% only.

Not shown on the figure, but still interesting to note, is that for the optimum agent configurations, the C-MSF simulations required 250 simulation ticks to achieve the maximal entropy savings, whereas the D-MSF simulations required 126 simulation ticks to achieve its maximal entropy savings. This leads to a monitoring time speedup of 50% when using the decentralized approach, mostly due to the communications bottlenecks inherently present in the C-MSF scenarios.

5.1.3 Comparisons and Discussion

The three techniques have already been theoretically compared in section 3.2.1; we will use the following section to simulate all three methods and analyze their results. First off then, let us compare **grid storage sizes.** Both OGF and TIMM use floating-point CVs, hence require 4 bytes per tessellation, whereas HIMM uses integer CVs, hence requires 1 byte per tessellation.

This makes HIMM, in theory, four times as storage efficient as both OGF and TIMM. We shall later see; however, that the forest of quadtrees representation actually saves storage size by integrating tessellations with common states.

Second, let us compare **CV update times**. The OGF technique uses Bayes' rule for its CV updates, a calculation that requires 1 product and 1 division, both involving floating-point arithmetic, hence 2 floating-point operations (FLOPs). The HIMM technique uses an increment/decrement of the previous CV, as well as a GRO weighing factor addition, the latter calculated over a 3x3 matrix through a summation of products. This update requires 10 additions and 9 products, all involving integer arithmetic, hence 19 integer operations (IOPs). The TIMM technique uses an increment/decrement of the previous probabilities and a summation of products for the new CV. This update requires 1 addition, 2 product and 2 logarithm, all involving floating-point arithmetic, hence 5 FLOPs. As shown in Figure 5.10, the OGF CV update time is the slowest, followed by the HIMM and TIMM CV update times, which are approximately equivalent.

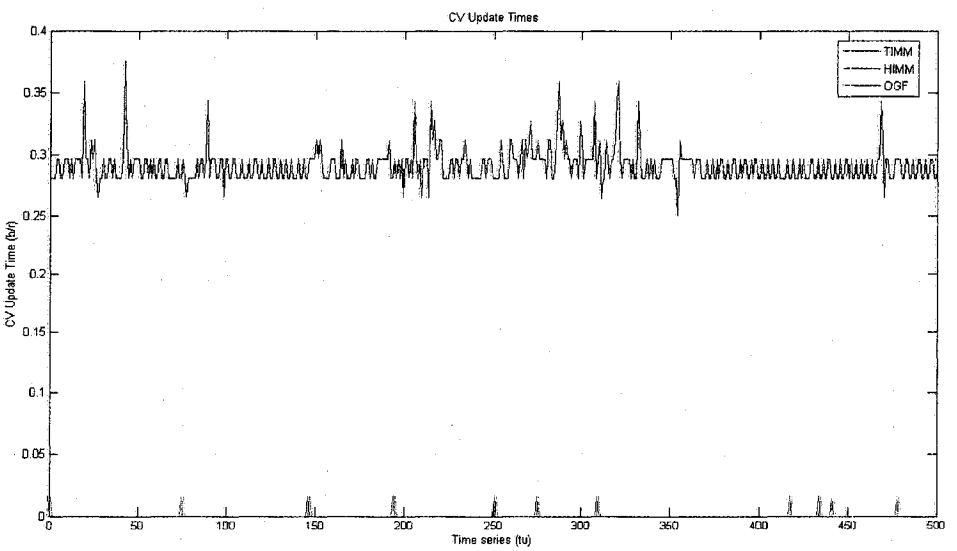


Figure 5.10: OGF vs. HIMM vs. TIMM CV update times

Third, let us compare **exploration strategy decision time steps**. Both the OGF and

TIMM techniques look at middle and high CVs, respectively, and proceed towards those regions; however, the HIMM techniques has to decide between the low and mid CV regions since a low to mid value can either indicate the absence of an obstacle or the lack of exploration by the agent. For example, the HIMM technique has to decide between cells with CVs that are 12 or less, in an environment with a value range of [0,15] as in [45]. This implies that the HIMM technique decides on approximately 80% of the range. On the other hand, the OGF technique decides on approximately 50% of the range, since unknown regions are typically composed of cells with probabilities between 0.25 and 0.75. Similarly, the TIMM technique decides on approximately 50% of the range, since unknown regions are typically composed of cells with entropies above 0.50, and (obviously) lower than 1.0. As shown in Figure 5.11, HIMM's decision step time is relatively high, as compared to OGF's and TIMM's.

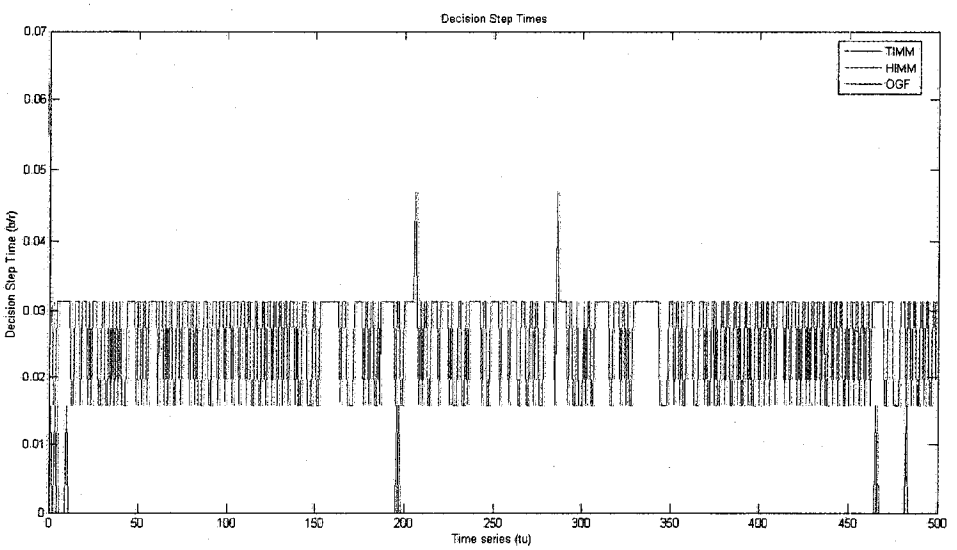


Figure 5.11: OGF vs. HIMM vs. TIMM decision step times

Fourth, let us compare **map extraction times**. Refer to Figure 5.12 for a graph showing the three techniques as applied to the same environment. It can be seen from the figure that the TIMM technique extracts the map faster than either of the OGF or the HIMM technique. All simulations ran for the same number of iterations; however, their actual simulation times

differed.

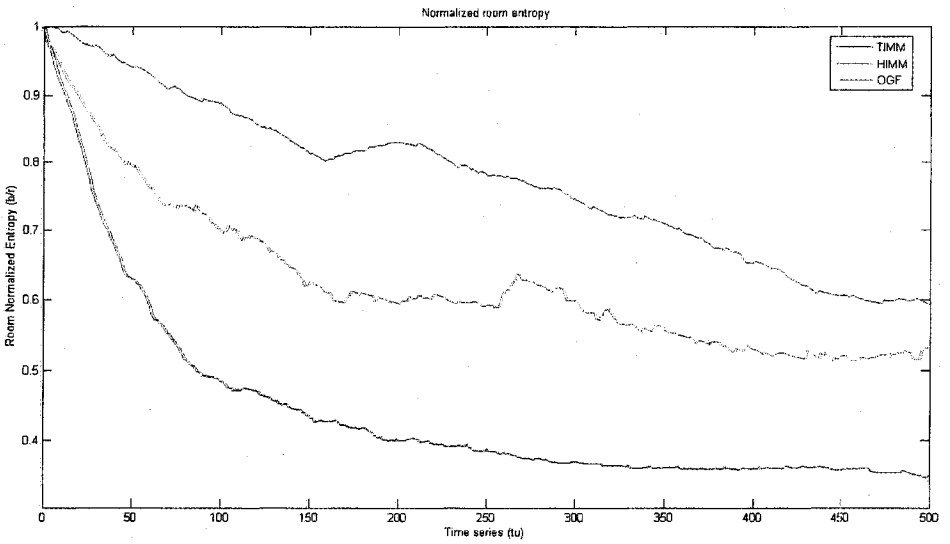


Figure 5.12: OGF vs. HIMM vs. TIMM map extraction times

5.1.4 Environment Mapping Simulations

To demonstrate the accuracy of the estimated environment maps, we present in this section environment maps generated from three different case runs. The entropy-directed multi-agent search is used, while the sensor coverage over the search area is provided by the various entropy landscapes. Below, we shall see the generated environment maps that are the output of such a search system.

For the figures below we shall follow the following indexing scheme: X-1 depicts the top left graph, X-2 depicts the top right graph, X-3 depicts the middle left graph, X-4 depicts the middle right graph, X-5 depicts the bottom left graph, and finally X-6 depicts the bottom right graph.

In Figure 5.13, we see the environment that was depicted in Figure 5.1. Figs. 5.13-3 and 5.13-4 show the occupancy and emptiness probability maps, while Figure 5.13-2 shows the estimated map, filtered by taking any probably occupied cell (i.e. $>= 0.5$) to be occupied, while

any probably empty cell (i.e. < 0.5) to be empty. Figure 5.13-5 shows the exact accuracy between the occupied map (Figure 5.13-3) and the true map (Figure 5.13-1). Figure 5.13-6 shows a gradient flow, with the x-axis component depicting empty cells, and the y-axis component depicting occupied cells. Hence, any vector that is completely horizontal indicates an empty cell, and any vector with a slight vertical heading indicates an occupied cell, with the heading of the vertical component depicting the probability of occupation.

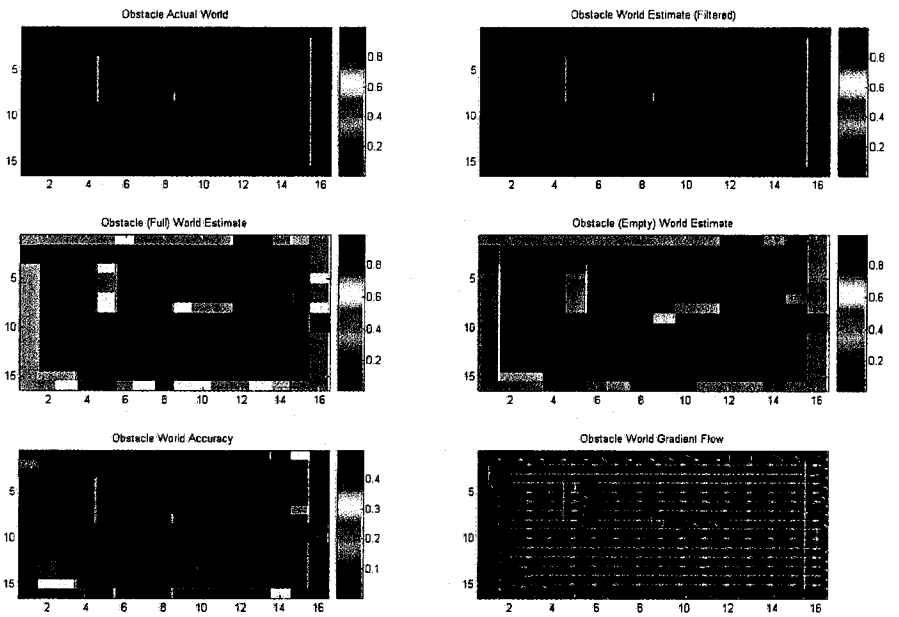


Figure 5.13: Environment mapping case 1

In Figure 5.14, we see another environment mapping case, and its associated mapping results. Again, Figs. 5.14-3 and 5.14-4 show the occupancy and emptiness probability maps, while Figure 5.14-2 shows the estimated map. Note that in this case, a cell was incorrectly labeled as occupied (see Figure 5.14-2). It is also interesting to note that certain areas cannot be searched and hence will retain their high entropy, and initial occupancy probabilities. The accuracy map (Figure 5.14-5), is obtained by calculating the absolute value of the results in Figure 5.14-3 minus the results in Figure 5.14-1.

In Figure 5.15, we see another environment mapping case, and its associated mapping results. Again, Figs. 5.15-3 and 5.15-4 show the occupancy and emptiness probability maps, while Figure 5.15-2 shows the estimated map. Note that also in this case, a cell was incorrectly labeled as occupied (see Figure 5.15-2). The accuracy map (Figure 5.15-5), is obtained by calculating the absolute value of the results in Figure 5.15-3 minus the results in Figure 5.15-1

In Figure 5.16, we see the same environment mapping case as in Figure 5.15, and its associated mapping results. Note two differences with these outputs: first, that the incorrectly labeled cell has been correctly labeled as empty in this run, and second, that taking the accuracy to be the difference between the estimated (filtered) map, rather than the occupancy map, yields a perfect match (see Figure 5.16-5). Hence, the accuracy map (Figure 5.16-5) here, is obtained by calculating the absolute value of the results in Figure 5.16-2 minus the results in Figure 5.16-1.

Note that for all the runs, 20 agents were used, with no noise being added. Noise will be included in the next section and the corresponding outputs will be shown.

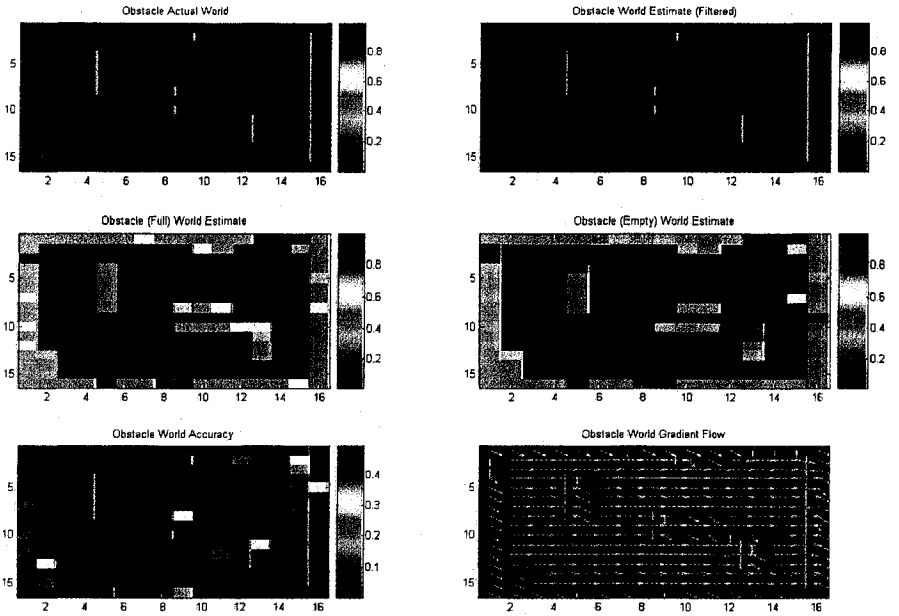


Figure 5.14: Environment mapping case 2

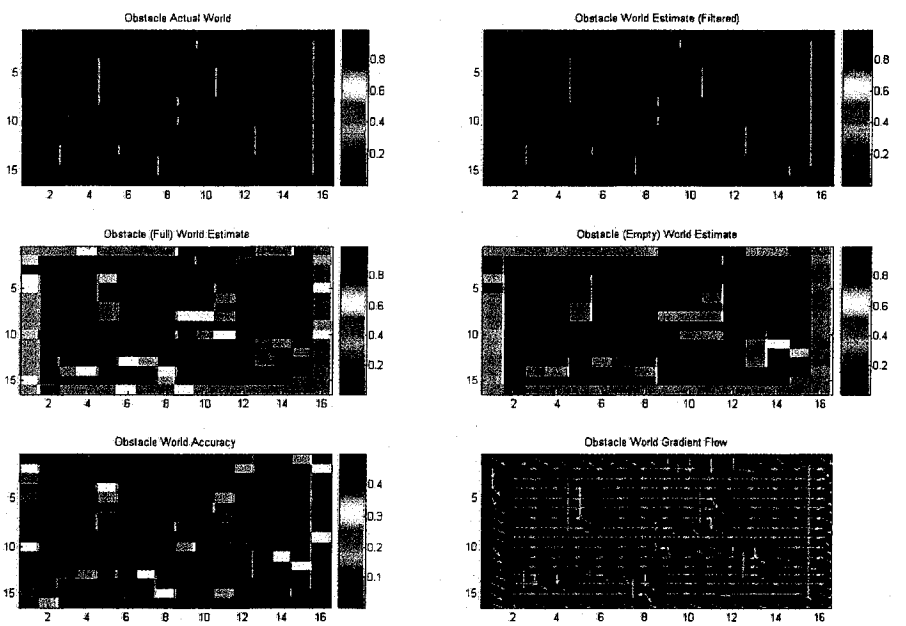


Figure 5.15: Environment mapping case 3a

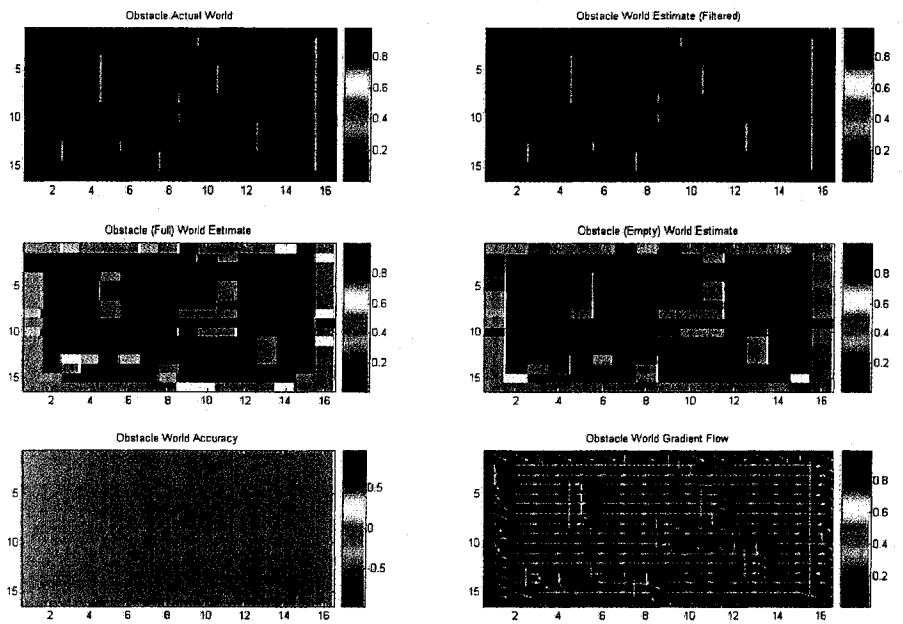


Figure 5.16: Environment mapping case 3b

5.1.5 Noise Simulations

To produce claims of statistical significance and general performance, imperfections in agent motion and position/orientation estimates are applied to the system. Presented in this section are environment maps generated from the same case run, while perturbed by additive white Gaussian noise. Below, we shall see the effects that such noise inclusions have on the system, and the mapping outputs and their associated accuracies.

For the figures below we shall follow the following indexing scheme: X-1 depicts the top left graph, X-2 depicts the top right graph, X-3 depicts the middle left graph, X-4 depicts the middle right graph, X-5 depicts the bottom left graph, and finally X-6 depicts the bottom right graph. Also of note is that we have changed the depiction of the color scheme to be uniform amongst all graphs, hence the range of colors will always be from 0.0 to 1.0.

The noise that was added was additive white Gaussian noise with a varying standard deviation, that corrupts the position, and the orientation estimates (called *pose noise*), as well as the sensor readings (called *sensor noise*). In all cases, we chose the following inclusion of noise, either to the pose estimates or to the sensor readings:

$$noisySignal = actualSignal + randn * \sqrt{V} \tag{5.1}$$

where,

noisySignal is the generated noisy signal;

actualSignal is the actual signal (e.g. x-coordinate or sensor reading);

randn is the normally distributed random function;

V is the input variance, whose square root represents the std. deviation.

In Figure 5.17, we see the environment that was used in the following runs. Figs. 5.17-3 and 5.17-4 show the occupancy and emptiness probability maps, while Figure 5.17-2 shows the estimated map. Figure 5.17-5 shows the accuracy measure between the estimated map (Figure

5.17-2) and the true map (Figure 5.17-1). Figure 5.17-6 shows the gradient flow depicting cell vectors that indicate the occupancy and emptiness probability measures as x and y components.

For this case, pose noise had a variance of 0.5, and sensor noise had a variance of 0.5. Both are shown in Figure 5.18, with Figure 5.18-1 showing the x-coordinate actual and noisy signals, Figure 5.18-2 showing the y-coordinate actual and noisy signals, Figure 5.18-3 showing the orientation actual and noisy signals, and finally, Figure 5.18-4 showing the sensor reading actual and noisy signals. Note that these noisy signal graphs are for the last agent only; however, the measures were tabulated for all agents involved in the run.

In Figure 5.19, we used a variance of 1.0 for both pose estimate and sensor reading noise inclusions (see Figure 5.20).

In Figure 5.21, we used a variance of 2.0 for pose estimate and 3.0 for sensor reading noise inclusions (see Figure 5.22).

In Figure 5.23, we used a variance of 9.0 for pose estimate and 2.0 for sensor reading noise inclusions (see Figure 5.24).

As can be seen, the system is quite robust against noise, both in pose estimate signals and sensor reading signals. In the last case run, it was noticeable that noise was affecting the mapping results. This was not very drastic, considering the standard deviation of the Gaussian noise corrupting the pose estimates was at 3.0, that theoretically indicates that some of the noisy values could be as far off as 49.8% off from the mean value. Note that for all the runs presented herein, 20 agents were used.

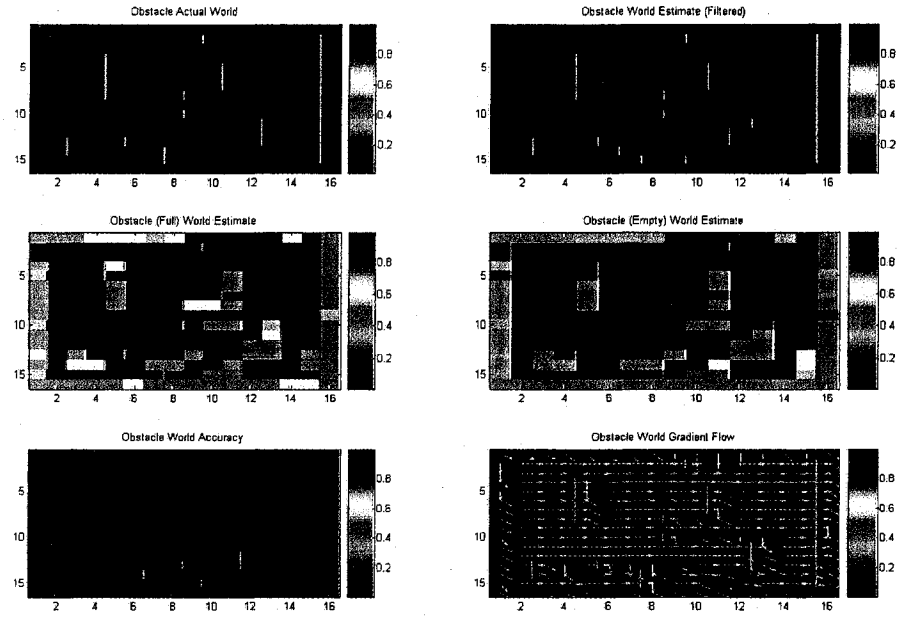


Figure 5.17: Environment mapping with noise case 1

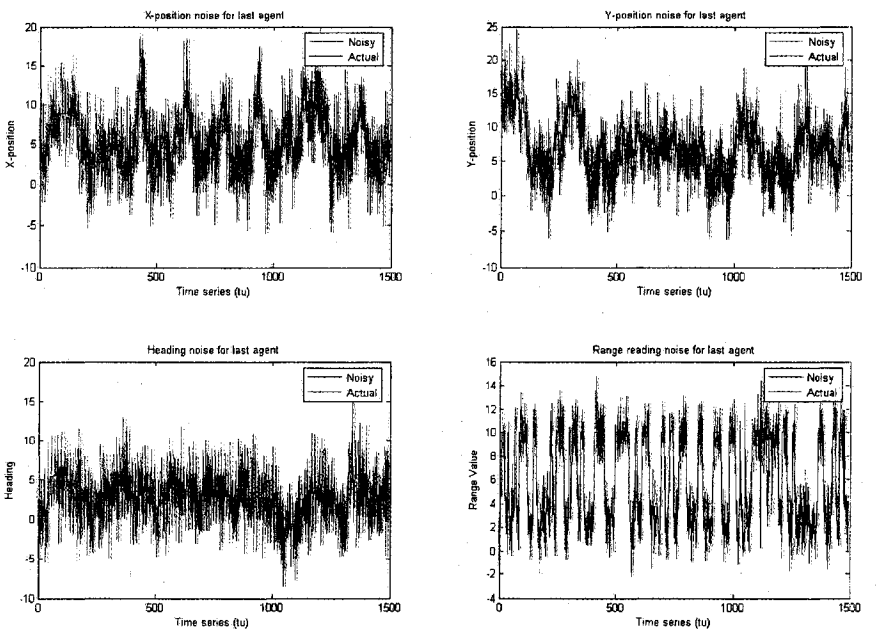


Figure 5.18: Noise signals for pose estimates and sensor readings (case 1)

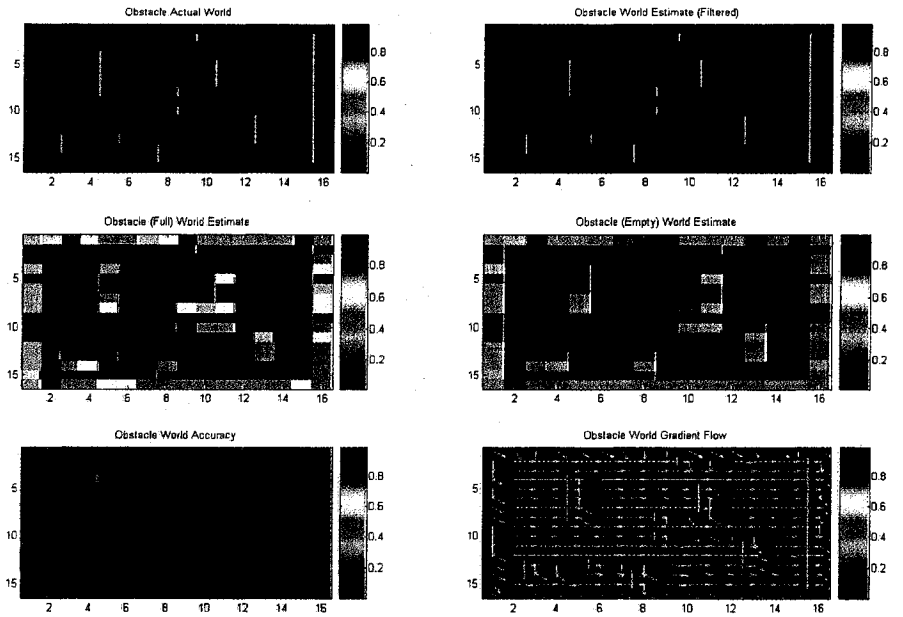


Figure 5.19: Environment mapping with noise case 2

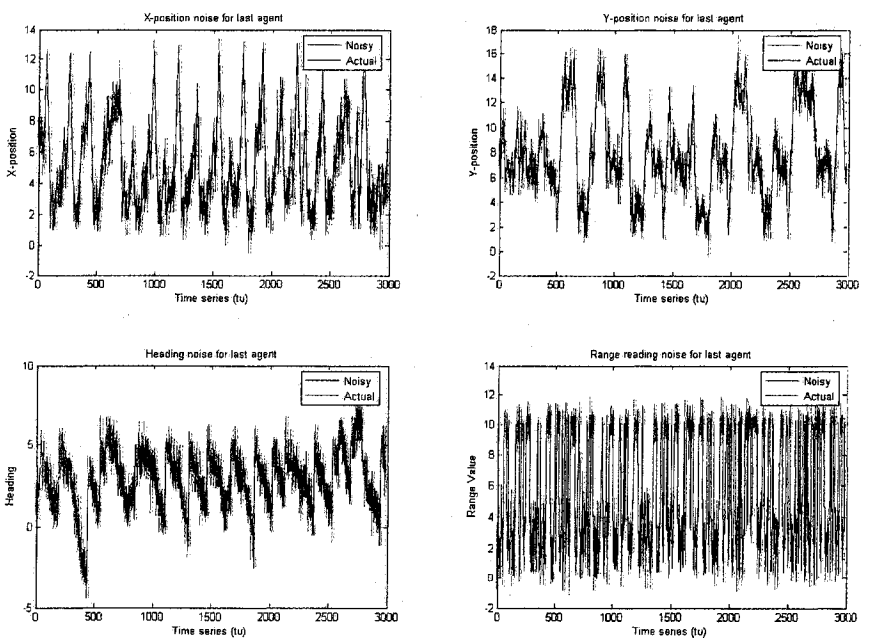


Figure 5.20: Noise signals for pose estimates and sensor readings (case 2)

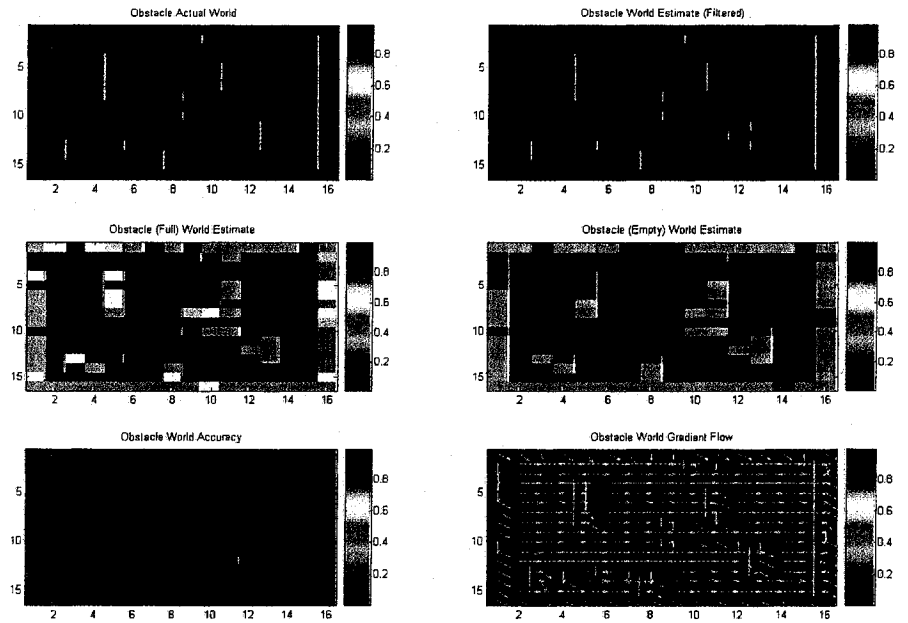


Figure 5.21: Environment mapping with noise case 3

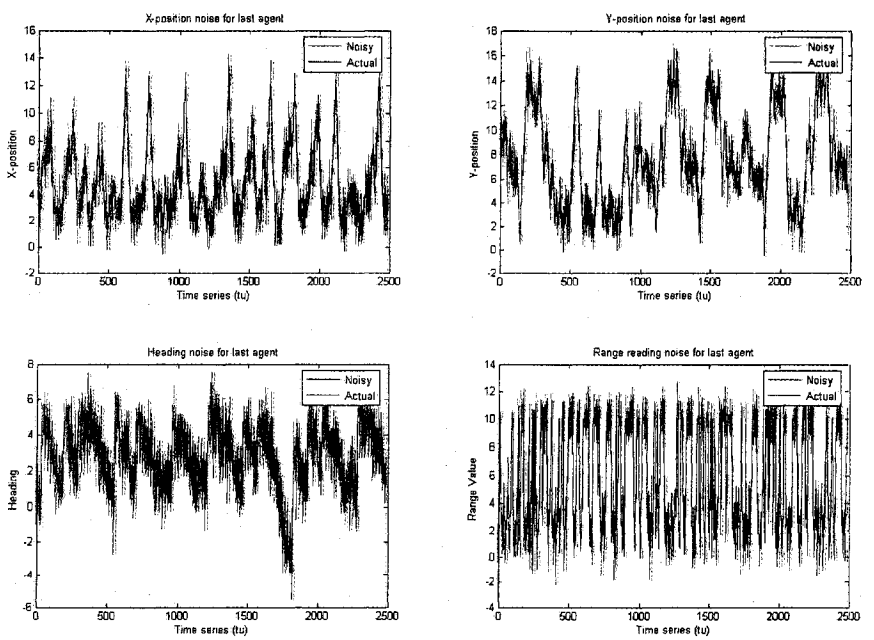


Figure 5.22: Noise signals for pose estimates and sensor readings (case 3)

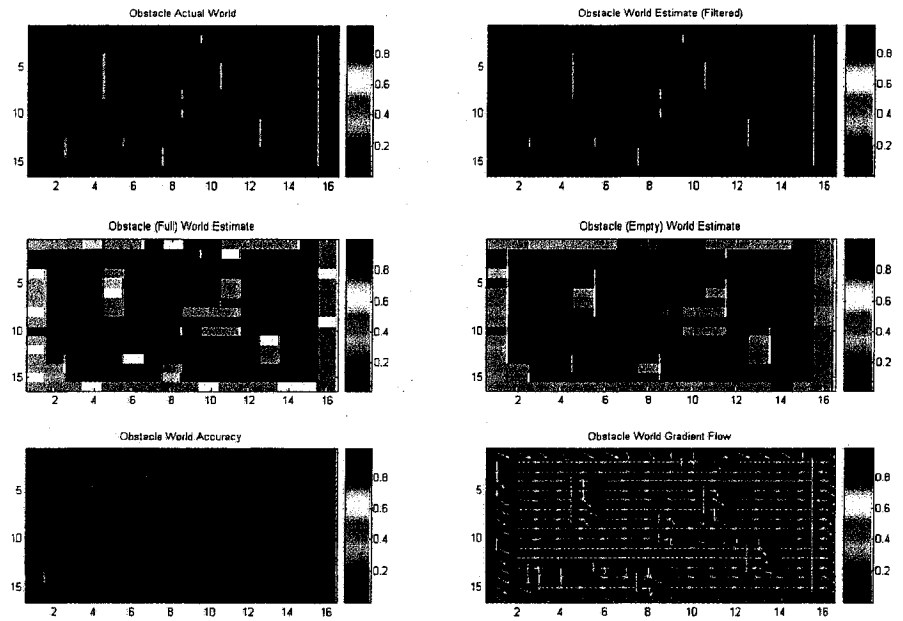


Figure 5.23: Environment mapping with noise case 4

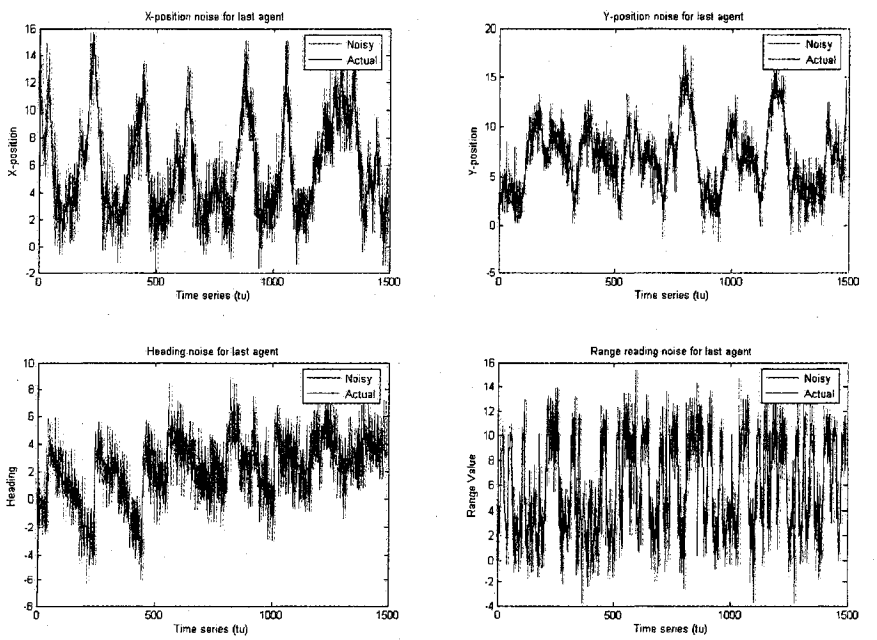


Figure 5.24: Noise signals for pose estimates and sensor readings (case 4)

5.1.6 Information Fusion Simulations

Since our quadtree data structure allows us to monitor multiple dimensions of the environment, let us take both obstacle detection and temperature monitoring and attempt to fuse the two together. We will use the transmitted information from one variable to the other to measure the correlation between the two environmental parameters. Mutual information provides us with a measure of fused information that one can gain about obstacles by observing the temperature. Since mutual information is symmetrical, the opposite also holds true (i.e. linking the temperature to obstacles). Note that the lower the mutual information, the more independent the parameters are. Figures 5.25 and 5.26 show the obstacle and temperature entropy landscapes, respectively. In this case, we have two obstacles and one high-temperature region.

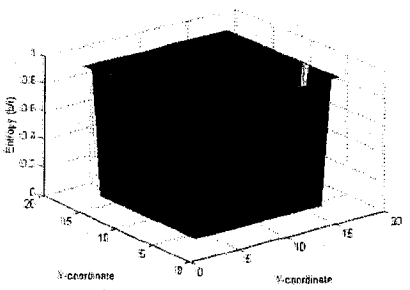


Figure 5.25: Obstacle landscape

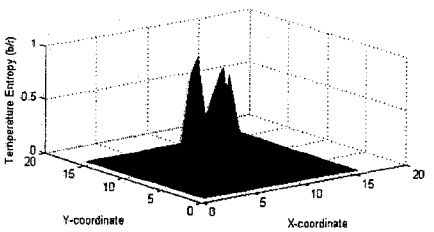


Figure 5.26: Temperature landscape

Graphing the mutual information of the two environmental parameters (Figure 5.27), we can clearly see that relations between both parameters are extracted and validated. One of the obstacles is causing the high temperature!

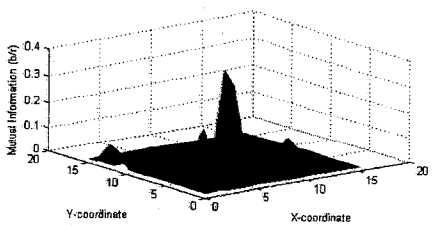


Figure 5.27: Mutual information landscape

5.2 Summary

This chapter has demonstrated the validity of the TIMM method, by presenting simulation scenarios that compare TIMM to previous solutions, namely the OGF and HIMM methods, in the context of map building through multi-agent entropy minimization. Four metrics were used in the comparison. As it concerns the grid storage size, it was found that the HIMM is four times as storage efficient as both OGF and TIMM. As it concerns the CV update time, it was found that the OGF method is the slowest, followed by the HIMM and TIMM methods. As it concerns the exploration strategy decision time step, it was found that the HIMM method is the slowest, followed by the OGF and TIMM methods. Finally, as it concerns the map extraction time, it was found that the TIMM method was the fastest, followed by the HIMM and OGF methods.

Simulations are also presented for accuracy purposes, showing the difference between the actual and estimated environment map outputs, as well as for noise immunity purposes, showing the robustness of the proposed method in the presence of additive white Gaussian noise, applied to the sensor readings and the position/orientation estimates. The chapter was concluded with a simulation of the mutual information metric and its application to sensor correlation.

Chapter 6

Experimental Setup and Results

Over the course of the project, numerous ISAs have been built to different specifications, starting with 2-wheel differential-steering ones, as well as synchro-drive ones. The main platform is either circular or hexagonal, and is approximately 25.4 cm in diameter, with a height of 10-20 cm, depending on the on-board sensors. Most of these agents are targeted for indoor applications, and can carry a 10 kg payload including batteries. The arsenal of sensors that the agents can carry includes video (web and moving cameras), ranging (IR and sonar), heading sensors (digital compass), UV, photo reflective and temperature sensors. The actuators include grippers and manipulator arms, as well as a piezo-buzzer. Both differential- and synchro-drive agents are able to rotate in place, and in a regular fashion, with a circle radius formed by a full rotation measuring 10 cm for the differential-drive agents and 20 cm for the synchro-drive agents. The agents are able to move at speeds nearing 35 cm/s on flat surfaces and are drivable using different means, including key commands, joystick guidance and control software program. The former two have already been demonstrated in [101] (see Figure 6.1 for a monitor snapshot of the GUI and the live video feedback), while the latter will be presented in this chapter. Most of the agents run on two standard 7.2 V 2000+ mAh NiMH batteries, that are intended for remote controlled toys, such as R/C cars and airplanes. Each battery is actually made up of six Class C 1.2 V batteries connected in series. A high-capacity trickle charger is provided so that each battery can be recharged to 95% of full capacity within 4 hours.

This chapter discusses the ISAs' hardware, software and communications details. This

chapter also presents and details the experimental testbed and the results obtained from experimental verification, as it concerns the four aforementioned metrics: grid storage size, CV update time, exploration strategy decision time step and map extraction time. Node pruning is discussed as an optimization technique for the grid storage size, while the quadtree-based data structure is shown to handle multi-resolution and multi-dimension views of the environment.

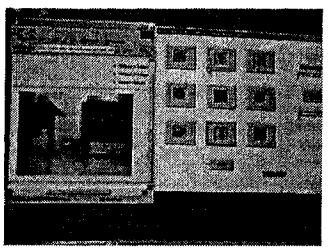


Figure 6.1: Key command control software with video feedback

6.1 On-Board Electronics

To support these operations, the ISA's hardware has to be flexible, as well as reliable, while its software has to be scaleable, as well as robust. After careful deliberations, it was chosen that the Altera Nios development kit [102] would be the ideal hardware platform, as it provides flexibility of design updates through its Stratix field-programmable gate array (FPGA), as well as reliability through its soft-core 32-bit reduced instruction set computer (RISC) Nios processor. However, we went through three major designs, the first based only on the Motorola HC9S12DP256B microcontroller [103] (ISA result shown in Figure 6.2, depicting a differential-drive two-wheel robotic platform with a wireless camera and an IR sensor), the second based on a combination of an Altera FPGA board and an HC9S12 microcontroller (ISA result shown in Figure 6.3, depicting a synchro-drive two-wheel robotic platform with an onboard manipulator arm, a camera and IR sensors), and the third and final one based only on the Altera Stratix-II FPGA board (ISA result shown in Figure 6.4, depicting a differential-drive two-wheel robotic platform with a digital compass, an IR sensor, a temperature sensor and a RF modem). The

robot controller designs allowed us more flexibility as we progressed, until we reached the final solution, that currently allows to add any digital interface in the FPGA, as well as any software interface in the soft-core microprocessor. The latter is a 50 MHz 32-bit RISC processor that can be instantiated more than once on the Stratix-II FPGA, easily provisioning for a multiprocessor system-on-chip. The robot controller also contains 16 MB of flash memory, 1 MB of static RAM, 16 MB of SDRAM, a CompactFlash connector for Type I CompactFlash cards and an on-board Ethernet MAC/PHY device. This board also serves as a development platform due to its on-board Mictor connectors for hardware and software debug and 5V-tolerant expansion/prototype headers for simple and fast sensor and experimental device additions.

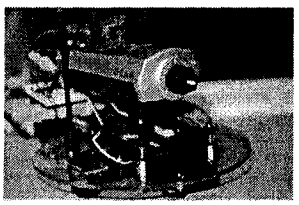


Figure 6.2: First design

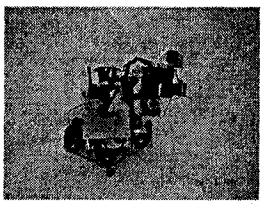


Figure 6.3: Second design

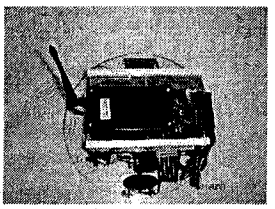


Figure 6.4: Third design

The ideal processor should possess the following characteristics: relatively fast response times, low power consumption and production cost, small footprint area, on-chip memory (cache, ROM and/or RAM cores), abundance of I/O capabilities, standard interfaces (serial, parallel, USB), robust instruction set architecture, availability of development tools, testability and reliability and industrial and academic support. Table 6.1, extracted from the author's invited talk [104], shows a comparison chart between all the available microcontroller boards and their associated embedded processor characteristics. It is imperative to remember that this is a physical system that employs computer control for a specific purpose and not for a general-purpose application (i.e. it is an embedded system), hence, after further study, it was found that the Stratix-II board with the Nios II microprocessor met most of these requirements.

Processor family	Intel 8051 μC	Motorola 68HC11 μC	Motorola 68HC12 μC	Motorola ColdFire μC	Motorola PowerPC μC	ARM μC	Atmel μC	Microchip μC	Altera Stratix-II μC
Processor architecture	RISC	CISC	CISC	CISC) (MCF5407)	RISC (MPC5500)	RISC (ARM7)	RISC (AVR)	RISC (PIC)	RISC (Nios-II)
Processor speed	12 MHz - 16 MHz	4 MHz	25 MHz	33 MHz - 333 MHz	Up to 300 MHz	50 MHz	Up to 16 MHz	Up to 40 MHz	Up to 50 MHz
ROM size	4 KB	8 KB - 12 KB	32 KB - 256 KB (multi-sector)	16 KB ICache, 8 KB DCache	4 MB Flash	40 KB - 192 KB Flash	Up to 128 KB Flash	Up to 512 bytes	On-chip configurable core
RAM size	128 bytes	256 bytes - 512 bytes	2 KB - 12 KB	4 KB SRAM	128 KB	4 KB SRAM	4 KB SRAM	Up to 368 bytes	16 MB
I/O capabilities	4 8-bit ports	5 8-bit ports	5 8-bit ports	16-bit ports	N/A	Up to 75 GPIO	Up to 53 GPIO	Up to 33 GPIO	Up to 41 GPIO
Interfaces	UART	UART, SPI, ADC	UART, SPI, SCI, ADC	UART, USART, I2C	None	UART	UART, SPI	UART, USB, I2C	UART, SPI, Ethernet
Data bus width	8-bits	8-bit 6800 or 6809 μP	16-bit HCS12 μP	32-bit MFL5xxx μP	32-bit MPC55xx μP	32-bit ARM7 μP	8-bit megaAVR μP	8-bit PIC16 family	32-bit Nios-II μP
Particulars	2 16-bit counters/timers	512 bytes of EEPROM	4 KB of EEPROM, PWM, BDLC, CAN bus	2 16-bit timers	MMU and DSP functionality	8-bit ADC, timers, PWM and watchdog	4 KB EEPROM, 10-bit ADC, PWM	8-bit ADC, 8-bit timer, comparator	Config. PWM, timers, EEPROM, DSP, RAM, 16 MB Flash

Table 6.1: Embedded processor comparison chart

6.2 Experimental Testbed

For our experimental setup, we decided to use our latest physical agent, the differential-drive Stratix-II hardware/software co-designed one previously shown in Figure 6.4. Using four of these agents (Figure 6.5), each with different sensory configurations, allowed us to test out our simulation results. The physical environment was a scale model of a hockey arena that the author constructed in preparation for a local IEEE robotics competition [105]. The environment can be seen in Figures 6.6 and 6.7. It is 198 *cm* (78 inch) long by 122 *cm* (48 inch) wide, with 15 *cm* (6 inch) radius curved corners. The peripheral walls are 2.5 *cm* (1 inch) in thickness and are white in colour, mainly for the latter's light reflexive properties. The inner periphery of the environment, accounting for the walls' thicknesses, is approximately 195 *cm* long by 119 *cm* wide. White-coloured blocks were used as obstacles and candles were used as hazards.

The environment was tessellated into 256 cells, each 12.19 *cm* long and 7.44 *cm* wide. After numerous experimentation, this was found to be the ideal tessellation cell count and dimension. Since our environmental data structure is based on a quadtree, hence requires a power of 2 tessellations, and since a 16-by-16 environment grid matched our simulation setup, we chose that many tessellations to cover the arena. Thus, the map resolution is approximately

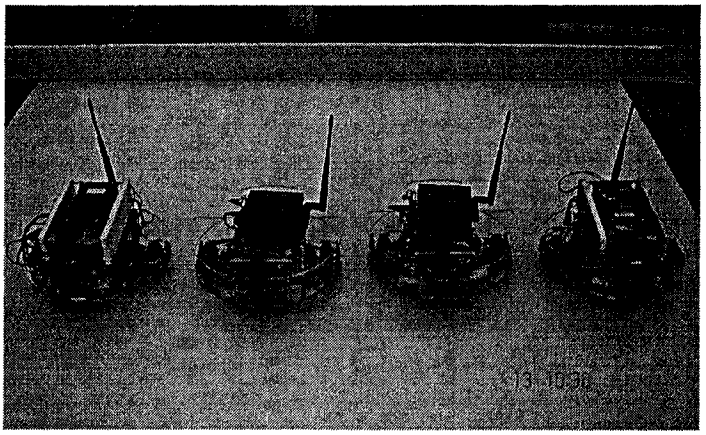


Figure 6.5: The four physical agents used in the final experiments

90.69 cm^2, while its measured accuracy is 5% of that range, or 4.53 cm^2.

6.3 Calibration Results

Calibration is the process by which an instrument's reading can be correlated to the actual value being measure. To establish the correct operation of our physical agent, some of its constituents, including the servo drive motors, the IR sensor and the compass sensor, required calibration. Their calibration details will be discussed next.

6.3.1 Servo Motor Calibration

As mentioned in section B.2.1, the servo motors used on-board the agents are PWM-driven. However, each motor's respective pulse widths varied due to their minute physical differences. Table 6.2 shows the calibration results for all four physical typical agents used in the experiments. Note that the servo motors were mounted symmetrically to each other; however, they are inverted, hence the same pulse width turns a motor in one direction and the other in the opposite direction. The following are the abbreviations of the 7 motor behaviors: BF = backward fast, BM, backward medium, BS = backward slow, S = Stop, FS = forward slow, FM = forward medium, and FF = forward fast.

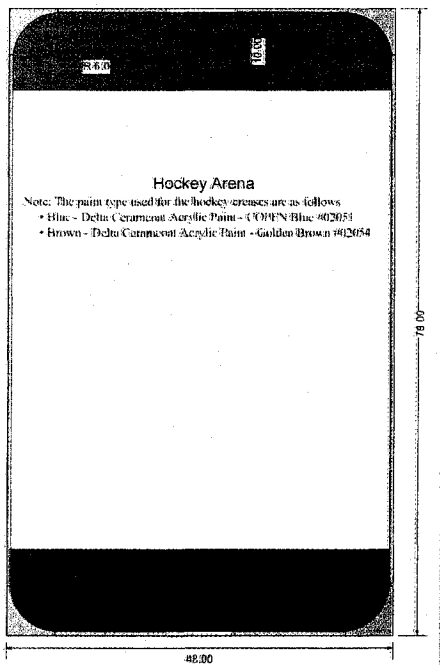


Figure 6.6: The physical environment, diagrammatic view

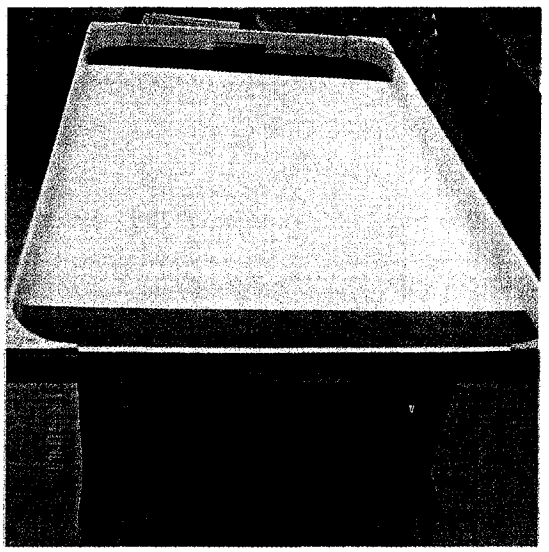


Figure 6.7: The physical environment, vertical view

Roboal-2A	Right Motor						
High time (ms)	1.274	1.331	1.356	1.374	1.413	1.429	1.597
Velocity (ticks/sec)	39.00	19.00	9.00	0.00	-11.00	-21.00	-39.00
Behavior	FF	FM	FS	S	BS	BM	BF
Roboal-2A	Left Motor						
High time (ms)	1.109	1.284	1.309	1.332	1.364	1.388	1.597
Velocity (ticks/sec)	-39.00	-19.00	-9.00	0.00	11.00	20.00	41.00
Behavior	BF	BM	BS	S	FS	FM	FF
Roboal-2B	Right Motor						
High time (ms)	N/A	1.316	1.341	1.362	1.392	1.419	N/A
Velocity (ticks/sec)	N/A	19.00	10.00	0.00	-10.00	-21.00	N/A
Behavior	FF	FM	FS	S	BS	BM	BF
Roboal-2B	Left Motor						
High time (ms)	N/A	1.262	1.287	1.308	1.350	1.379	N/A
Velocity (ticks/sec)	N/A	-20.00	-10.00	0.00	11.00	21.00	N/A
Behavior	BF	BM	BS	S	FS	FM	FF
Roboal-2C	Right Motor						
High time (ms)	N/A	1.326	1.350	1.366	1.401	1.425	N/A
Velocity (ticks/sec)	N/A	19.00	10.00	0.00	-10.00	-20.00	N/A
Behavior	FF	FM	FS	S	BS	BM	BF
Roboal-2C	Left Motor						
High time (ms)	N/A	1.210	1.294	1.316	1.368	1.395	N/A
Velocity (ticks/sec)	N/A	-21.00	-10.00	0.00	10.00	21.00	N/A
Behavior	BF	BM	BS	S	FS	FM	FF
Roboal-2D	Right Motor						
High time (ms)	1.242	1.298	1.321	1.342	1.372	1.396	1.509
Velocity (ticks/sec)	39.00	20.00	10.00	0.00	-11.00	-21.00	-41.00
Behavior	FF	FM	FS	S	BS	BM	BF
Roboal-2D	Left Motor						
High time (ms)	1.229	1.296	1.319	1.340	1.374	1.397	1.533
Velocity (ticks/sec)	-39.00	-19.00	-10.00	0.00	11.00	21.00	41.00
Behavior	BF	BM	BS	S	FS	FM	FF

Table 6.2: Servo motor calibration results

Shown in Figure 6.8 is the relationship between pulse width high time and the actual velocity, measured using the feedback wheel encoders in ticks per second. As can be seen, certain motors could not reach the required calibration velocities (e.g. Roboal-2C's right motor could not meet FF's requirement of 40 ticks/sec). The *X-like* pattern for each of the motors is indicative of their symmetrical, but inverted mounting positions.

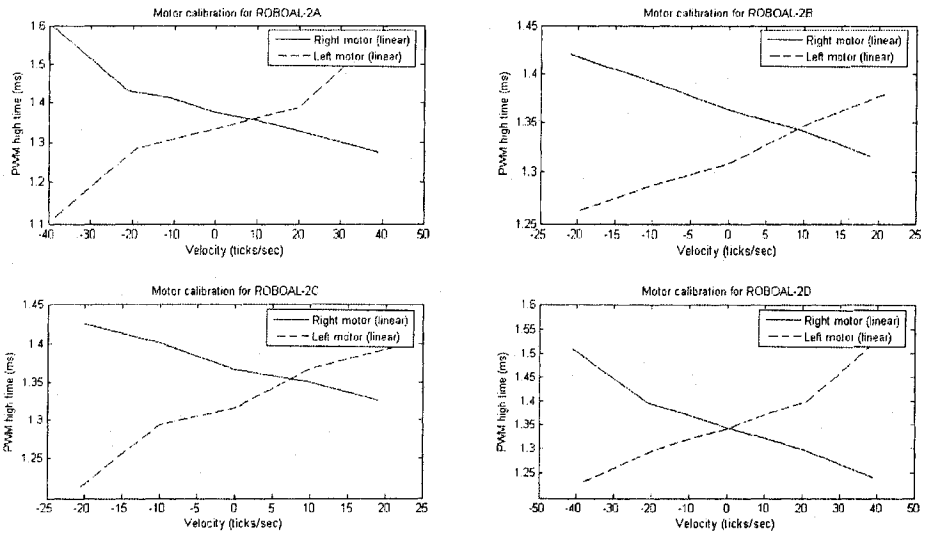


Figure 6.8: Servo motor calibration results (with linear fits)

After much experimentation, it was found that the best-fit equations were quartic polynomial ones. Figure 6.9 shows the quartic best-fit curves for each servo motor of the physical agent *2A*. The remaining figures for the calibration of the other agents are omitted; however, their calibration equations are presented below.

Equations 6.1 and 6.2 represent the high time (ms) as a function of velocity (ticks/sec) for Roboal-2A's left and right servo motors, respectively.

$$HT_{2A-L}(v) = 3.769e-010 * v^4 + 2.635e-006 * v^3 + 2.972e-006 * v^2 + 0.00186 * v + 1.333 \quad (6.1)$$

$$HT_{2A-R}(v) = 3.703e-008 * v^4 - 1.331e-006 * v^3 - 1.963e-005 * v^2 - 0.002104 * v + 1.38 \quad (6.2)$$

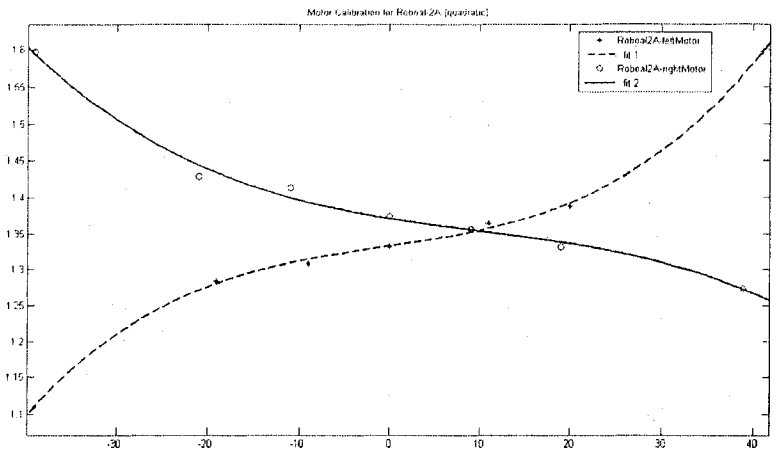


Figure 6.9: Servo motor (2A) calibration results (with quartic fits)

Equations 6.3 and 6.4 represent the high time (ms) as a function of velocity (ticks/sec) for Roboal-2B's left and right servo motors, respectively.

$$HT_{2B-L}(v) = -1.789e-007 * v^4 - 1.143e-007 * v^3 + 0.0001018 * v^2 + 0.00295 * v + 1.308 \quad (6.3)$$

$$HT_{2B-R}(v) = -1.268e-007 * v^4 - 3.729e-007 * v^3 + 5.768e-005 * v^2 - 0.002513 * v + 1.362 \quad (6.4)$$

Equations 6.5 and 6.6 represent the high time (ms) as a function of velocity (ticks/sec) for Roboal-2C's left and right servo motors, respectively.

$$HT_{2C-L}(v) = -5.297e-007 * v^4 - 2.067e-006 * v^3 + 0.000203 * v^2 + 0.003493 * v + 1.316 \quad (6.5)$$

$$HT_{2C-R}(v) = -2.618e-007 * v^4 - 2.368e-007 * v^3 + 0.0001212 * v^2 - 0.002526 * v + 1.366 \quad (6.6)$$

Equations 6.7 and 6.8 represent the high time (ms) as a function of velocity (ticks/sec) for Roboal-2D's left and right servo motors, respectively.

$$HT_{2D-L}(v) = 1.495e-008 * v^4 + 9.168e-007 * v^3 - 4.597e-006 * v^2 + 0.002239 * v + 1.343 \quad (6.7)$$

$$HT_{2D-R}(v) = 9.609e-009 * v^4 - 6.967e-007 * v^3 + 9.372e-007 * v^2 - 0.002154 * v + 1.344 \quad (6.8)$$

6.3.2 IR Range Sensor Calibration

In the case of the IR range sensors, a linearization formula is obtained, based on the inverse relationship of the sensor sheet, as demonstrated by its data sheet measuring output vs. object distance curve [106]. Again, we will only show the calibration results of the sensor on agent *2A* (see Figure 6.10), but the final results for all IR range sensors.

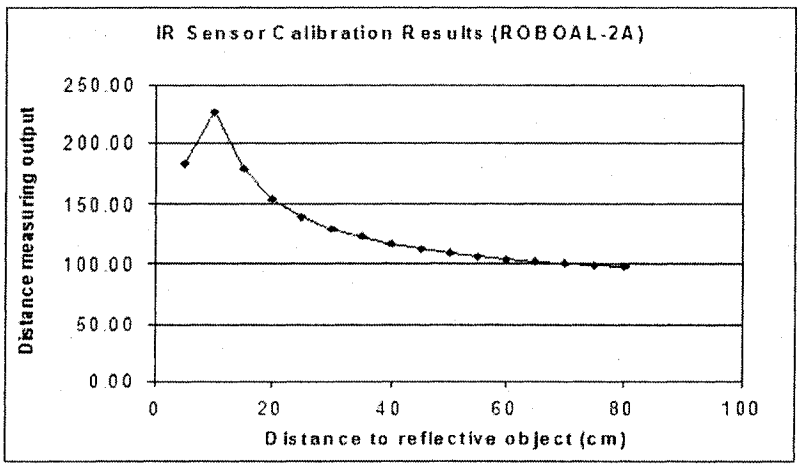


Figure 6.10: IR range sensor calibration results (2A)

The generic linearization formula for the Sharp GP2D02 IR range sensor is as follows:

$$D = \frac{K_g}{X - K_o} \quad (6.9)$$

where,

X is the output of the Sharp IR range sensor in decimal;

K_g is the gain constant that determines the shape of the inverse curve;

K_o is the offset constant that shifts the inverse curve up or down;

D is the distance to the reflective object in cm.

Following the derivation in [107], we have the following equations for the gain and offset constants:

$$K_g = \frac{D_1 D_2 (X_1 - X_2)}{D_2 - D_1} \tag{6.10}$$

$$K_o = \frac{D_1 X_1 - D_2 X_2}{D_1 - D_2} \tag{6.11}$$

where,

X_1, X_2 are two outputs of the Sharp IR range sensor in decimal; and

D_1, D_2 are two distance measurements in cm.

Table 6.3 shows the IR range sensor calibration results for the four physical agents.

6.3.3 Compass Sensor Calibration

The magnetic digital compass was calibrated using a magnetic needle compass and the procedure outlined in the component's data sheet, mainly by passing the compass through the four cardinal positions and registering calibration values in the compass. However, another calibration is required to map the obtained heading values to the global environment frame. Again, we will only show the calibration results of the sensor on agent *2A* (see Figure 6.11); however, we will present the final results for all compass sensors.

The generic mapping formula for the compass sensor in our physical environment is as follows:

$$\theta^R = \begin{cases} K_c - \theta^E, 0 \leq \theta^E \leq K_c \\ 360 + (K_c - \theta^E), K_c \leq \theta^E \leq 360 \end{cases} \tag{6.12}$$

where,

θ^E is the output of the compass sensor in degrees;

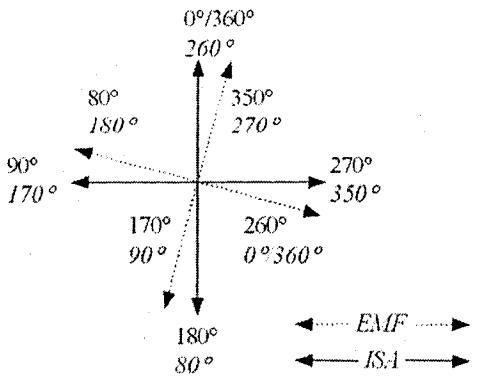


Figure 6.11: Compass sensor calibration results (2A)

θ^R is the actual heading in the global reference frame; and

K_c is the calibration constant for each compass.

Table 6.3 shows the compass sensor calibration results for the four physical agents.

Roboal-ID	K_o	K_g	K_c
2A	79.50	1475.00	260.0
2B	83.75	1412.50	250.0
2C	87.50	1325.00	247.0
2D	95.50	1325.00	263.0

Table 6.3: Compass sensor calibration results

6.4 Current Analysis

As previously mentioned, the final ISA design is the one shown in Figure 6.4. A typical agent consists of a Stratix-II board with an LCD character display, one wireless modem, one digital compass, one infrared range sensor, two servo motors and one infrared beacon. A fully-loaded agent consists of the same components as a typical agent; however, additionally includes a flame detector, six robotic arm motors, two line-tracking sensors, one microcontroller, one camera and two sonar range sensors. Fully-loaded agents were experimented with throughout the project, but it was decided to only include typical agents in the experimental setup, to provide

a proof-of-concept, while minimizing current draw. The final current analysis for both types of agents is shown in Table 6.4. As can be seen, the worst case current draw, for a typical agent, is approximately 2A, with an average current draw, per component, of approximately 250 mA.

6.5 Agent Hardware

This section will briefly describe the hardware systems integration that was involved with the final design of the physical agents. As shown in Appendix B, numerous robotic IP cores were developped using hardware description languages, to allow for a hardware/software co-design of the functionalities required by each agent. The real-time and high frequency functions were targetted for digital controllers, while the communication, navigation and control functions were targetted for software tasks. Figure 6.12 shows the high-level view of our Stratix-II FPGA. All the connected modules have been mentioned and reside outside of the FPGA, hence off-chip. Appendix D details the hardware integration of the physical agents.

6.6 Agent Software

As for the software environment, it was decided that a real-time operating system (RTOS), and an accompanying IP stack would be necessary for scalability and robustness of the access point. The following characterize our ideal operating system: multitasking and interrupt support, vast language and microprocessor support, ease of tool compatibility (compiler, assem-

Module	Current req. (mA)	Voltage req.	Model	Worst draw (mA)	Quantity	Full case (mA)	Quantity	Typical case (mA)
IR beacon	10-15	9Vunreg/5Vdc reg.	Robotmaker IRCF-L	15	1	15	1	15
Flame detector	30	5Vdc reg.	Hamamatsu UVTron	30	1	30	0	0
Servo motor	40-300	5-8Vdc	Futaba FP-S148	300	2	600	2	600
Robix arm motor	150	5-8Vdc	Hitec HS422	150	6	900	0	0
IR sensor	25	5Vdc reg.	Sharp GP2D02	25	1	25	1	25
Line tracking sensor	50-65	5-8Vdc	Lynxmotion TRA-01	65	2	130	0	0
Stratix-II board	540-926	17Vdc unreg.	Altera EP2S60	926	1	926	1	926
LCD character display	1.8	0.3-7.0V	Optrex DMC16207	1.8	1	1.8	1	1.8
HC12 board	200	5Vdc reg.	Adapt-9S12	200	1	200	0	0
Camera	200	5Vdc reg.	CMUCam-1	200	1	200	0	0
RF modem	400	7-18Vdc	Aerocomm CL 900MHz	400	1	400	1	400
Compass	15	5Vdc reg.	Devantech CMPS03	15	1	15	1	15
Sonar sensor	30-50	5Vdc reg.	Devantech SRF-04	50	2	100	0	0
				Totals	21	3542.9	8	1982.8

Table 6.4: Current analysis for both fully-loaded and typical agents

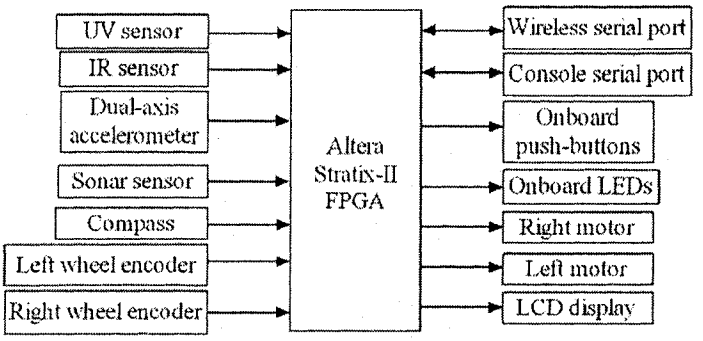


Figure 6.12: Altera Stratix-II interconnections

bler), wide array of services (queues, semaphores, timers), small area footprint (both program and data), scaleable design, availability of debugging tools, standards compatibility, extensive device driver support and industrial and academic support. Table 6.5, again extracted from [104], shows a comparison chart between all the available RTOSs and their associated characteristics. It is imperative to have a low interrupt latency, to allow for re-entrancy and to support pre-emptive scheduling, as all help us meet our real-time constraints when dealing with the agent's computational requirements, hence, after much deliberation, it was found that the uC/OS-II [108] RTOS met most of these requirements. As for the IP stack, lightweight IP (lwIP) [109] was also selected as the most suitable candidate.

The chosen RTOS allows us to build our higher level control software with the knowledge that this software can be easily ported to any other platform, as previously shown, when we discussed the three major designs: the application software was relatively unchanged, since the same RTOS was ported for all the utilized microprocessors. The chosen RTOS is open-source for academic applications, while the agent application software is open-source under the GPL. The entire software was written in ANSI C; however, C++ entry is supported as well.

6.6.1 SLAM software

Let us now briefly explain the agent's SLAM software structure that is depicted in Figure 6.13. The world model holds the agent's current view of the external environment surrounding it. It feeds the *control subsumption architecture*, that is in charge with reactively behaving to the world's sensory inputs by controlling the agent's motors. A navigation module decides how to freely navigate in the world, by localizing the agent (localizer module) and driving the *proportional-integrator controller (PIC)* in charge of keeping the physical agent moving at a constant velocity and in a straight forward direction. The utilized motors are actually servo motors, requiring the aforementioned *pulse-width modulation (PWM)* controllers. A separate, but analogous, *communication subsumption architecture* controls the on-board modems to react to C-MSF/D-MSF requests and acknowledges, as well as to broadcast sensory and positioning information.

The navigation functionality is shown in Figure 6.14. The initialization task (*initTask*) takes cares of starting up the four modules: the *encoder module* in charge of keeping track of both the right and left wheel incremental shaft encoders (*ENC_velocityTask()*), the *PIC module* in charge of maintaining straight-line motion at a constant velocity (*PIC_speedControlTask()*), the *PWM module* in charge of setting the actual calibrated motor speeds and the *localiza-*

RTOS	Mentor Graphics (Nucleus)	Cygnus Solutions (eCos)	Lynx Real-Time (LynxOS)	Microsoft Corp. (Windows CE)	QNX Software (QNX)	UC Berkeley (TinyOS)	Avocet Systems (AvSYS)	Micrium (uC/OS-II)
Target CPUs	68K, ARM, MIPS, x86, ColdFire, SPARC, H8, SH, TI DSPs	ARM, MIPS, MPC8xx, SPARC, Toshiba TX139	68K, MIPS, MPC8xx, x86, SPARC, PA-RISC	ARM, MIPS, PowerPC, SH, x86, Strong Arm, NEC	MIPS, MPC8xx, x86	Network processors	65816, 68HC08-16, 8051, Z8, Z80, 6809/01/03	ARM, AVR, Nios, x86, PowerPC, StrongARM, PIC-18xx, MIPS, 68K, MicroBlaze, Z80
Languages supported	C, C++, Java	Assembly, C, C++	Ada, asm, C/C++, Java, Fortran, Perl	Assembly, C, C++, Java	Assembly, C, C++, Java	nesC	C	C
ROM footprint	1 - Varies	1 - Varies	33 - 256	270 - 626	40 - Varies	3500	0.8 - 640	2048
RAM footprint	1 - Varies	1 - Varies	11 - 115	40 - 720	Varies	4500	0.8 - 640	200
Multitasking	Round robin, time slice, dynamic priorities	Round robin, time slice, fixed priorities	Round robin, time slice, fixed priorities	Round robin, time slice, dynamic priorities	Round robin, time slice, dynamic priorities	Priority scheduling	Time slice, priorities priorities	Round robin, time slice, fixed priorities
Licensing	Per license	Free	Per license	Per license	Per license	Free	Per license	Free for research
Particulars	POSIX, TCP/IP, source code	POSIX, TCP/IP, source code	POSIX, TCP/IP, source code	POSIX, TCP/IP, source code	POSIX, TCP/IP, source code	Event-based	POSIX, TCP/IP, source code	POSIX, TCP/IP, source code

Table 6.5: RTOS comparison chart

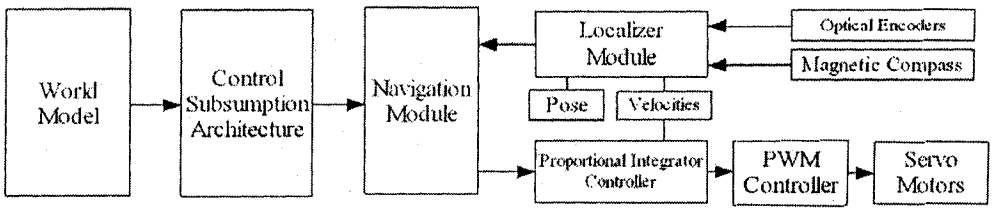


Figure 6.13: Agent SLAM software

tion sensor (LS) module in charge of periodically maintaining the estimate of the agent's pose (*LS_odometryTask()*). Simultaneously, the *ISA_arbitrateTask()* is periodically firing behaviors in the subsumption architectures, and setting the desired heading and speed according to each behavior, while the *ISA_coordUpdateTask()* is periodically reading the agent pose and transmitting that information to the available access points.

The control subsumption architecture is shown in Figure 6.15 and contains six behaviors that, if fired, subsume the lower priority behaviors, and control the servo motors accordingly. An arbitration task periodically wakes up and decides which behavior fires, depending on the current sensor inputs. The latter task sleeps for 10 *ms*, hence providing the agent with a sampling frequency of 100 *Hz*. The realized control behaviors are:

(1) **Escape** This behavior fires if a bump is felt by the dual-axis accelerometer (**this behavior has been removed from the final design software**);

(2) **Avoid** This behavior fires if an object is detected to be too close to the agent;

(3) **Contain** This behavior fires if an environmental sensor value increases beyond a certain threshold;

(4) **Drive** This behavior fires whenever a wheel encoder or digital compass sensor value changes, and is the main driving behavior of the agent, allowing it to reach its goals;

(5) **Explore** This behavior fires if there are additional unexplored regions in the environment. It consists of randomly moving the agent within the environment; and

(6) **Monitor** This behavior is the default one, and is fired if no other higher priority behavior fires. It allows the agent to drive back and forth within a certain path to monitor its local environment.

The communication subsumption architecture is shown in Figure 6.16 and contains four behaviors that, if fired, subsume the lower priority behaviors, and control the wireless modem accordingly. The two lower priority behaviors, *Request* and *Acknowledge*, only appear in the distributed scenarios. An arbitration task periodically wakes up and decides which behavior fires, depending on the current sensor inputs. The realized communication behaviors are:

(1) **Localize** This behavior fires if a change occurs in the agent's pose;

(2) **Announce** This behavior fires if a change occurs in the agent's sensors;

(3) **Request** This behavior fires when additional mapping or status information is required by the agent; and

(4) **Acknowledge** This behavior fires when a request has been processed and accepted.

Let us now explore the data flow diagram (Figure 6.17) showing the networking tasks involved in setting up each agent, as well as each access point (presented in the next section):

Initialization Task (not shown) Initializes the operating system data structures and creates the other tasks

LWIP Rx Ethernet Task Handles the Ethernet frames and routes them to the UDP or TCP layer accordingly

LWIP TCP/IP Task Handles the TCP packets and routes them to the appropriate application that requested the packet or to the data link layer if a packet is being sent out

NETUTILS DHCP Timeout Task Sets a static IP address after two minutes if lwIP has not been able to set a dynamic IP address due to lack of a successful DHCP server response

NETUTILS Get System Time Task Attempts synchronization with the access point's daytime server at IP 10.0.0.10, TCP port 13 for 5 seconds, and defaults to last previously known time if it fails to contact the time server

ISA Clock Task Displays the time of day clock every 1 second

ISA Clock Server Task Updates the time of day clock every 1 second

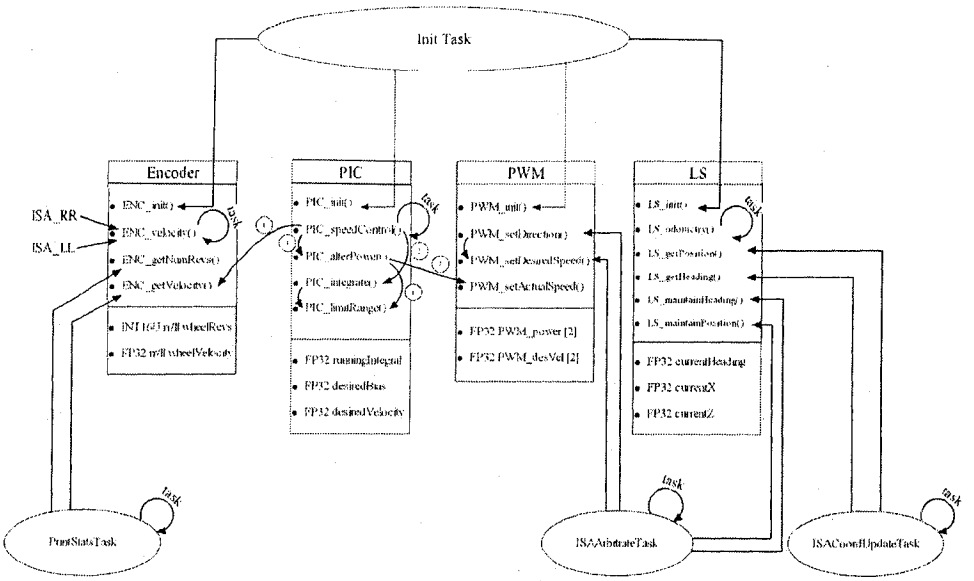


Figure 6.14: Agent navigation software

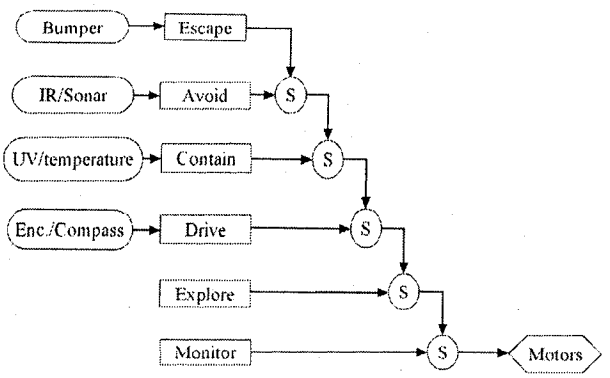


Figure 6.15: Control subsumption architecture

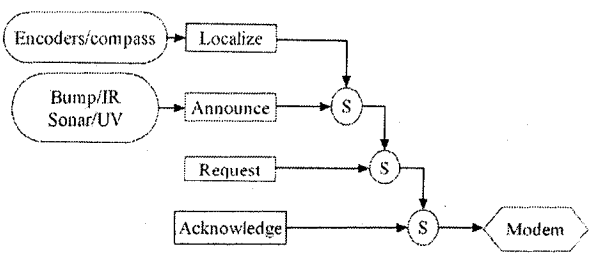


Figure 6.16: Communication subsumption architecture

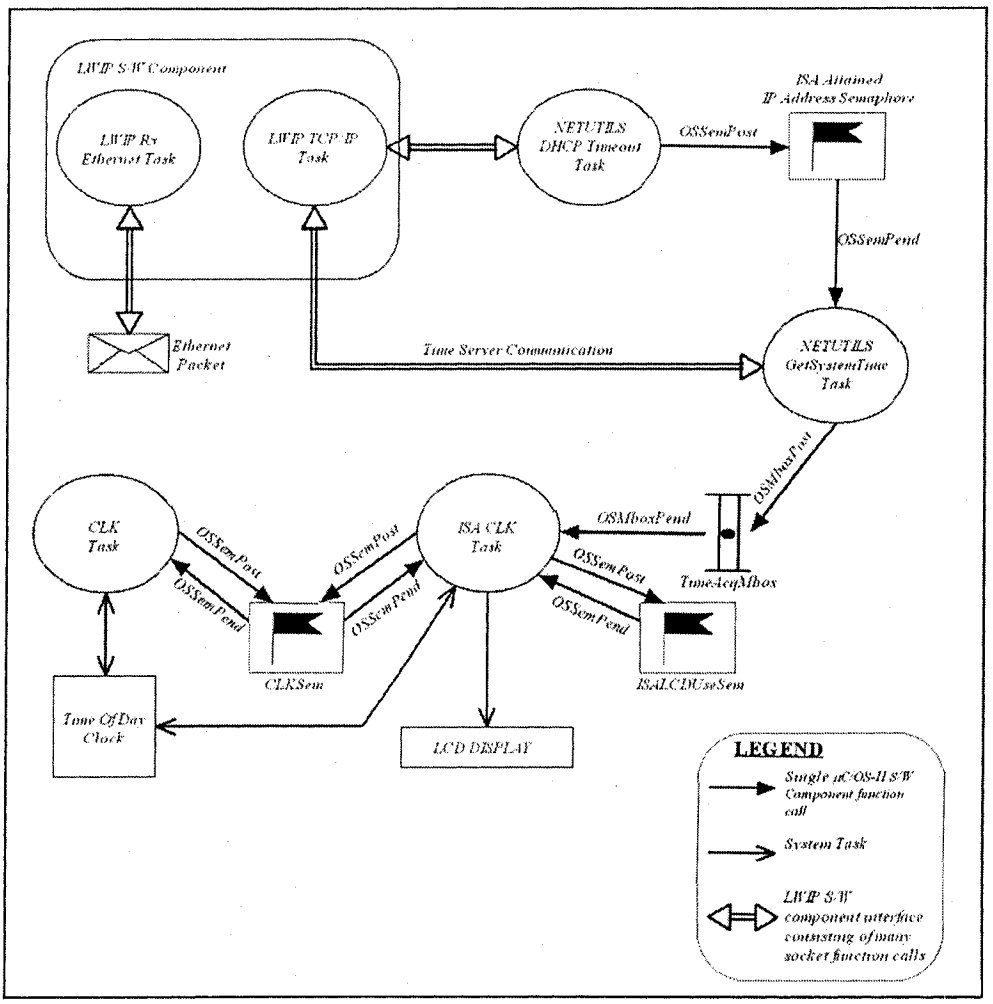


Figure 6.17: ISA/EAP dataflow diagram

6.6.2 Application profiling

While developping the agent's application software, it was necessary to maintain a decent RTOS utilization, so as to allow the high, as well as low, priority tasks a fair chance at the processor. To better measure this utilization, we have inherited and updated a set of Python scripts [110], that allow the user to monitor a formatted serial output and graph the task utilization in terms of the number of times the task ran, the total number of clock cycles it has needed for those runs, and finally, the task's cumulative total execution time. There are over 30 tasks that are running in the application; however, we track 17 of them, since the others either delete themselves after an initialization (e.g. *initTask*) or block waiting for some major event to occur (e.g. watchdog tasks).

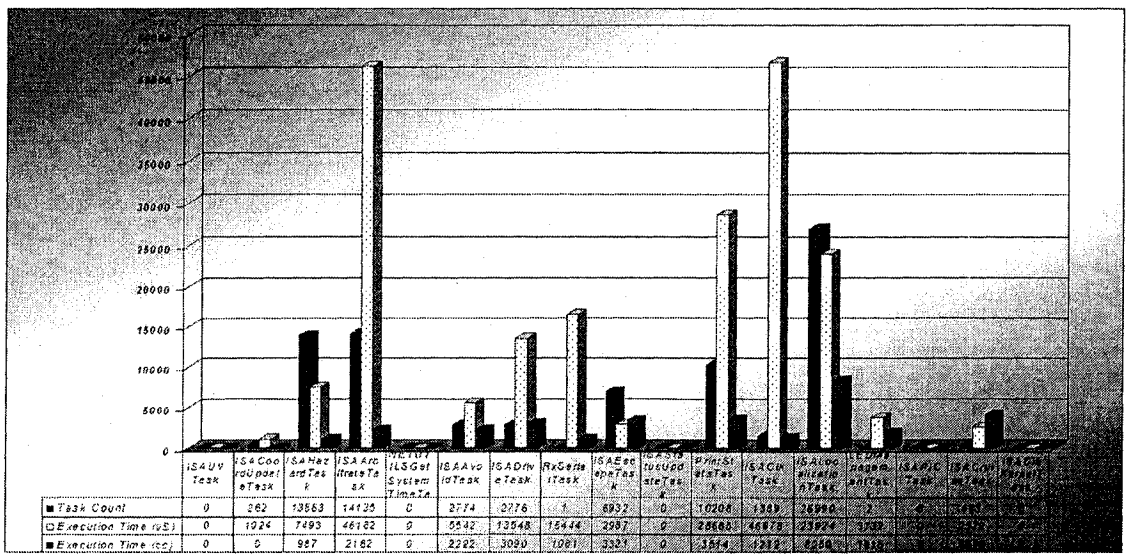


Figure 6.18: Agent application profiling

As can be seen in Figure 6.18, the two computationally intensive tasks are *ISAArbitrateTask* and *ISAClkTask*. Both have high execution times due to both their high priority and frequency. The former is the arbitrator task for the subsumption architectures, while the latter is the time-of-day update task, hence they both need to run often and quickly. Next up comes the *PrintStatsTask*, that is only enabled during application profiling, and hence is removed when the

agents are online. The next major computationally intensive task is *ISALocalizationTask*, that is in charge of maintaining the pose of the agent. This task runs frequently and has a medium priority, as it requires the processor to keep a stable pose estimate for the agent. After this task, it can be seen that all the other tasks receive a fair chance to run: this includes the subsumption behavior tasks, the socket communication tasks, as well as the system management tasks.

6.7 Agent Communication

The ISAs form a mobile ad-hoc network, and communicate using our devised *Intelligent Robotic Communication (InteRCom)* protocol. Figure 6.19 shows the DISS network architecture, including the multiple *access points (APs)* that are required to translate the proprietary InteRCom network protocol to widely used ones such as Ethernet, Bluetooth or Wi-Fi. Each access point acts as a network address translator (NAT), to allow each ISA to have access to the Internet, but also to shield the ISAs from unidentified, and potentially harmful, users. The latter is analogous to NAT software that is used to shield LAN computers from malicious users and/or code modules that are roaming the Internet. In this thesis, we have concentrated on the *Ethernet Access Point (EAP)*, as the IEEE 802.3 standard is the most accessible and cost-effective medium of communications for the monitored environment.

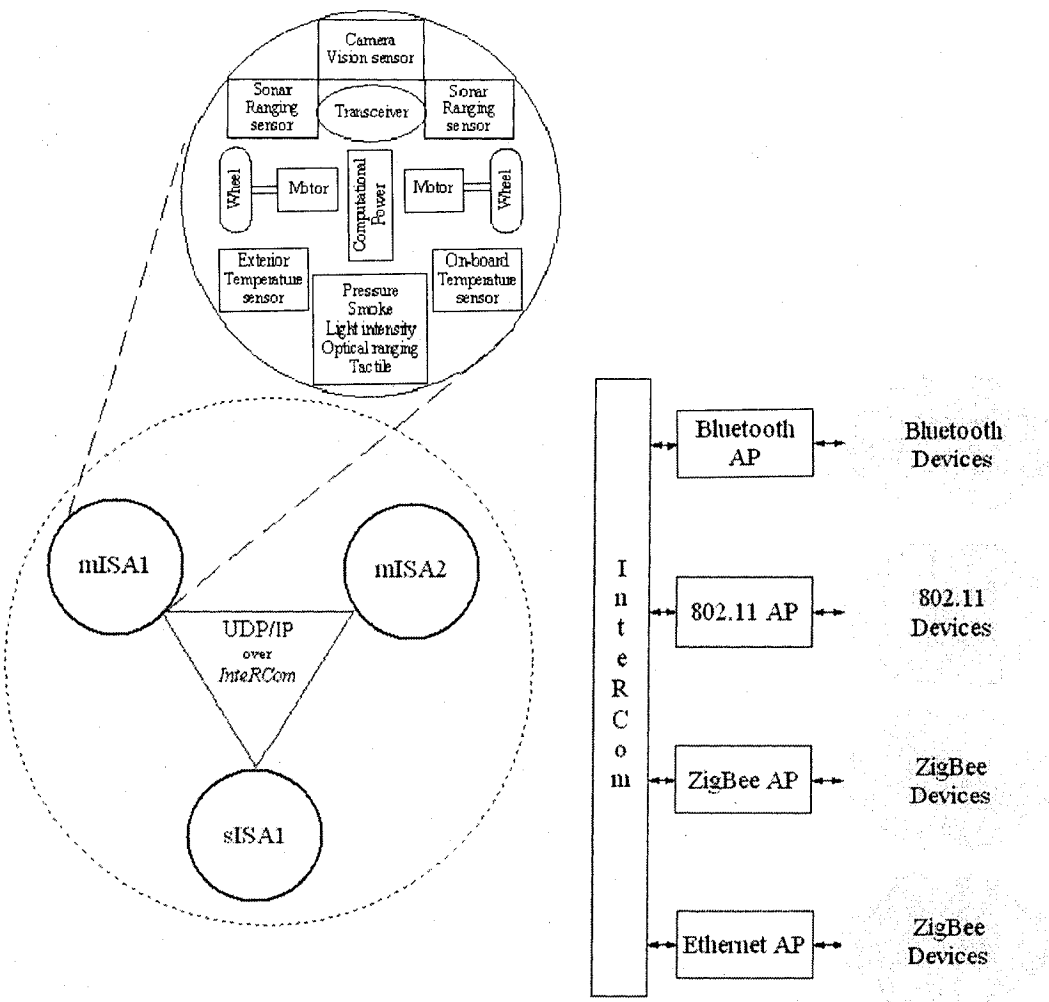


Figure 6.19: DISS network architecture

Wireless technology	Bluetooth	IrDA	IEEE 802.15.3a Ultrawideband (UWB)	IEEE 802.11a	IEEE 802.11b (Wi-Fi)	IEEE 802.11g	IEEE 802.15.4 (ZigBee)
Data rate (Mb/s)	1-2	4	100-500	54	11	54	250 kbps and 20 kbps
Output power (mW)	100	100 mW/sr	1	40-800	200	65	30
Range (meters)	100	1-2	10	20	100	50	30
Frequency band	2.4 GHz	Infrared	3.1-10.6 GHz	5 GHz	2.4 GHz	2.4 GHz	2.4 GHz and 868/915 MHz
Comments	7 active nodes	Very short-range	Low power, short-range applications	Wireless LANs with high data rate	Wireless LANs with low data rate	Wireless LANs with lower power	Low duty-cycle applications

Table 6.6: Wireless technology comparison chart

However, we have performed a study to characterize our ideal wireless technology: imperative low power utilization, simple transceiver circuitries, resiliency to multipath effects, worldwide medium availability, standards compatibility, freely licensable, medium to wide range operation, respectable data transmission rate and industrial and academic support. Table 6.6, also extracted from [104], shows a comparison chart between all the available wireless technologies and their associated characteristics. Due to the limited power supply, researchers are trying to combine all three components (sensing, processing, and communicating) into tiny low-power, low-cost units. After much deliberation, it was found that the IEEE 802.11b met most of these requirements; however, due to its high-cost and power requirements, it was decided, instead, to use a radio modem that has a wide transmission range and lower power dissipation.

Let us now turn our attention to the communications protocol. InteRCom is a cost-effective protocol that attempts to meet the policies set forth in [111]. Those policies are summarized below:

- Connection establishment and resource allocation;
- Quality-of-service capable sensor traffic delivery;
- "Intelligent networking" on distributed architectures; and
- Ad-hoc networking infrastructure support.

The protocol was designed to be scaleable and robust. It is scaleable because it is extremely easy to incorporate heterogeneous sensor, as well as haptic feedback, information

through InteRCom's packet structure. The protocol is robust because it is built on top of the UDP/IP stack that was aforementioned, and it adds error checking at the application level, to counter the no reliability constraint of the UDP network layer protocol. Figure 6.20 shows InteRCom's packet structure. InteRCom currently operates over an augmented wireless serial line IP (SLIP) protocol that allows for multiple devices to communicate by sharing a common wireless channel. This is accomplished through the additions of the InteRCom header and trailer to every IP datagram. As can be seen in the figure, the packet is subdivided into 5 main fields:

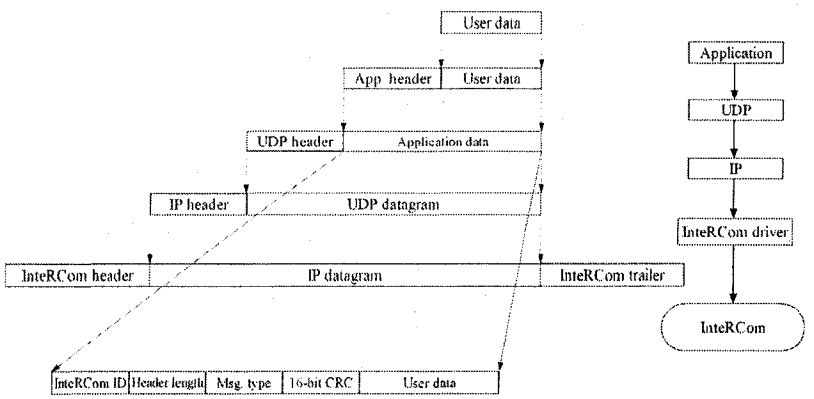


Figure 6.20: InteRCom packet structure

- *InteRCom ID*: This 8-bit value is used to uniquely identify the communicating device;
- *Header length*: This 8-bit value is used to indicate the header length in bytes;
- *Message type*: This 8-bit value is used to indicate the type of message being sent (more on this in the next paragraph);
- *CRC*: This 16-bit value is a cyclic redundancy check that was performed by the transmitting device on the entire application packet; and
- *User data*: This is a dynamic length value, depending on the number of parameters that is being sent. Typically this is a 96-bit field as ternary floating-point values are sent, either the agent pose or its sensory information.

We have experimented by employing the ABACOM Technologies' [112] 4800 bps/433.92 MHz/half-duplex AM transceiver modules called RTL-DATA-SAW, along with the vendor's proprietary DPC-64 data encoder and decoder. Each ISA and EAP was equipped with such a combination, along with the InteRCom drivers to enable UDP/IP communication over wireless SLIP. Other transceivers that have been used are Laipac's [113] 1 Mbps/2.4 GHz/full-duplex ShockBurst transceiver modules called TRF-2.4G, as well as Radiotronix' [114] 19.2 kbps/902-928 MHz/full-duplex wireless transceiver modules called EWM-900-FDTC. We have finally settled on the Aerocomm [115] CL4490-1000 RF serial data device operating in the 902-928 MHz spectrum (FHSS), with data rates up to 115.2 kbps. The output power is 1000 mW and the transmission range is 450 m indoors, and up to 32 km line-of-sight outdoors. The device's power dissipation is rated at 400 mA when transmitting and 40 mA when receiving, while the module weighs about 170 g.

6.7.1 EAP Software

To allow for a multi-agent and multi-user environment, it is necessary to design in listening server applications that can service many requests transparently. For this purpose, five such servers were realized (refer to Figure 6.21 for a graphical description of the involved software modules):

- **AP-monitor server**: Resides on the AP, and serves any ISA, located in its vicinity, trying to send or receive information to or from the outside world;
- **AP-user server**: Resides on the AP, and serves any human user attempting to post an address-centric query about the monitored environment;
- **AP-AP server**: Resides on the AP, and serves any fellow AP attempting to post or reply queries about a "hidden" (to the initiating AP) portion of the monitored environment. The queries could originate from vicinity ISAs or the AP itself; " ISA monitor server:

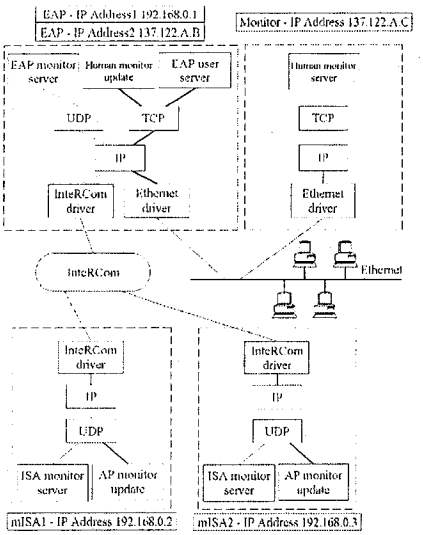


Figure 6.21: DISS server breakdown

Resides on the ISA, and serves any human monitor or fellow ISA attempting to post a data-centric query about the monitored environment; and

- **Human monitor server**: Resides on a networked computer, and serves any AP attempting to update the global model through periodic updates about the state of the monitored environment.

Furthermore, six types of conversations have been defined. Each of these conversations, consisting of two speakers, a protocol, and a dialog, permits information exchange by utilizing one of the four servers aforementioned. The conversations types are listed and described below:

- **(Type I) Human user ↔ EAP**: The human user requests address-centric information from the EAP about the latter's monitored environment (e.g. "provide me with the temperature in room 1010");

- **(Type II) Human monitor ↔ EAP**: human monitor requests data-centric information from the EAP about the latter's monitored environment (e.g. "provide me with all the agents' coordinates and sensor/status information that are in your vicinity");

- **(Type III) EAP ↔ R-ISA**: The sensor/status and coordinate information are exchanged between the two speakers in response to a human monitor/user request or as an update from a R-ISA;
- **(Type IV) Human monitor ↔ R-ISA**: The human monitor would like to "lock onto" a particular R-ISA to closely monitor and/or control its movements and actions (similar to *robotting into* the agent);
- **(Type V) R-ISA ↔ R-ISA**: The R-ISA requests information from another R-ISA in its vicinity. The information could be sensory- (e.g. "what is the current pressure ?") or behaviorally-driven (e.g. "can anyone help me to cross this bridge ?"); and
- **(Type VI) EAP ↔ EAP**: The EAP requests for data-centric information from another EAP, on the behalf of an R-ISA in the former's vicinity or to better synthesize the former's partial environment model.

It is important to note that the difference between a human monitor and a human user lies in their respective privileges: a human monitor is seen as a person who will not only remotely monitor the environment, but also actuate upon it, whilst a human user is seen as a person who is only interested in finding out status information from the environment, such as temperature, pressure, humidity and smoke detection. It can be seen now why a **Human user ↔ R-ISA** conversation cannot be allowed: the EAP's NAT functionality does not allow it! Moreover, a **Human user ↔ Human monitor** conversation presently does not make sense, hence is assumed to be non-existent in the architecture.

The EAP's networking software is similar to that of an agent; however, its *NETUTILS Get System Time Task* attempts synchronization with the NIST time server at time-a.nist.gov (IP 129.6.15.28, TCP port 13) for 5 seconds, and defaults to the last previously known time if it fails to contact the time server. The EAP then launches a daytime daemon for all the agents to synchronize to, to maintain the same system time between the EAP and the ISAs that are within its antenna proximity and are registered with it. One other important feature of the EAP is to

be able to handle simultaneous InteRCom connections. This was done by creating a server that listens to incoming connection requests, and then transfers the socket communication to a freely available socket, and goes back to listening for additional connection requests. This concept is demonstrated in Figure 6.22, that will be elaborated upon later.

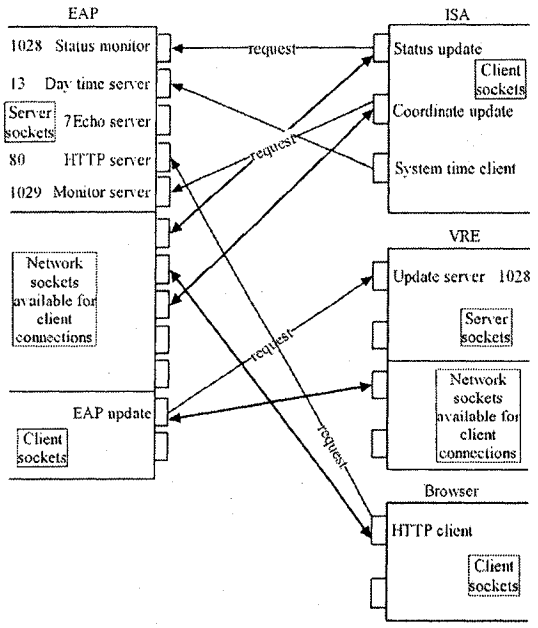


Figure 6.22: Network socket communications

Since the EAP monitors its registered agents, a monitor server is spawned at startup, that listens for InteRCom status and coordinates connection requests. The mobile agents proceed to connect to the respective servers, switch to an available socket and send status and coodinate information back to the access point. One major problem that was faced early on was that certain agents would wander outside the EAP's antenna reception range, or would stop communicating with the EAP on a particular socket. To resolve this issue, a watchdog task has been created, that runs every time a watchdog timer expires, and kills any such rogue InteRCom connections by closing the socket and killing the child task. The latter is the actual task that handles the ISA-EAP communication on a different socket from that of the monitor server. Let us explore the data flow diagram (Figure 6.23) showing the watchdog timer task involved in removing rogue

connections:

Monitor Server Task Listens on ports 1028 and 1029 for InteRCom connection requests and handles them by spawning children task and switching the communication to a freely available socket from the network socket pool

Monitor Child Task Communicates with a particular ISA for the duration of the latter's existence. Handles both coordinate and status packets

Monitor Watchdog Task Receives task id and last activity time from a queue and proceeds to kill any tasks that have not reported for a threshold period of time

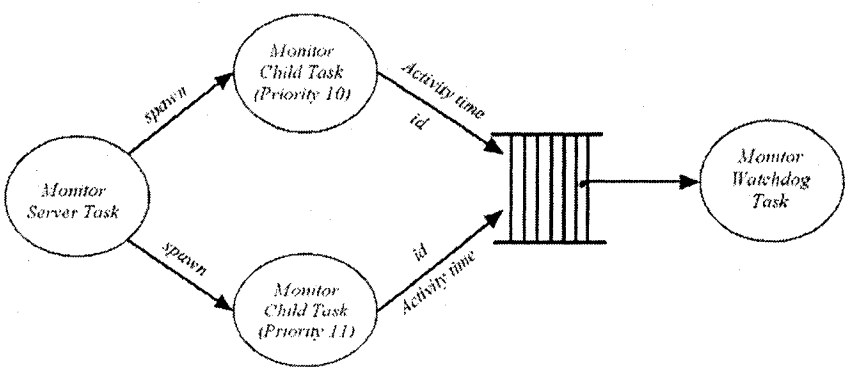


Figure 6.23: Watchdog dataflow diagram

In realizing type I and II conversations, the EAP has to communicate with the human monitor server and update the latter with the coordinate and status information present in its local tables. Figure 6.24 shows the sequence diagram involved in these conversations. Once the EAP's update client has connected with the monitor server (port 1028), it issues a server initialization request (*Init Server*), that is acknowledged by the monitor server, informing the EAP that a connection has been established and the server is ready to accept its packets. The EAP proceeds to upload the coordinate and status information of all registered agents, and receive new target poses from the monitor server, that it stores in its local tables. The EAP can

break the connection with the server by issuing a *Stop Server* command, whence the server goes back to listen for additional EAP commands.

Finally, as shown in Figure 6.22, the EAP also starts an HTTP server at TCP port 80, capable of handling up to six simultaneous browser client connections. Currently, two HTML pages are serviced from the EAP, a home page and a live page. The former (Figure 6.25) is a static page that offers the user with a link to the live page, links to other EAP home pages, and links to related Internet web sites. The home page consists of 1423 ASCII bytes and two images, one for the background at 2504 bytes and another of a robot at 34942 bytes.

The live page (Figure 6.26), on the other hand, is a dynamic HTML page, that uses a meta tag to command the browser to refresh every 5 seconds and not to reload the page from the browser cache. The command, that is placed in the HTML header, is as follows:

```
<META HTTP-EQUIV="refresh" CONTENT="5; URL=smrLab.html">
```

The live page consists of 1096 ASCII bytes. It is worth noting that the EAP's HTTP receive buffer size is 2048 bytes, while its transmit buffer size is 8192 bytes.

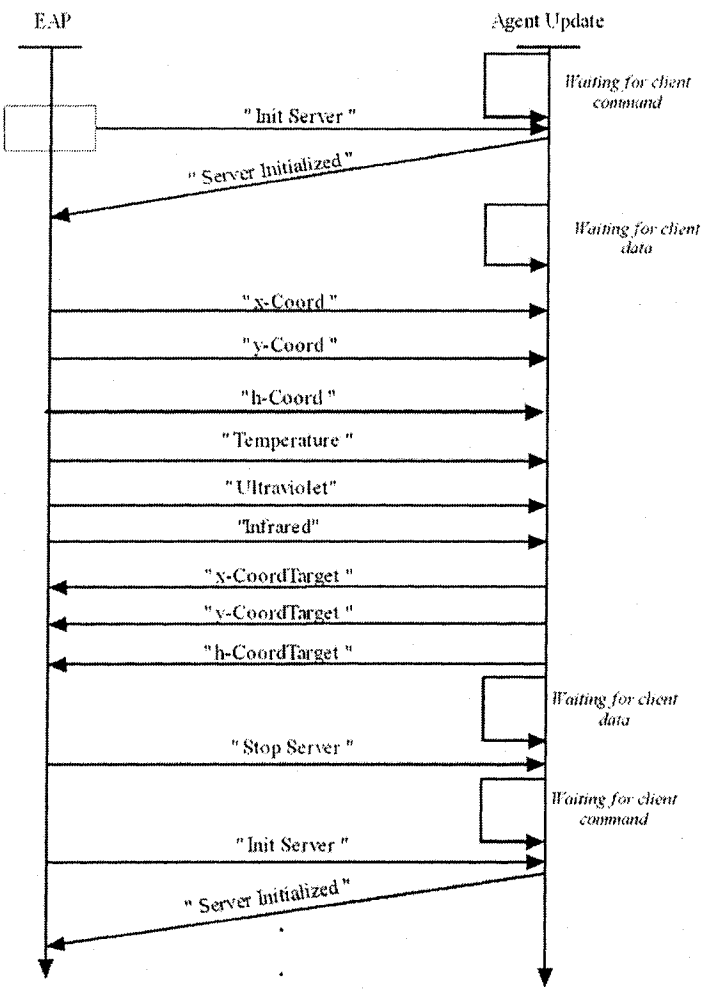
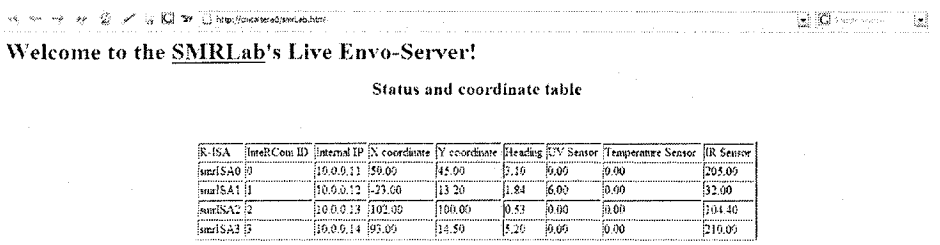


Figure 6.24: Agent update sequence diagram

Figure 6.25: EAP's HTTP server home page



R-ISA	IntelRCom ID	Internal IP	X coordinate	Y coordinate	Heading	UV Sensor	Temperature Sensor	IR Sensor
smrlSA0	0	10.0.0.11	59.00	45.00	2.16	6.00	0.00	205.00
smrlSA1	1	10.0.0.12	-23.60	13.20	1.84	6.00	0.00	32.00
smrlSA2	2	10.0.0.13	102.00	100.00	0.53	0.00	0.00	104.40
smrlSA3	3	10.0.0.14	95.00	14.50	5.20	0.00	0.00	210.00

Figure 6.26: EAP's HTTP server live page

6.8 Experimental Verification

To verify our methodology, we ran our entire system with up to four physical agents; however, we included a few more logical agents. Since we cannot physically build as many agents as we would like, we have written an entire ISA emulator that can run on any x86-based computer. The emulator behaves exactly as an ISA would, updating the EAP with coordinate and status information through a physical wireless communications module; however, the data that it sends is not real, but is generated by the emulator, and not by real sensors. This does provide us with the option to test the limitations of the wireless protocol, the fidelity of the synthesized VRME, as well as the cooperation amongst field agents. Moreover, the emulator is considered a first generation attempt at an static ISA, as the computers themselves are not mobile. A screenshot of the emulator is shown in Figure 6.27.

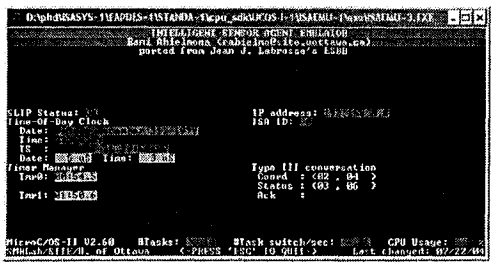


Figure 6.27: ISA emulator snapshot

The experimental setup is shown in Figure 6.28, with a sample scenario that the agents were monitoring. Lit candles were used to indicate hazards, while objects were used to indicate obstacles. The four physical agents proceed to map the environment using the TIMM methodology, and produce a VRME output such as the one shown in Figure 6.29. The obstacles were modeled as grey vertical strips, while hazards were modeled as red horizontal strips. Note that an in-depth discussion of the VRME will be presented in chapter 7; however, the image outputs are presented here for experimental verification purposes.

Shown in Figure 6.30 is the path taken by one physical agent as it traverses the envi-

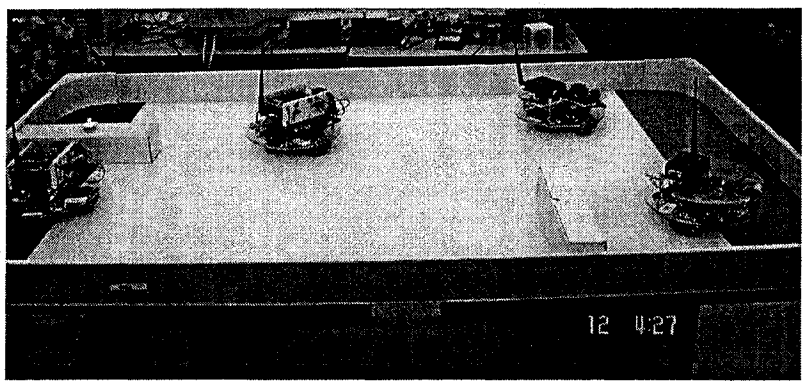


Figure 6.28: ISA experimental snapshot

ronment, while mapping and monitoring it. The path cells are modeled as green horizontal strips (a different green hue is used for each agent), and were placed at each coordinate in the environment whenever an agent sent an InteRCom coordinate update packet to the EAP.

Shown in figures 6.31 and 6.32 is another run through the world with a different setup, adding more hazards this time, and using less agents (2 physical ones).

Shown in Figure 6.33 is the occupancy entropic grid of the environment, with each cell modelled as starting with a maximum entropy value, and is decreased by the agents' collaborative efforts. A cell's height and hue are modified proportionally to its entropy value. Hence, it can be seen that there are *floating cells*, indicating that those parts of the environment have either not been explored yet, or are not explorable at all. The end result, of course, is the flattening of all entropic cells, resulting in a minimization of environmental entropy. These graphs are built in real-time and from real-world data, and validate the ones shown in simulation in chapter 5. Different landscapes can be visualized, including in our case, occupancy and UV entropy landscapes, as well as their joint entropy landscape.

Presented in Figure 6.34 is the mutual information landscape, and analogous to Figure 5.27, it demonstrates the learned part of the world as floating cells, indicating relationships between the environmental parameters being extracted, and the uninformative part in grey or white, indicating no direct relationship between one variable and the other. Note that certain

cells have negative values of mutual information, that theoretically should not occur, as non-negative mutual information is not defined. This tends to suggest some level of mutual exclusiveness in that the two events (in this case, occupancy and UV) occur together less often than chance. These should be taken to zero, whenever they occur.

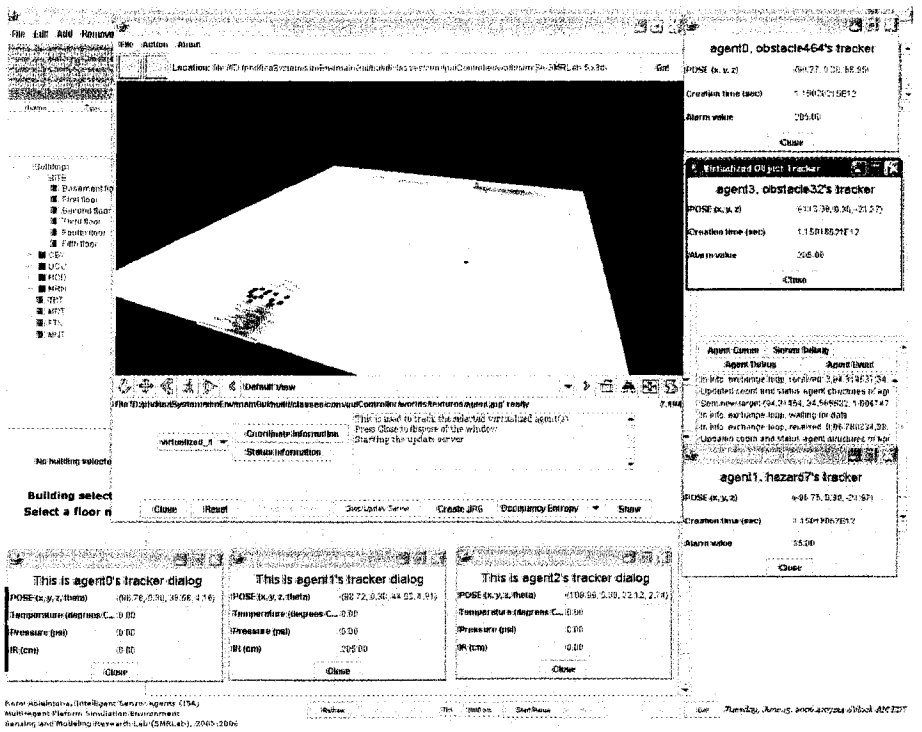


Figure 6.29: VRME experimental snapshot

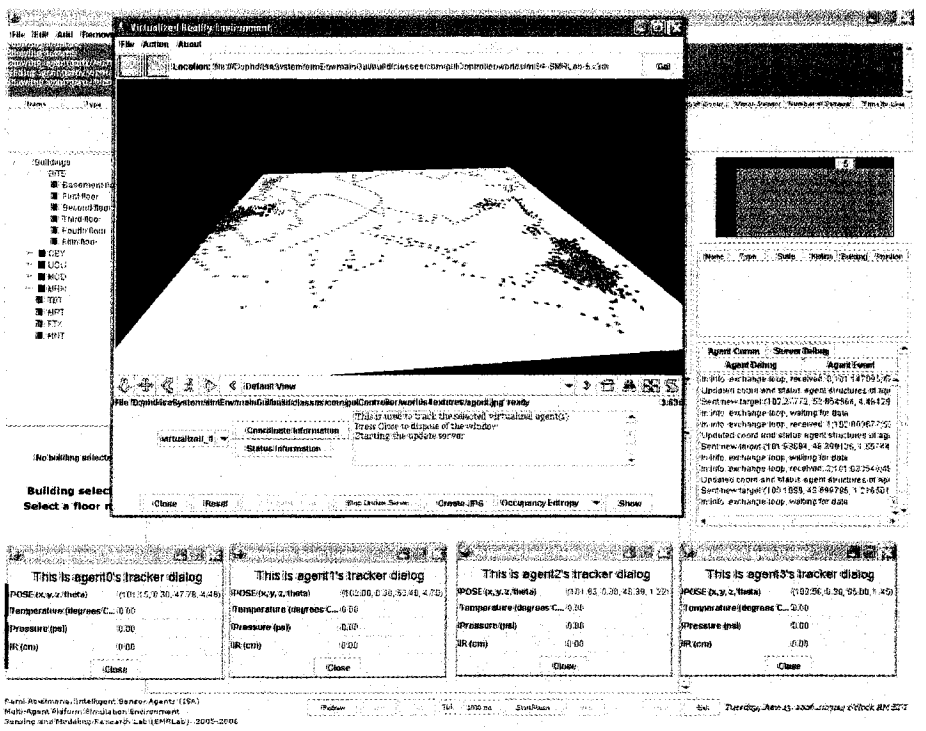


Figure 6.30: VRME experimental snapshot - an agent's path

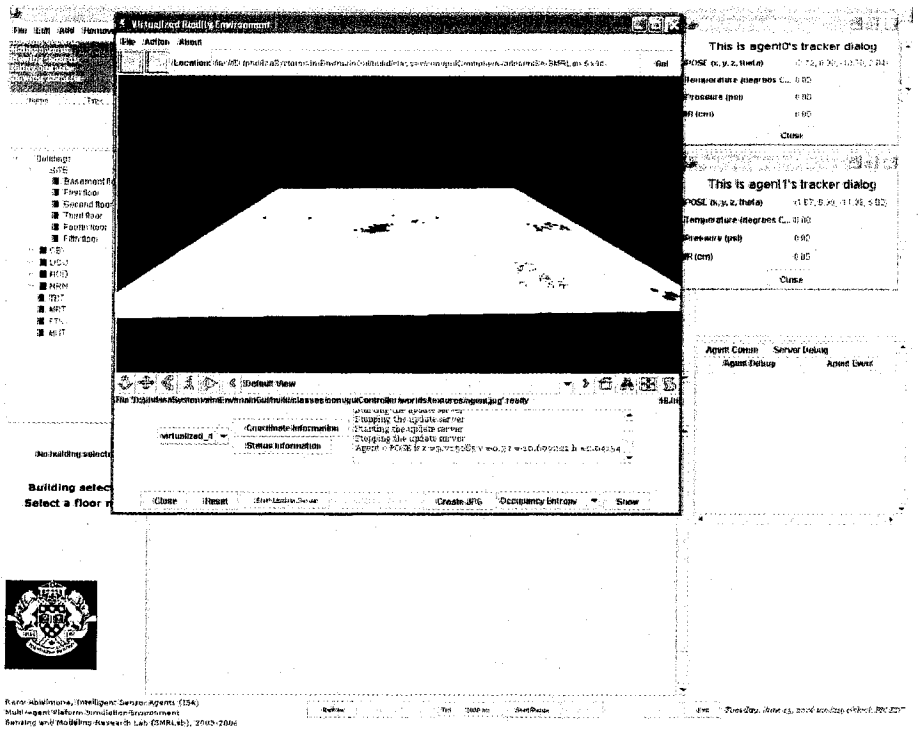


Figure 6.31: VRME experimental snapshot - a different setup

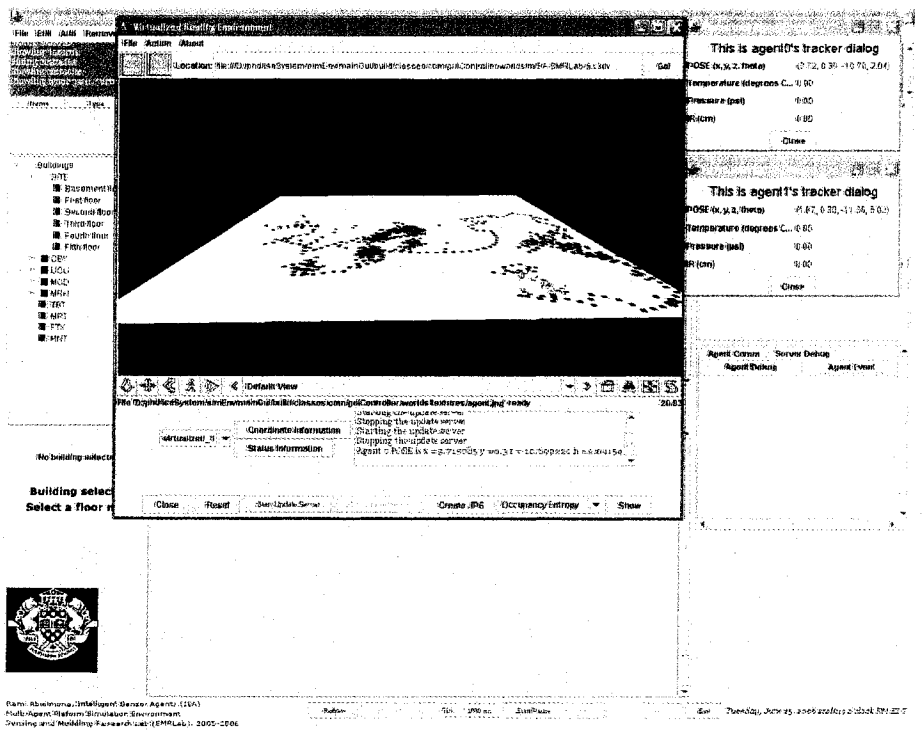


Figure 6.32: VRME experimental snapshot - another agent's path

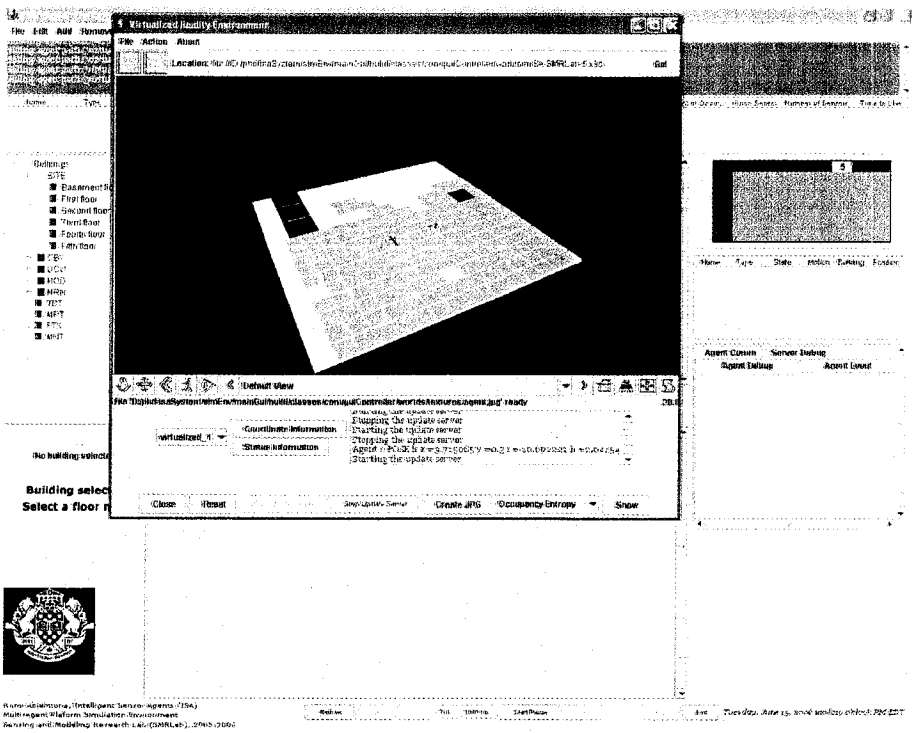


Figure 6.33: VRME experimental snapshot - with occupancy entropy

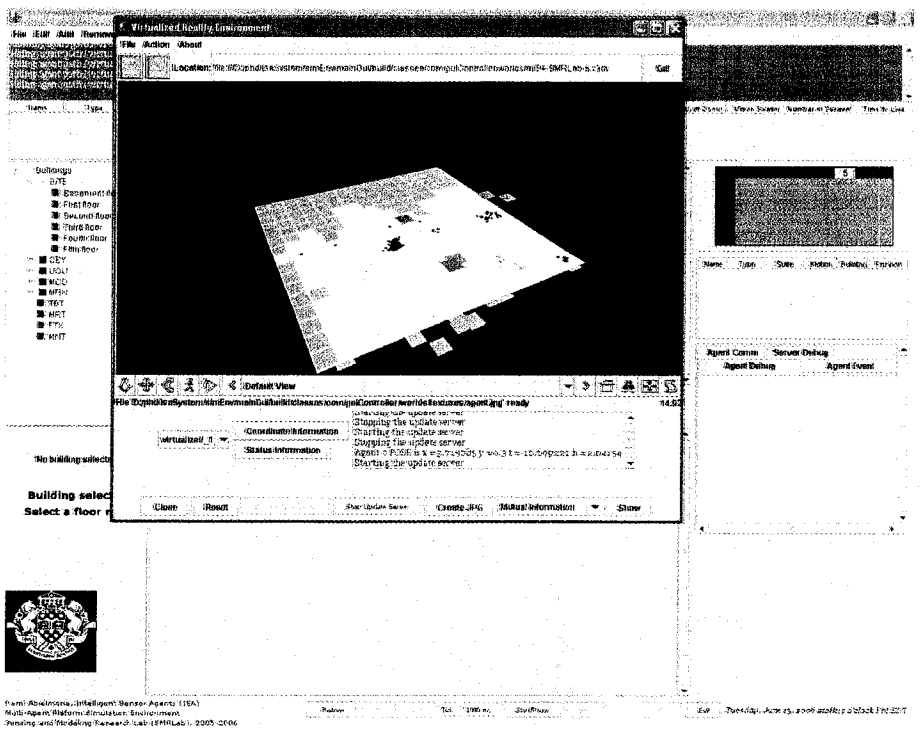


Figure 6.34: VRME experimental snapshot - with mutual information

6.9 Comparisons and Discussion

We ran the multi-agent system in numerous environmental setups, with a different number of physical and logical agents, as well as with a different number of obstacles and hazards. The experiments demonstrated that the TIMM method is viable for a real-time mapping and monitoring application, while the real-world VRME demonstrated that a remote world can be dynamically created and updated, so as to allow the user to navigate it.

The final grid storage size on the EAP, in the centralized case, and the mapping agent, in the decentralized case, is ideally 4772 bytes. This is the entire unpruned labelled quadtree data structure, that takes into account all resolutions and dimensions. Since there were 5 levels to account for 256 tessellations in the map, and each entropic value, and the two corresponding probability values, were stored as a IEEE 32-bit floating point number, along with the pointers that link the quadtree nodes together, we arrive at $340 * 4 = 1360$ bytes for pointers and $(340 + 1) * 4 = 1364$ bytes for data storage (the extra 4 bytes are for storing the pointer-less root node), for a total of 2724 bytes. We add to that the state probabilities that are only kept at the lowest depth of the quadtree, and we find our ideal total of 4772 bytes. This value is higher than the certainty or histogrammic grids that would be obtained using the OGF (1024 bytes) and HIMM (256 bytes) methods; however, we note the following three aspects.

The first is that since we have selected a forest of quadtree structure, where neighbouring quadrant cells that are in the same state (e.g. all hot or all occupied), then we may prune those cells and only keep the parent node, and assign the entropy value associated with the children cells. Using that technique, and after mapping is done, we were able to prune our data structure for the experimental run shown in Figure 6.29 to 45 nodes (only 24 of which are lowest depth nodes) and 44 pointers for a total of 548 bytes, a savings of approximately 88%. This is very comparable to the HIMM method, and much lower than the value for the OGF method. Note that this is the storage size for the pruned labelled quadtree at the EAP or mapping agent. The other agents only a local map consisting of a 8-by-8 cell grid, and only keeps track of the

state probabilities, within those cells, leaving the entropy and information fusion operations for the more resourceful entities. Thus, each agent held 128 floating point numbers (64 cells * 2 probabilities per cell, for the occupancy case), for a total of 512 bytes.

The second aspect of our data structure is that it offers a complete multi-resolution view of the environment. This is especially useful when we need to visualize our environment at multiple resolutions as well as when storing it on platforms with even more limited storage resources than the ones used in our experiments. Both the OGF and HIMM methods only offer single-resolution maps. The inclusion of the mapping agent allows us to embed the data structure within the multi-agent system; however, as mentioned, the mapping agent must be able to support an initial unexplored data structure totalling 4772 bytes. After exploration, and when subsequently queried by the individual agents, the map can be easily divided along different nodes and transmitted to the requesting agent.

The third aspect of our data structure is that it offers a multi-dimensional view of the environment. Multiple dimensions, such as occupancy and UV, are integrated in the TIMM technique. As we have seen, this provides us with the correlations that are required for off-line analysis such as mutual information extraction. To support multiple dimensions, the EAP/mapping agent's maximum storage resources grow linearly by a factor of $(n * m) + p$ floating point values, where n is the number of states for that particular dimension, m is the number of cells in the map, and p is the number of nodes in the quadtree. All other agents' storage resources linearly grow by a factor of $n * m$ floating point values, where n is the number of states for that particular dimension, and m is the number of cells in the agent's local map. Both the OGF and HIMM methods can support multiple dimensions; however, they do not relate dissimilar measured phenomena, i.e. the readings have to be pertaining to the same physical phenomena. This disadvantage is resolved by the TIMM method and its associated data structure.

Let us end this section by commenting on two additional performance metrics. In the occupancy grid scenario, the worst-case measured certainty value update time for the undertaken experiments using the TIMM method was $20.25\,\mu s$, running on a 7 DMIPS Nios-II soft

core processor, capable of 0.45 MFLOPs. This value encompassed the 2 floating-point updates of the state probabilities (additions or subtractions), the 2 logarithms for both values, the 2 floating-point multiplications of those results, and finally a floating point addition. Hence, 7 floating-point operations are performed for each certainty value update; however, to arrive to the correct certainty value, we need to traverse the labelled quadtree. In so doing, we require n integer comparisons, one for each level in the tree. In our 256 cell map, that amounts to 5 comparisons. This differed from our simulation results, as in the latter we did not account for the extra logarithm and product or the tree traversal. Having said that, the simulation results showed that OGF method's floating-point division was the main culprit for the extra delay in updating the certainty value. The HIMM method had the fastest CV update time, since it involved simple integer increments and decrements.

As it concerns the exploration strategy decision time step, the worst-case measured time step to find out which area to explore next was $30.04\,\mu s$. This was extracted while running an experiment for the occupancy grid scenario using the TIMM method. The value was found by studying the case where an agent has to decide which quadrant to head to next. First, the lowest level of the quadtree is explored by comparing, at most 3 floating-point comparisons, and if no decision can be made (i.e. the neighbouring entropies are equivalent), then we move one level up, and so on, until we reach the root node of the tree. In our 256-cell, 5-level case, that means $3 * 4 = 12$ comparisons. Note that there is no search required in inversely traversing the quadtree as the parent of a child's key is simply (L-1, (int) K/4), where L is the level we are at in the tree, and K is calculated by multiplying the value of the second coordinate of the parent node by four, and adding a value of 0, 1, 2 or 3 for the NW, NE, SW, or SE children, respectively.

Figure 6.35 shows the map extraction times, in seconds, for the different agent configurations, but using the same environmental setup. This figure presents the results for numerous agent configurations, including ones with only physical agents, ones with only logical agents (using the ISA emulators discussed above), and ones combining both. We ran the experiment for

approximately 16 minutes, where we found that the environmental entropy level had saturated, due to the presence of obstacles and hazards that could not be driven across, hence keeping those cellular entropies close to their initial values. We ran the experiments with four physical agents, two emulator agents, and then a hybrid of all six. It was found that the hybrid agent configuration performed the best given the same environment setup, minimizing the environmental entropy to below 0.275, with the physical agents only configuration following closely behind at 0.295. Now, let us note that the emulators are obviously not present in the environment, per se, instead they communicate with the agents and the EAP/mapping agent whenever they need to do so. This tends to lead to a lesser amount of agent *collisions* where an agent's IR sensor detects another agent, instead of an obstacle. That explains the slightly faster and better map extraction time results.

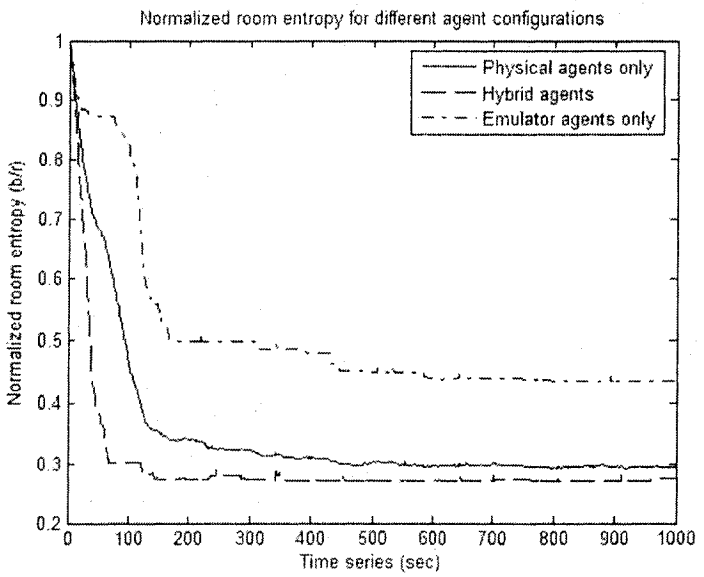


Figure 6.35: Experimental map extraction time results

6.10 Summary

This chapter has shown that the experimental results verified the simulated ones for the TIMM method, as we were able to consistenly explore environments in real-time, and feed the

virtualized reality model through the access point. Through experimentation and optimization, we found that although the TIMM method's grid storage size is slightly higher than the OGF method's, the former does offer a multi-resolution and multi-dimension view of the environment. We also measured the worst-case certainty value update time to be $20.25\,\mu s$, and the worst-case exploration strategy decision time step to be $30.04\,\mu s$, while concluding the chapter with the map extraction times for three configurations, one involving physical agents-only, another involving emulator agents-only and a last one involving hybrid agents. It was found that the hybrid agent configuration performed the best, mainly due to the lesser amount of agent collisions that the physical agents would provide.

Chapter 7

Virtualized Reality Interface

"The person who says it cannot be done
should not interrupt the person doing it"
(Chinese proverb)

The user interface of a multi-agent system is an important element of its overall structure. It is the method by which the user is presented with a stable and consistent view of the environment, and is allowed to interact with its constituents. This chapter presents a unique user interface, termed the virtualized reality model of the environment, that provides the user with a real-time view of the real-world environment that is being monitored.

Such an environment synthesizes a world model from the fusion of the distributed sensor measurement data. It can be used by remote operators, such as air traffic controllers and space mission commanders. The proposed interface conforms to the X3D standard and supports the dynamic insertion/deletion of virtualized agents and objects. It also supports the dynamic loading of remote environment models for real-time decision making.

7.1 Virtualized Reality Model of the Environment

Human virtual environment tele-immersion has been a research endeavour undertaken by numerous institutions. Some eccentric researchers have gone as far as presenting the idea of "robotting into" a machine [116], where human operators can assume control of a remote robotic machine by immersing themselves into an environment of virtual sensing and actuating.

This, the author extrapolates, will allow for a robotic industrial revolution where human domestic workers are hired to control and oversee the operation of robots, physically present halfway across the world, and to perform repetitive tasks that home owners can subscribe to (i.e. clean the house, do the laundry, wash the dishes, etc). Other exciting ideas involve the concept of *mind children* [117] and [118], where the human mind is uploaded to a robotic machine, and assumes full control over the physical structure: complete immersion at its best! However, before we arrive to these extreme scenarios, it is imperative to deal with the many issues that are present in today's partially immersive environments: number and type of sensors, sensor bandwidth and deployment limitations, sensor data reliability, low-cost quality of service data communications network provisioning, as well as partial and heterogeneous sensor view integration. In this section, we will concentrate on the latter issue.

To partially immerse a human monitor into the environment, it is necessary to begin the creation of a virtual environment that can support multiple agents gathering various information about objects present and events occurring in the physical world, and that can synthesize a world model from the fusion of that distributed data. The outcome is what has been referred to earlier in this thesis as a VRME. Our first attempt at this task did not involve complicated head-mounted displays or haptic gloves, but instead we chose to implement a VRME that can be viewed on a standard web browser, so that we do not lose the functionality of having a remote human monitor log in to check up on the state of the environment. What follows is a brief description of the design and implementation of this VRME.

Seeing as the remote access could be at any location and at any machine on the wireline network, the Java programming language became the method of choice for network and graphical functionalities, while the Virtual Reality Modeling Language (VRML) was chosen for the implementation of the actual environment model in 3D. Research was performed concerning other languages such as JavaScript 3D, Java 3D, and Maya, but were all found lacking in desired functionality or were too computationally intensive for the purpose at hand. Finally, a tool set had to be selected for the actual VRML development, and the combination of graphical

modeling using VizX3D [119] and textual coding using VRMLPad [120] proved to be the most efficient.

7.1.1 Preliminary Design

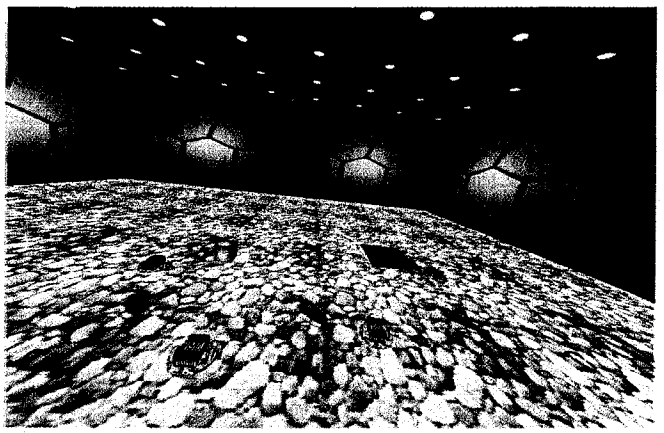


Figure 7.1: VRME with agents, obstacle and hazard

A virtualized reality model was built and is shown in Figure 7.1. The agents can be seen as two-wheel mobile robots modeled with a computational engine and a range sensor in the front. Obstacles and hazards are currently modeled as outlined in the figure, by shapes of different sizes and colours. Along with the VRML model, a Java applet was implemented within the same browser page, to provide the human monitor with real-time information about the queried agent. Initially only the coordinates, temperature and pressure information were brought out to the monitor through this applet (refer to Figure 7.2 for a screenshot of the browser window). On that figure, the reader can also notice another window: the Cortona Console. The Cortona VRML Client [121] was utilized as a plug-in for the browser, to enable the latter to render VRML objects within a web page. The Cortona Console happens to be a debugging window that was used for catching the signals coming into and going out of the VRML world. The combination of all three components shown in Figure 7.2 presents the first attempt at a VRME.

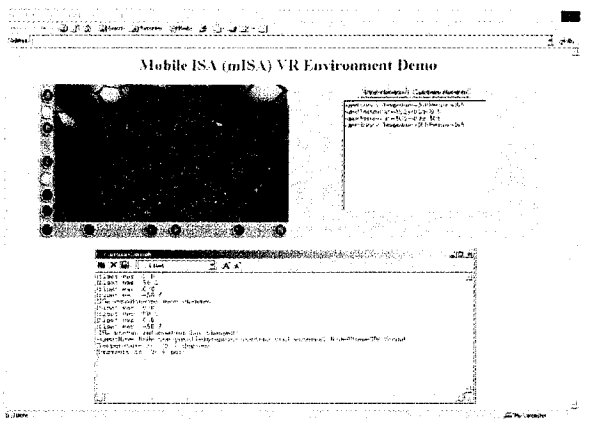


Figure 7.2: Java applet alongside the VRME

The VRME's behavior is shown in Figure 7.3. A time sensor is implemented and triggers the world every second (action 1). A file is reloaded at every "world clock tick" and is checked for ISA changes. If a change is detected, then the fileModified flag is set (action 2), and that causes another timer, animationTimer, to generate one second ticks to yet another timer, agentTimer (action 3). The latter periodically updates the position of the ISA in the VRME through the function, set_fraction, of the agentMove node (action 4). This causes the ISA to move through the function, set_translation of the agent node (action 5). Finally, each ISA in the model is equipped with a touch sensor, that, when activated (action 6), sends a signal to the getStatus function of the agentMove node to update the human monitor with the sensory information on-board the "virtually touched" ISA.

To complete this section, it is worth noting that at the outset of the VRME design, no physical agents existed, hence a Java client-server application was written to simulate various ISAs updating the VRME with their coordinate and status information. In this case, the clients represented the ISAs and the server represented the human monitor. However, as the physical agents came to life, the client-server application was dropped for the real-world communication between ISAs and human monitors.

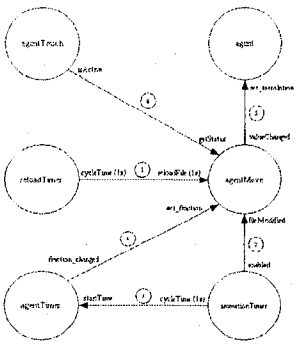


Figure 7.3: State diagram of VRML world

7.1.2 Final Design

Over the course of the past few months, the VRML world has been systematically converted to the more recent 3D scene and object representation: **X3D**. The latter is a standard published by the Web3D consortium, and has a rich set of componentized features that can be described using XML. It contains four major profiles, each with extensions that meet specific application needs. The profiles are [122]:

Interchange the basic profile for communicating between applications. Supports geometry, texturing, basic lighting and anmiation

Interactive the profile for interacting with a 3D environment. Supports sensor nodes, enhanced timing and additional lighting

Immersive the profile for full 3D graphics and interaction. Supports audio, collision, fog and scripting

Full the most complete profile. Supports NURBS, H-Anim and GeoSpatials

Each profile encompasses the features offered by the one below it. In our VRME application, we use the immersive profile since we require the use of sensor nodes (touch and time), as well as scripting nodes. The application has been completely written in Java and X3D, and combines all of the libraries defined below (see Figure 7.4):

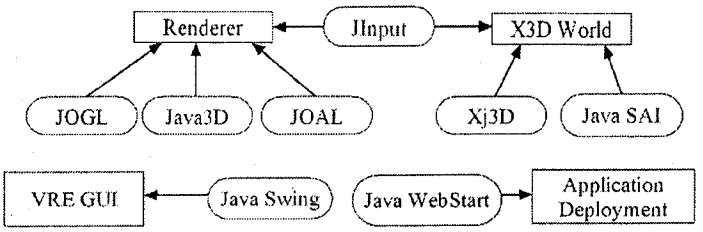


Figure 7.4: Utilized Java libraries

JOGL This is a reference implementation of the Java Bindings for OpenGL, and is intended to provide hardware-support for 3D graphics writeen in Java [123]

Java3D This is a Java API that interfaces to 3D graphics and sound rendering systems [124]

JOAL This is a reference implementation of the Java OpenAL API and is intended to provide hardware-support for 3D spatialized audio for applications written in Java [125]

JInput This is an implementation of an API for game controller discovery and polled input [126]

Xj3D This is a toolkit for the creation of VRML97 and X3D content written in Java. The toolkit allows the designer to integrate a VRML/X3D browser into their own application [127]

Java SAI This is the Scene Access Interface, that is the successor (and merger) of the Scripting Authoring Interface and External Authoring Interface developed for the VRML standard. It provides a compatible way to run and dynamically modify X3D scenes and makes part of the original X3D ISO standard [122]

Java Swing This is a set of lightweight GUI components, written in pure Java, intended for a platform-independent and sophisticated screen displays [128]

Java Web Start This is a technology that enables standalone Java software applications to be deployed with a single click over the network, while ensuring the most current version of the application is deployed [129]

It is worth mentioning that we have used a specific binding of the SAI for the Java language, that has been approved as another ISO standard [130]. Rendering was done by first using the Java3D API, but we have achieved a stable X3D scene animation and navigation using the JOGL/JOAL APIs at approximately 40 frames/second. The Xj3D toolkit was used for the development of the X3D world, while the Java SAI was used to inject programming controls and behaviors into the static X3D scenes, allowing for a more dynamic world. The JInput API was used to allow for an abstraction of both the keyboard and mouse input devices when navigating the X3D world, and detecting events, such as key presses or mouse down events. The Java Swing components were used in the development of the container of the X3D world, as well as the user interface events, and finally, the Java Web Start technology was used for remote deployment of both the VRME application and its parent MAPSE application (to be discussed in subsequent chapter). To deploy the application remotely, a file conformant to the Java Network Launching Protocol (JNLP) was created, that defined the remote locations of all the libraries that are required to be downloaded to the client the first time the application is run. This also includes downloading and installing the appropriate Java Runtime Environment (JRE) on the client's machine, if the latter does not contain one already. Finally, the VRME application is delivered by the JNLP file as a Java ARchive (JAR) file, along with a self-signed certificate that must be accepted by the user before the application is intalled on their machine.

7.1.2.1 VRME Details

To support the apparation of a new mobile agent into the environment, the EAP must first register the former into its local mapping tables, and transmit the new agent identification to the VRME application. The latter proceeds, through the Java binding of the SAI, to instantiate a new agent node graph, as indicated in Figure 7.5. The agent node transform is given a name (e.g. *agent0*), with three characteristics, mainly translation, rotation and scaling factors, as well as children node. The latter contains three such nodes: *Shape*, *TouchSensor* and *TimeSensor*. The *Shape* node defines the geometry of the agent node, as well as its appearance. In our case,

agents are modeled as boxes with a texture of an agent's picture envelopping the box. The *TouchSensor* node defines an event that occurs when a mouse down or a mouse up event occurs on the agent transform node. The *ViewPoint* node defines an agent-centric viewpoint for the navigation of the environment from the agent's perspective. Finally, four *TimeSensor* nodes, all with 1 second intervals, define the controlled behavior of the agent within the X3D world. The *translation and rotation timers* control the rendering fluidity of the agent's movements within the 3D world. The *animation timer* controls the translation and rotation timers by providing them with their start times when the agent is required to animate. Finally, the *reload timer* is used to periodically check with the external world, whether the agent is required to animate.

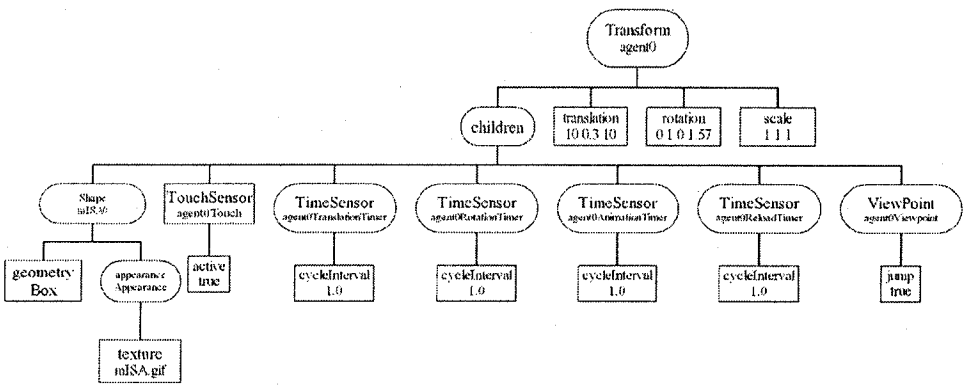


Figure 7.5: Virtualized agent node graph

Along with dynamically creating an agent tranform node, we also must create the script node that allows for the agent's behavior to be controlled from the Java language binding provided by the SAI. The *agentMoveScript* is dynamically added for each agent, and its inputs and outputs dynamically routed to the specific components, as shown in Figure 7.6. The sequence of events occurs as follows:

(1) The reload timer generates an event that is routed to the *reloadValues* input of the *agentMoveScript* node;

(2) The *agentMoveScript* node generates a signal that is routed to the *enabled* input of the

animationTimer node;

(3) The *animationTimer* node generates a signal that is routed to both *startTime* inputs of the *translationTimer* and *rotationTimer* nodes;

(4) The *translationTimer* and *rotationTimer* nodes generate *fraction_changed* signals that are routed to the *setPosFraction* and *setHeadingFractio* inputs of the *agentMoveScript* node;

(5) The *agentMoveScript* node proceeds to translate and rotate the agent virtualization by generating appropriate *set_translation* and *set_rotation* signals to the *agent0* node;

(6) Finally, the *agent0Touch* node generates an *isActive* signal every time the agent is clicked upon by a mouse, that retrieves the agent's sensory status for display.

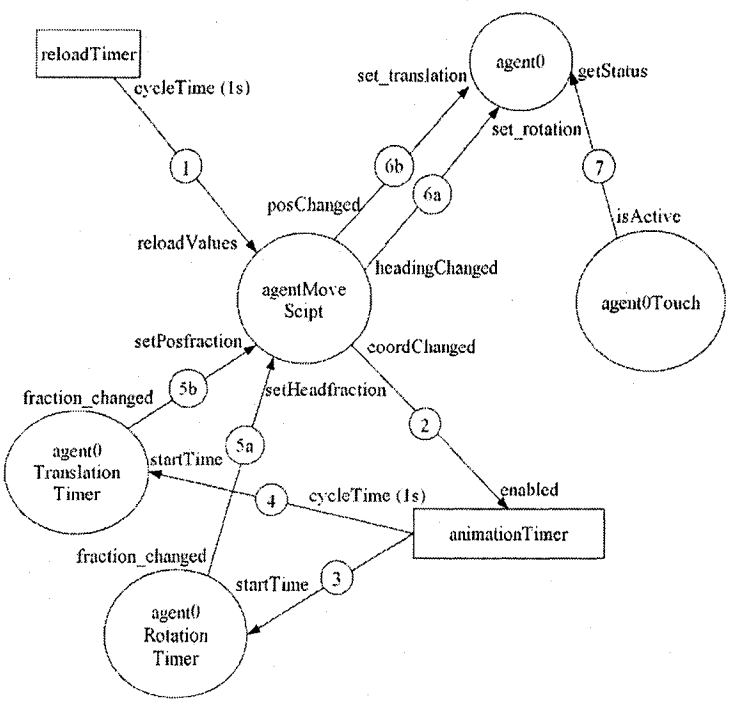


Figure 7.6: Virtualized agent state diagram

Note that, in addition to the dynamic insertion of virtualized agent nodes, the VRME ap-

plication also supports the dynamic insertion of virtualized objects, mainly obstacles (white thin vertical stripes), to indicate walls and barriers, hazards (red thick horizontal plates), to indicate environmental parameters that are beyond set thresholds, paths (coloured thin horizontal plates), to indicate agent path traversals, and finally, cells (grey thin horizontal plates), to indicate the entropy and mutual information values of a particular coordinate.

Let us now try to briefly describe the complex design and operation of the VRME application. Shown in Figure 7.7 is the collaboration diagram of the most important classes involved in the execution and communication of the application. Again, let us go through the sequence of operations for the application:

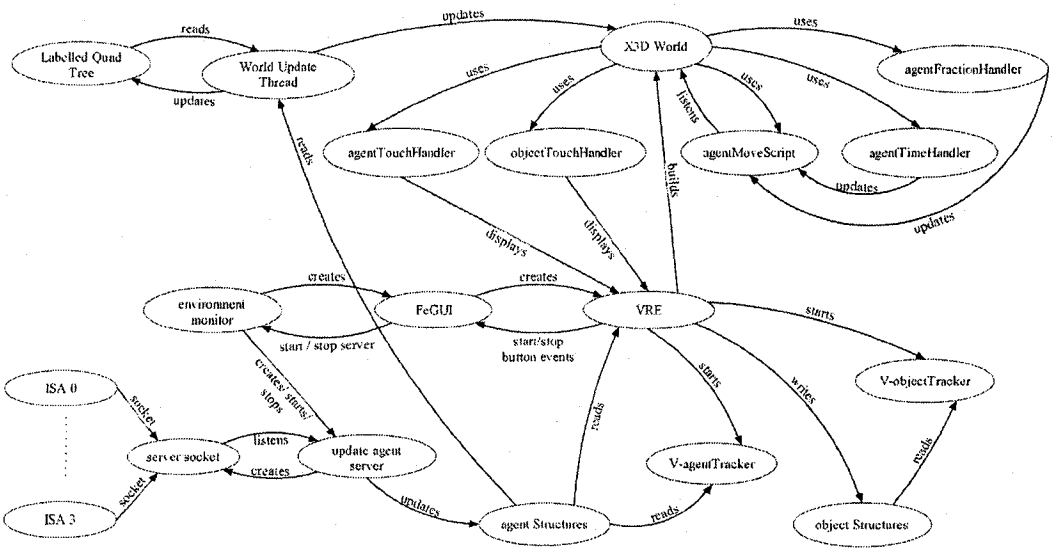


Figure 7.7: VRME collaboration diagram

(1) The environment monitor class starts the front end GUI (*FeGUI*) class as well as the *update agent server*. The former is the GUI of the MAPSE application, while the latter is a thread that listens in for socket connection requests from the EAP, to download the recent ISA information from the latter's local mapping tables;

(2) The *agentStructures* and *objectStructures* objects are instantiated at startup and hold the virtualized agents and objects, respectively;

(3) A *V-agentTracker* object is instantiated for each virtualized agent to track its coordinates and status information. The same holds for objects with their *V-objectTracker* instances;

(4) The VRME application is launched by pressing a button within the FeGUI. The VRME proceeds to create the *X3D World* instance, and populate it with virtualized agents and objects;

(5) As aforementioned, each virtualized agent uses a script node, while handling touch and time events, through the *agentTouchHandler* and the *agentTimeHandler* instances, respectively. The *agentFractionHandler* is used by the X3D world objects to calculate the fraction that has been translated or rotated by the agent;

(6) Finally, a *world update server* is launched as a thread, and proceeds to update the environment data structure, as well as the X3D world, whenever changes occur in the *agentStructures* object instance.

Let us conclude this section by viewing a few snapshots of the VRME in action. Table 7.1 briefly describes each snapshot.

Snapshot #	Description
1	Initial default view of the X3D scene with the 4 physical agents
2	Preliminary mutual information grid
3	The four agents begin their map exploration using the TIMM method
4	Intermediate occuppancy grid with agent paths shown
5	Intermediate mutual information grid with agent paths shown
6	Viewpoint from agent3's perspective with agent paths and entropy cells shown
7	Obstacle objects are detected and dynamically added to the virtualized model
8	Advanced mutual information grid, showing that certain cells have negative values

Table 7.1: VRME snapshot descriptions

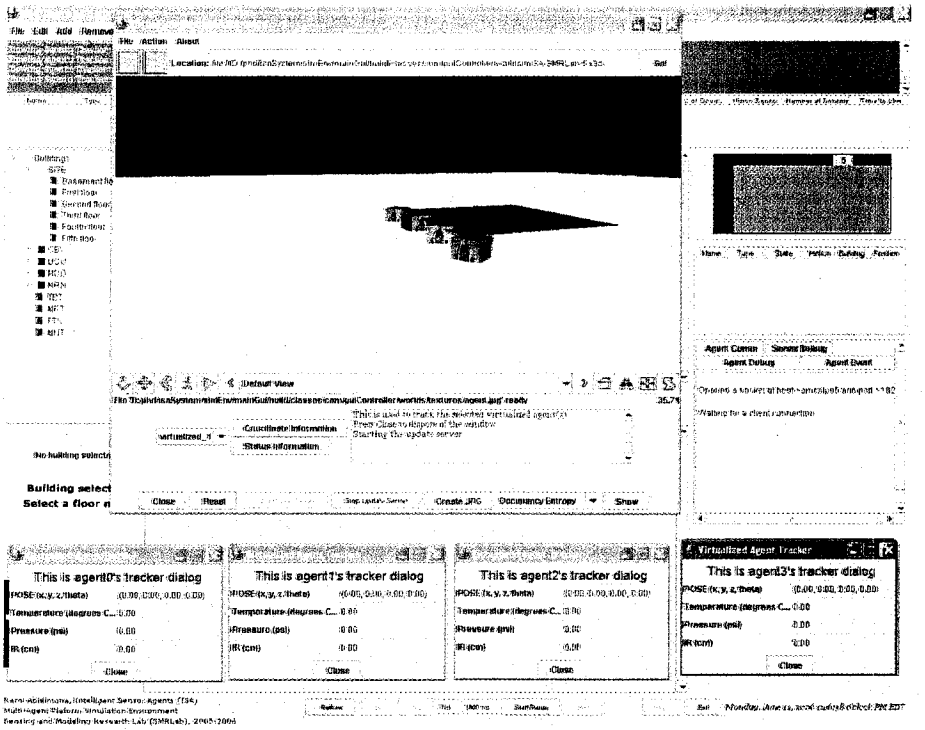


Figure 7.8: VRME snapshot 1

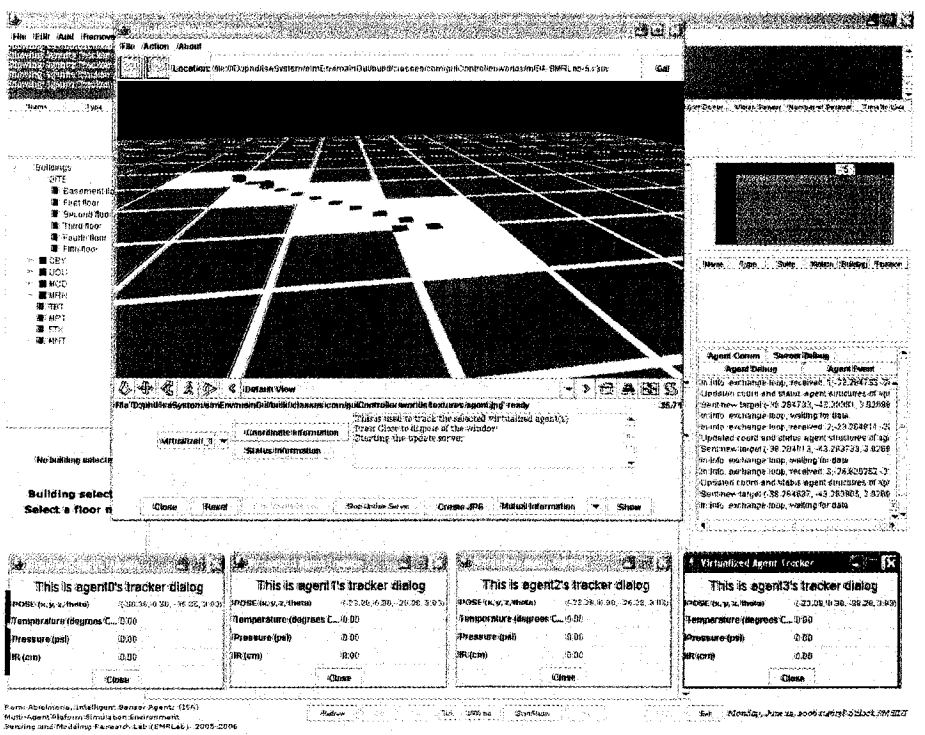


Figure 7.9: VRME snapshot 2

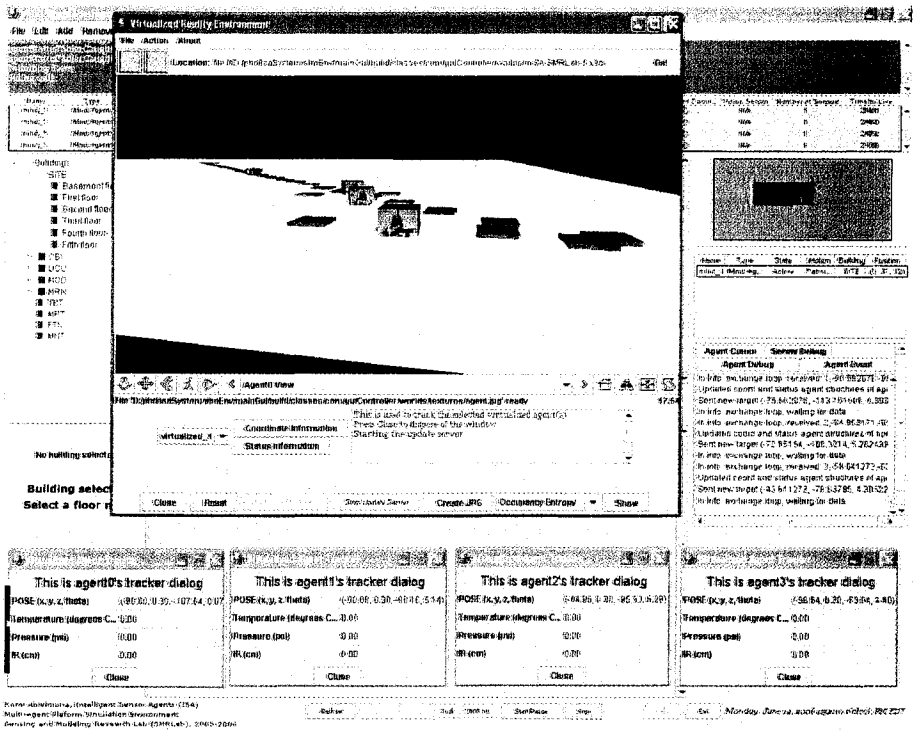


Figure 7.10: VRME snapshot 3

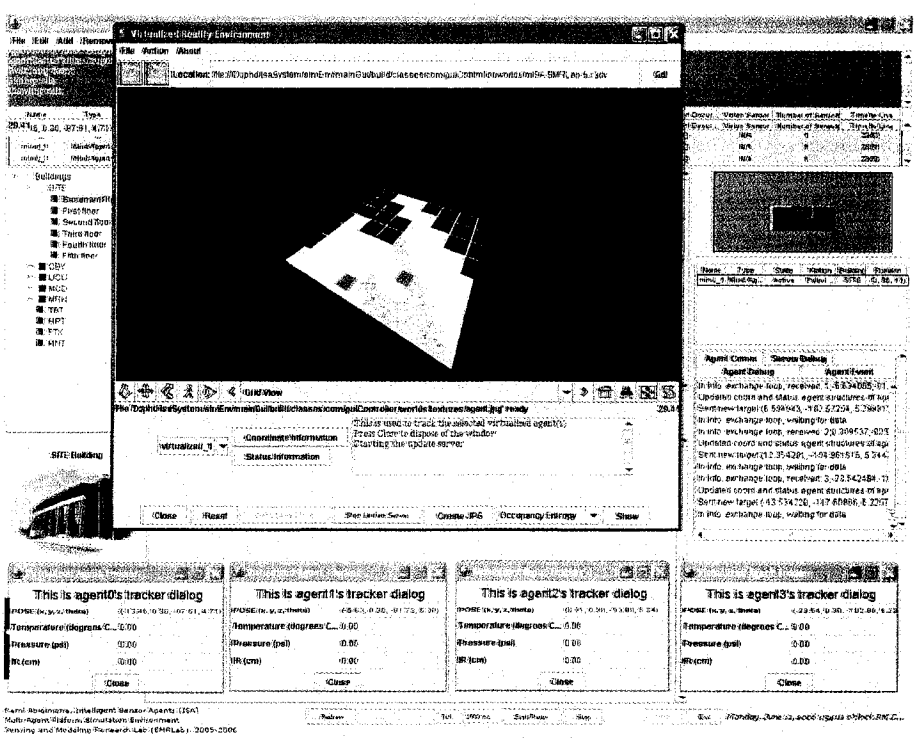


Figure 7.11: VRME snapshot 4

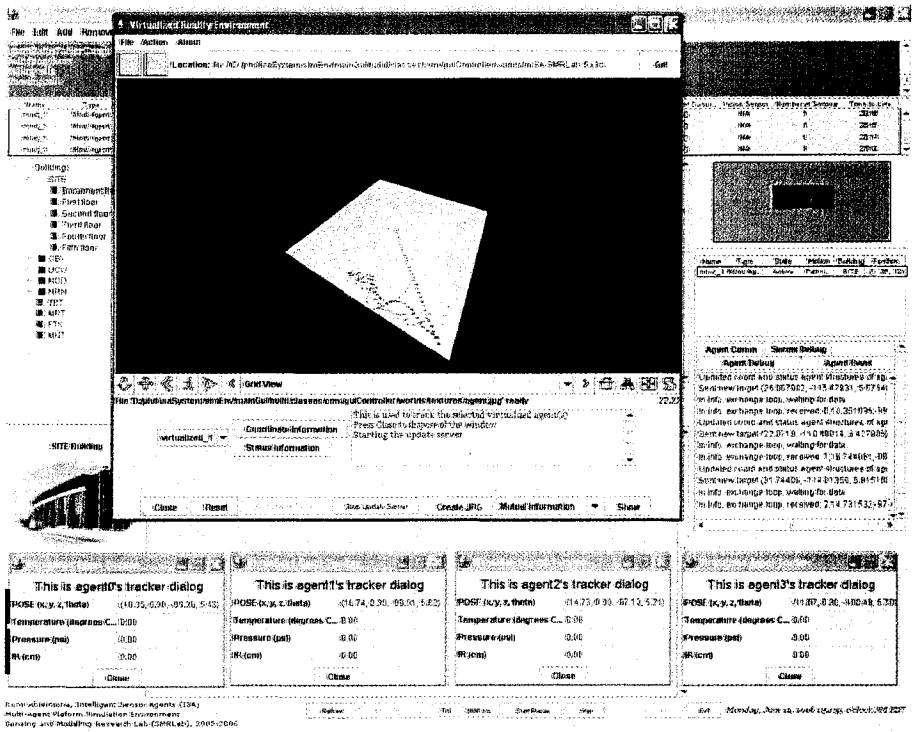


Figure 7.12: VRME snapshot 5

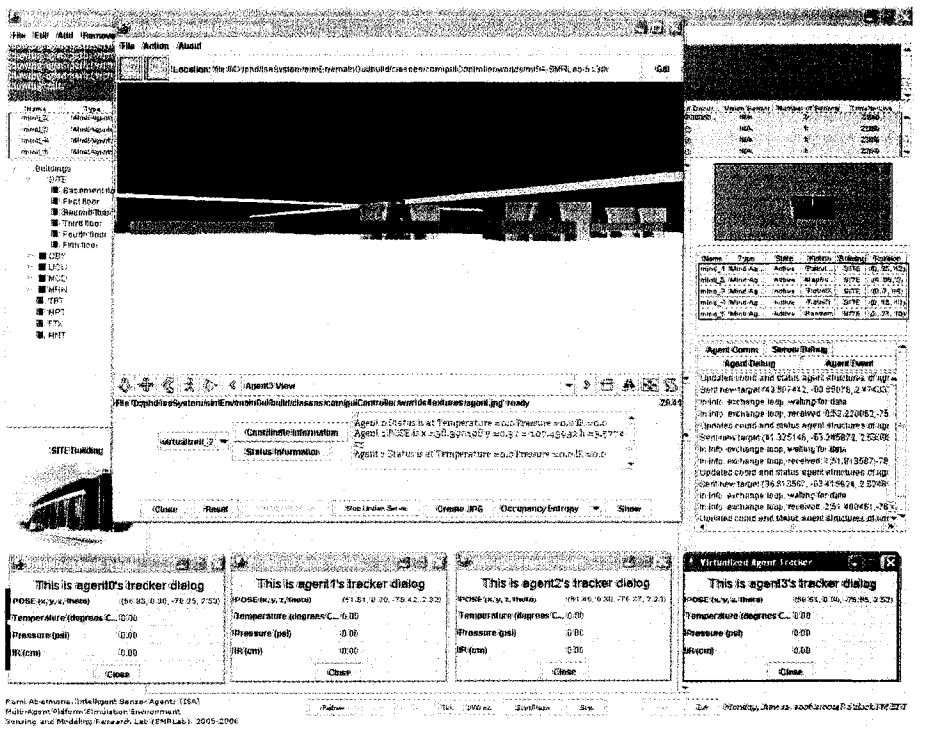


Figure 7.13: VRME snapshot 6

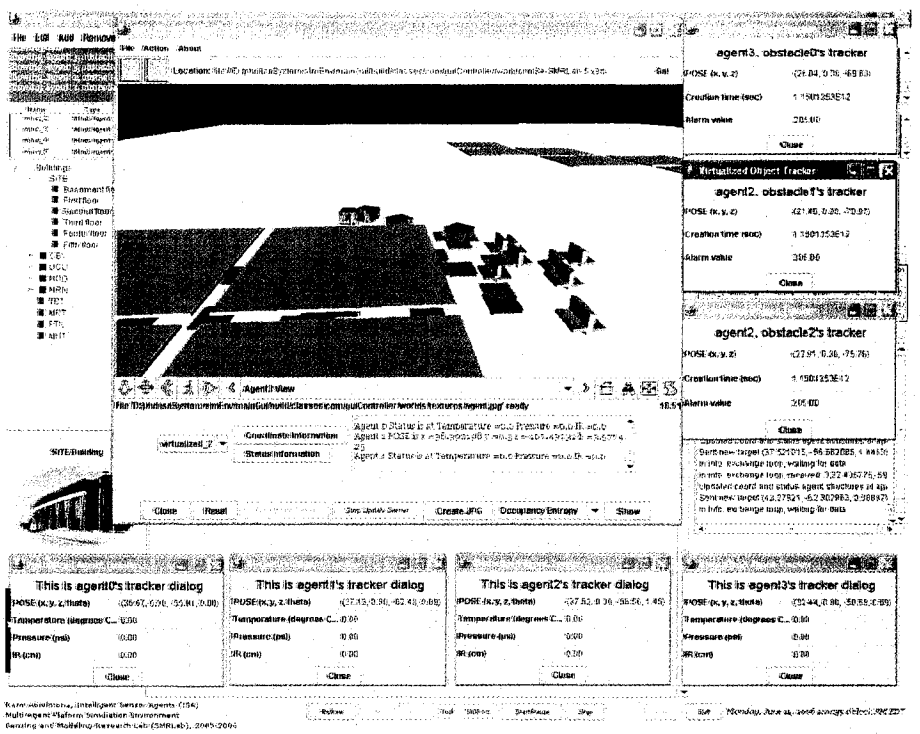


Figure 7.14: VRME snapshot 7

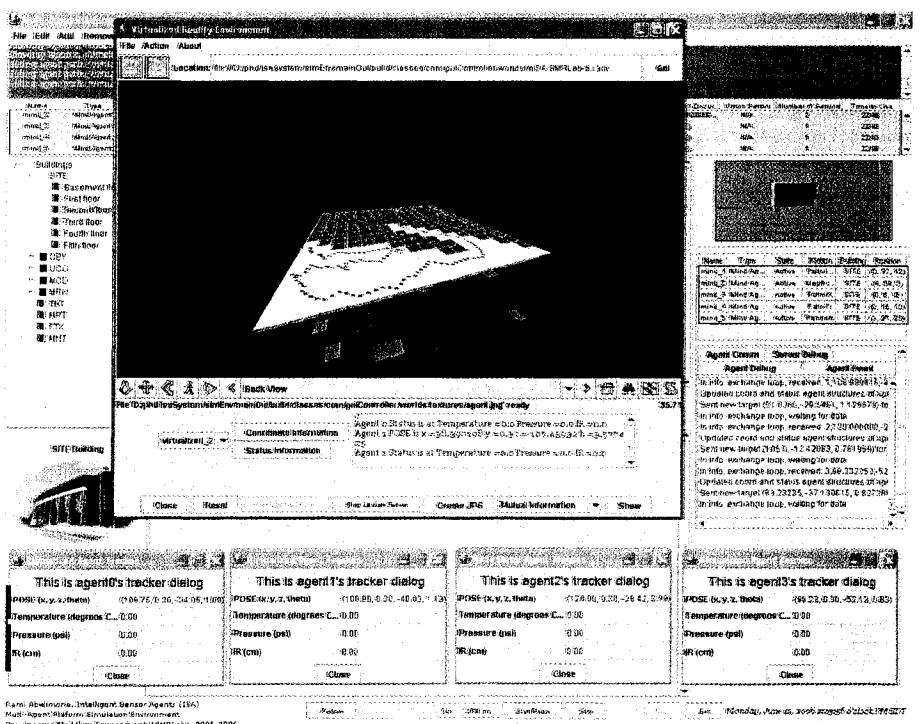


Figure 7.15: VRME snapshot 8

7.2 Summary

The use of a virtualized reality model is an important concept, since it allows the remote operator to immerse themselves into the environment they are monitoring, leading to a better decision-making process. This is demonstrated, in this chapter, by the various environment views (e.g. agent vs. environment) and measures (e.g. obstacle occuppancy and mutual information) that are conveyed to the remote operator, and their real-time updates. Furthermore, the dynamic insertion/deletion of virtualized agents, supporting the ad-hoc nature of the underlying network, is presented through the seamless integration of a virtualized agent node graph, and its associated functionality represented by a state diagram, that scales to the excess of 5000 virtualized agents. Similar concepts are applied to virtualized objects, the latter scaling to the excess of 10000 virtualized objects.

Chapter 8

Conclusions and Future Work

"I have offended God and mankind because my work didn't reach the quality it should have"

(Leonardo da Vinci)

Here, conclusions resulting from this work are summarized and the achievements of the work are presented, before further work that would build on this is discussed.

8.1 Conclusions

In this thesis, a novel environment monitoring method, useful for realization on a multi-agent system composed of agents that each possess a limited amount of computational power and communications bandwidth, was designed, developped and experimentally verified. The introduced TIMM technique allows for an efficient representation of the environment, as well as the generation of a simple and compact map, that can be easily stored and transmitted among the agents themselves, or between an agent and a processing station.

A multi-agent framework, called MAPS, has also been detailed in this thesis. The formal specifications serve as a guideline for the realization of a multi-agent system for environment monitoring, that is robust, fault-tolerant, platform-independent and scaleable. An associated simulator, MAPSE, built on the rules set forth by the framework, has also been developped and discussed. The latter allows us to develop, integrate and verify various hypotheses and scenarios involving multi-agent systems in a simulated world.

A novel agent architecture has been presented and outlined. The retroactive agent architecture describes a revolutionary concept in the design of intelligent machines. Originally attributed to Vernon B. Mountcastle, who synthesized the idea that every part of the human's cortex is made up of the same structure, the *common cortical theory* is now being viewed, by some neuroscientists, as the elusive link to building truly intelligent machines. The *memory-prediction model* of the brain has already been introduced from a computer scientific perspective by [97], but has recently made tremendous advancements in the synthesis of intelligent machines [20].

Finally, a virtualized reality model of the environment, VRME, was developped, created and validated by feeding real-world and real-time sensory data begotten from physical agents, and dynamically creating a world filled with virtualized agents, paths, obstacles, hazards and cells. The synchronization between the VRME and the various access points (e.g. EAP) also allows for agents to drop in and out of range, or in and out of the environment completely, without affecting the virtualized world. This seamless immersion aids human monitors to remotely get safer and better acquainted with unknown, and possibly hostile, environments, by deploying a foraging society of robotic agents that synthesizes the VRME in real-time.

8.2 Achievements

The following technical achievements have been accomplished described in this thesis:

- *Mobile robotic platforms*: An autonomous wireless robotic platform has been developed. It is robust, flexible and easy to reproduce;
- *Agent-based resource management*: This framework has been developed and allows for a seamless integration of ISAs into DISS;
- *Subsumption architecture*: Retroactive hierarchical control schemes have been realized and demonstrated to successfully control and coordinate four robots joinly exploring their environment;

- *Reinforcement learning*: Although not discussed in this thesis, a reinforcement learning technique called learning classifiers has been realized and demonstrated for the control of dynamically reconfigurable mobile robotic structures [19];

- *Local and world mapping*: A local map is maintained on the ISAs through a combination of dead-reckoning using odometric readings and heading self-correction. A global map is synthesized and displayed graphically;

- *Social behavior*: Cooperative, competitive, and negotiated agent relationships have been observed. This leads to insight allowing improvement in performance for contract negotiation between cooperative agents;

- *Wireless communications protocol*: A scaleable and robust protocol has been realized and tested;

- *Multi-sensor fusion*: A multi-sensor fusion technique, based on entropy minimization and information maximization and correlation, has been developped and simulated;

- *Novel monitoring method* A real-time technique has been developed for the monitoring of an environment using a set of mobile agents that possess a limited amount of computational power and communications bandwidth;

- *MAPS formal specifications*: A platform-independent formalism has been developped from the ground up, that allows for different implementations of multi-agent systems for environment monitoring;

- *MAPSE virtual tool*: A software simulator that allows us to study various strategies within a virtual environment, to find out which strategy maximizes cost-effectiveness, while minimizing delay and energy has been realized;and

- *VRME*: A real-time virtualized reality model of the environment that accepts physical sensor data and updates a global model of the environment, has been realized.

8.3 Aspirations

In a world of pervasive and ubiquitous computing, we find ourselves embedding computing into the very fabric of our lives. Today, we can wirelessly stream music from a central server residing in our household. Tomorrow, we will be able to task and query sensor networks that will serve as home networks, taking care of such things such as home security, appliance interactions and control automation. The late Mark Weiser from Xerox PARC once said: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguashible from it". Our world has been transformed by the onset of the computer, migrating mundane and repetitive tasks into the realm of the machine, as well as passing down a certain level of trust in our everday lives that the computer (and its derivatives) will partake in our decisions as much as our own conscience and intuition. Multi-agent systems, and in particular, sensor networks, will play an important part in taking the next step: making the computer *disappear.*

We have entered a new era of computing, where intelligent sensor systems will be prevalent in many of the everyday tasks that are either difficult for a human to achieve (e.g. fixing a part inside a car), or mundane for a human to partake in (e.g. monitoring light intensity and closing the blinds accordingly). Future research directions can already be seen within the community:

- Design of tiny, low-power, low-cost modules
- Network layer discovery and self-organization algorithms
- Collaborative signal processing and information synthesis
- Tasking and querying interfaces with the sensor network
- Security for protection against intrusion and spoofing
- Reconfiguration techniques into suitable sensor network configuration

Observing a few market trends, one can notice that sensors are getting smaller in size and variable in nature, computing power is getting bigger and embedded, and communications bandwidth is getting higher while transceivers are getting smaller. The development of multi-agent systems and sensor networks will spawn what the author terms the **adhesive computing era**. Adhesive computing is a computing infrastructure that bonds with its human counterpart, providing the latter with a consistent view of the digital world. It is quite imperative for the human to interact with a controlled digital environment, or face the chaos that exists today. Adhesive computing glues the many interfaces we have with the digital realm together. A multi-agent system, such as the one described in this thesis, will become the machine interface to that digital realm, through which all tasking and querying will be conducted.

If we look at today's digital advancements, we will notice many types of innovative computing fields, such as *pervasive computing*, where the computer becomes omni-present, *ubiquitous computing*, where the computer becomes ever-present, *context-aware computing*, where the computer becomes self-adaptive, *fault-tolerant computing*, where the computer becomes self-correcting, *reconfigurable computing*, where the computer becomes self-configurable, and *evolvable computing*, where the computer becomes self-refining. Other existing types of computing include network, personal and symbolic. Moreover, *adhesive computing* allows the computer to become self-connecting.

Multi-agent systems can be applied to a few areas: military (e.g. object tracking), health (e.g. vital sign monitoring) environment (e.g. natural habitat analysis), home (e.g. motion detection), manufacturing (e.g. assembly line fault-detection) entertainment (e.g. virtual gaming) and the digital lifestyle (e.g. parking spot tracking). Multi-agent systems provide flexibility, fault-tolerance, high sensing fidelity, low-cost and rapid deployment. It is the author's firm belief that the work presented in this thesis will contribute to the foundation of the adhesive computing era, as well as the advancement of intelligent machine design. To that end, commercialization of certain topics discussed in this thesis has been entertained by a local company, with the hope that we can scale our system to millions of sensors collaborating together to

control the information overload that humans cannot possibly deal with.

8.4 Future Work

Future work includes (i) the realization of service discovery and registration in the EAP to allow for better human monitor and user interaction with the VRME, (ii) the introduction of a scavenging behavior for ISAs, as discussed in [131], (iii) the inclusion of an actuatory feedback from the VRME back to the wireless agents, (iv) the offline comparison of the retroactive agent architecture as opposed to the reactive, proactive and hybrid agent architectures, (v) the inclusion of the evolutionary control, in the MAPS specifications, for directing the spawning of elite agent chromosomes and (vi) the compensation for both sensor motion (while sensing) and for obstacle motion (target tracking).

References

[1] G. Giralt, "The Remote-Operated Autonomous-Robot concept: from deep sea to outer space applications," **Intl. Symp. Missions Technologies and Design of Planetary Mobile Vehicles**, 1992.

[2] Sensor nets top R&D list for Homeland Security agency, "EE Times," **http://www.eetonline.com/article/showArticle.jhtml?articleId=18310193**, Last seen on December 31 2003.

[3] S. Kumar and D. Shepherd, "SensIT: Sensor information technology for the warfighter," **Proceedings of the 4th Int. Conf. on Information Fusion**, pp. TuC–1–3–TuC1–9, 2001.

[4] C.-Y. Chong and S. Kumar, "Sensor Networks: Evolution, Opportunities and Challenges," **Proceedings of the IEEE**, vol. 91, no. 8, August 2003.

[5] V. Lesser, C. Ortize, and M. Tambe, **Distributed Sensor Networks: A Multiagent Perspective**, Kluwer Academic Publishers, October 2003.

[6] E. H. Jr. Callaway and E. H. Callaway, **Wireless Sensor Networks: Architectures and Protocols**, CRC Press, August 2003.

[7] V.R. Lesser and L.D Erman, "Distributed Interpretation: A Model and an Experiment," **IEEE Trans. Computers**, vol. C29, no. 12, pp. 1144–1163, December 1980.

[8] M. Pauly and K.-F. Kraiss, "Monitoring indoor environments using intelligent mobile sensors," **Proc. 24th IECON Conference IEEE Industrial Electronics Society**, vol. 4, no. 4, pp. 2198–2203, February 1998.

[9] D. J. Bruemmer D. D. Dudenhoeffer and M. L. Davis, "Modeling and simulation for exploring human-robot team interaction requirements," **Proc. 2001 Winter Simulation Conference**, 2001.

[10] G. Weiss (editor), **Multiagent Systems**, The MIT Press, 2001.

[11] G. Roth and M.D. Levine, "Minimal Subset Random Sampling for Pose Determination and Refinement," **World Scientific in Advances in Machine Vision, (C. Archibald, E. Petriu, Edit.)**, pp. 1–21, 1992.

[12] P.K. Allen K.A. Tarabanis and R.Y. Tsai, "A Survey of Sensor Planning in Computer Vision," **IEEE Trans. Robotics Automat.**, vol. 11, no. 1, pp. 86–104, 1995.

[13] R. Abielmona E.M. Petriu, T.E. Whalen and A. Stewart, "Robotic Sensor Agents," **IEEE Instrum. Meas. Mag.**, vol. 7, no. 3, pp. 46–51, 2004.

[14] E.M. Petriu R. Abielmona and T. Whalen, "Heterogeneous Multi-Agent Framework for Environment Monitoring Applications," **Proc. VECIMS'05, IEEE Intl. Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems**, pp. 151–156, July 2005.

[15] T. Kanade, P. Rander, and P. J. Narayanan, "Virtualized Reality: Constructing Virtual Worlds from Real Scenes," **IEEE MultiMedia**, vol. 4, no. 1, pp. 34–47, 1997.

[16] A. Assal and V. Groza, "Agent-based Resource Management for Smart Robotic Sensors," **IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2004**, vol. 4, pp. 2239–2242, May 2004.

[17] R. Abielmona, E. M. Petriu, and V. Groza, "Can an Intelligent Society of Robots Survive in a Hostile Environment?," **IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2003**, vol. 2, pp. 1235–1238, May 2003.

[18] R. Brooks, "A robust layered control system for a mobile robot," **IEEE Journal on Robotics and Automation**, pp. 14–23, 1986.

[19] E.M. Petriu R. Abielmona and V.Z. Groza, "Agent-Based Control for a Dynamically Reconfigurable Mobile Robotic Structure," **Proc. ROSE'2004, IEEE Intl. Workshop on Robotic Sensing**, pp. 24–29, May 2004.

[20] J. Hawkins and S. Blakeslee, **On Intelligence**, OWL Books, 2004.

[21] V. B. Mountcastle, **Perceptual Neuroscience: The Cerebral Cortex**, Harvard University Press, 1998.

[22] R.G. Smith, "The contract net protocol: high-level communication and control in a distributed problem solver," **IEEE Trans. on Computers**, vol. 29, no. 12, 1980.

[23] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent Cooperation Among Communicating Problem Solvers," **IEEE Trans. Computers**, vol. 36, no. 11, pp. 1275–1291, 1987.

[24] N. Carver and V. Lesser, "The DRESUN Testbed for Research in FA/C Distributed Situation Assessment: Extensions to the Model of External Evidence," **Proceedings of the International Conference on Multiagent Systems**, 1995.

[25] K. Decker, "TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms," in **Foundations of Distributed Artificial Intelligence**, G. O'Hare and N. Jennings, Eds. jan 1996, pp. 429–448, Wiley Inter-Science.

[26] D. D. Dudenhoeffer and D. J. Bruemmer, "Command and Control Architectures for Autonomous Micro-Robotic Forces," Tech. Rep. INEEL/EXT-2001-00232, Idaho National Engineering and Environmental Laboratory, 2001.

[27] Parallax Inc., ," Internet, February 2004, http://www.parallax.com.

[28] F. Amigoni and M. Somalvico, "Multiagent Systems for Environment Perception," **AI Environmental Application Conference**, 2003.

[29] G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A Solution to the Simultaneous Localization and Map Building (SLAM) Problem," **IEEE Transactions on Robotics and Automation**, vol. 17, no. 3, pp. 229–241, 2001.

[30] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thurn, "Collaborative Multi-Robot Exploration," **Proceedings of 2000 IEEE International Conference on Robotics and Automation**, pp. 221–228, dec 2000.

[31] I. Rekleitis, G. Dudek, and E. Milios, "Multi-Robot Collaboration for Robust Exploration," **Annals of Mathematics and Artificial Intelligence**, vol. 31, no. 1-4, pp. 7–40, 2001.

[32] D. Ponsa and J. Vitira, "Mobile Monitoring System using an Agent-Oriented Approach," **Proceedgins of the IEEE Int. Conf. on Emerging Technologies and Factory Automation**, vol. 2, pp. 1511–1515, 1999.

[33] A. Halme, P. Appelqvist, P. Jakubik, P. Kahkonen, T. Schonberg, and M. Vainio, "Bacterium Robot Society- A Biologically Inspired Multi-Agent Concept for Internal Monitoring and Controlling of Processes," **IEEE/RSJ International Conference on Intelligent Robots and Systems**, 1996.

[34] Robot Electronics, "CMPS03 Magnetic Compass," June 2006, http://www.robot-electronics.co.uk/htm/cmps3doc.shtml.

[35] Robot Maker, "Infrared Control Freak," June 2006, http://www.robotmaker.co.uk/IRCF/IRCF_benefits_Features.htm.

[36] G. Dudek and M. Jenkin, **Computational Principles of Mobile Robotics**, Cambridge University Press, Cambridge, UK, 2000.

[37] G. Bastin G. Campion and B. D'Andrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," **IEEE Trans. on Robotics and Automation**, vol. 12, no. 1, pp. 47–62, February 1996.

[38] G.W. Lucas, "A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators," **The Rossum Project**, May 2000.

[39] H.R. Everett, **Sensors for Mobile Robots: Theory and Applications**, A.K. Peters, 1995.

[40] R. Joshi and A.C. Sanderson, **Multisensor Fusion: A Minimal Representation Framework**, World Scientific, 1999.

[41] R.R. Brooks and S. S. Iyengar, **Multi-Sensor Fusion: Fundamentals and Applications with Software**, Prentice-Hall Inc., 1998.

[42] L. P. Jones and S. S. Iyengar, "Space and Time Efficient Quadtrees," **IEEE Transactions on Pattern Analysis and Machine Intelligence**, vol. PAMI-6, no. 2, 1984.

[43] A. Elfes, "Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception," **Autonomous Mobile Robots: Perception, Mapping and Navigation**, vol. 1, no. 3, pp. 60–70, 1991.

[44] W.S. McMath, S.K. Yeung, C. Gal, E.M. Petriu, L. Korba, and S. Elgazzar, "Multi-sensor data fusion for mobile robot position recovery," **Proceedings of the Instrumentation and Measurement Technology Conference, IMTC/94**, vol. 1, pp. 181–184, May 1994.

[45] J. Borenstein and Koren, "Histogram In-Motion Mapping for Mobile Robot Obstacle Avoidance," **IEEE Transactions on Robotics and Automation**, vol. 7, no. 4, pp. 535–539, August 1991.

[46] C.E. Shannon, "A mathematical theory of communication," **Bell System Technical Journal**, vol. 27, pp. 379–423, July 1948.

[47] A. Papoulis, **Probability, Random Variables, and Stochastic Processes**, McGraw Hill, second edition, 1984.

[48] P. Lemay, **The Statistical Analysis of Dynamics and Complexity in Psychology: A Configural Approach**, Ph.D. thesis, Universite de Lausanne, 1999.

[49] T. M. Cover and J. A. Thomas, **Elements of Information Theory**, John Wiley, 1991.

[50] Wikipedia the free encyclopedia, "Specification Language," Internet, February 2004, http://en.wikipedia.org/wiki/Specification_language.

[51] C.B. Jones, **Systematic Software Development using VDM**, Series in Computer Science. Prentice Hall International, 1986.

[52] K. Lano, **The B Language and Method: A guide to Practical Formal Development**, Springer, 1996.

[53] J.M. Spivey, **The Z Notation: A Reference Manual**, Series in Computer Science. Prentice Hall International, 2nd edition, 1992.

[54] R. Milner, **Communication and Concurrency**, Series in Computer Science. Prentice Hall International, 1989.

[55] C.A.R. Hoare, **Communicating Sequential Processes**, Series in Computer Science. Prentice Hall International, 1985.

[56] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," **Computer Networks and ISDN Systems**, vol. 14, no. 1, pp. 25–59, 1998.

[57] E.A. Emerson and J.Y. Halpern, "'Sometimes' and 'not never' revisited: on branching time versus linear time temporal logic," **Journal of the ACM**, vol. 33, no. 1, pp. 151–178, 1986.

[58] J. Elgot-Drapkin and D. Perlis, "Reasoning situated in time I: Basic concepts," **Journal of Experimental and Theoretical Artificial Intelligence**, vol. 2, no. 1, pp. 75–98, 1990.

[59] B. Chellas, **Modal Logic: An Introduction**, Cambridge University Press, 1980.

[60] D. Harel, "Statecharts: A visual Formalism for complex systems," **The Science of Computer Programming**, vol. 8, pp. 231–274, 1987.

[61] J.-R. Abrial, "Data Semantics," in **IFIP TC2 Working Conference on Data Base Management**, J. W. Klimbie and K. L. Koffeman, Eds. apr 1974, pp. 1–59, Elsevier Science Publishers (North-Holland).

[62] B. A. Sufrin, "Formal system specification: Notation and examples," in **Tools and Notations for Program Construction**, D. Neel, Ed. Cambridge University Press, 1982.

[63] S. Stepney R. Barden and D. Cooper, **Z in Practice**, Prentice Hall, U.K., 1994.

[64] P. Wegner, "Dimensions of object-based language design," **OOPSLA'87 Proceedings, ACM SIGPLAN Notices**, vol. 22, no. 12, pp. 168–182, 1987.

[65] M. Luck and M. d'Inverno, **Understanding Agent Systems**, Springer-Verlag, 2nd edition, 2004.

[66] D. Carrington, D. Duke, R. Duke, P. King, G. Rose, and G. Smith, "Object-Z: An Object-Oriented Extension to Z," **Formal Description Techniques (FORTE'89)**, 1990.

[67] K. Achetz and Wolfram Schultz, "A formal object-oriented method inspired by fusion and object-z," in **ZUM'97: The Z Formal Specification Notation**, M. Hincky J. Bowen and D. Till, Eds., pp. 92–111. Springer-Verlag, 1997.

[68] G. Smith and R. Duke, "Specifying Concurrent Systems Using Object-Z," **Proceedings 15th Australian Computer Science Conference (ACSC-15)**, pp. 859–871, 1992.

[69] J. Liu J. Sun, J.S. Dong and H. Wang, "Object-Z Web Environment and Projections to UML," **International World Wide Web Conference (WWW-10)**, 2001.

[70] A Hussey and D. Carrington, "Using Object-Z to Specify a Web Browser Interface," **6th Australian Conference on Computer-Human Interaction (OZCHI'96)**, 1996.

[71] G. Kassel and G. Smith, "Model Checking Object-Z Classes: Some Experiments with FDR," **Asia-Pacific Software Engineering Conference (APSEC 2001)**, pp. 445–452, 2001.

[72] F. Kammuller G. Smith and T. Santen, "Encoding Object-Z in Isabelle/HOL," in **2nd International Conference of Z and B Users (ZB 2002)**, M.C. Henson D. Bert, J.P. Bowen and K. Robinson, Eds. 2002, pp. 82–99, Springer-Verlag.

[73] G. Smith, "A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems," **Formal Methods Europe (FME'97), volume 1313 of Lecture Notes in Computer Science**, 1997.

[74] J. Derrick and E. Boiten, **Refinement in Z and Object-Z**, Springer-Verlag, U.K., 2001.

[75] G. Smith, "Combining CSP and Z," Tech. Rep. TRCF-97-1, Universitat Oldenburg, 1997.

[76] M. Benjamin, "A Message Passing System: An Example of Combining CSP and Z," **Z User Workshop: Proc. Fourth Ann. Z User Meeting**, pp. 221–228, dec 1989.

[77] C. Fischer, "CSP-OZ: A Combination of Object-Z and CSP," Tech. Rep. TRCF-97-2, Universitat Oldenburg, Apr. 1997.

[78] J. Padberg H. Ehrig and F. Orejas, "From basic views and aspects to integration of specification formalisms," **Bulletin of the EATCS**, vol. 69, pp. 98108, 1999.

[79] S. Schneider, **Concurrent and Real-time Systems: The CSP Approach**, Wiley, 1st edition, 1999.

[80] B. Mahony and J.S. Dong, "Blending Object-Z and Timed CSP: An introduction to TCOZ," in **Proceedings of the 20th International Conference on Software Engineering (ICSE'98)**, K. Futatsugi, R. Kemmerer, and K. Torii, Eds. apr 1998, pp. 95–104, IEEE Computer Society Press.

[81] H. Wang J. S. Dong and J. Sun, "Semantic Web for Extending and Linking Formalisms," **Formal Methods Europe (FME'02 - FLoC) LNCS**, pp. 587–606, 2002.

[82] C.A.R. Hoare C. Zhou and A.P. Ravn, "A calculus of durations," **Information Processing Letters**, pp. 269–276, 1991.

[83] J. Hoenicke and E.-R. Olderog, "CSP-OZ-DC: a combination of specification techniques for processes, data and time," **Nordic Journal of Computin**, pp. 301–334, 2002.

[84] N. Plat and P. G. Larsen, "An overview of the ISO/VDM-SL standard," **SIGPLAN Not.**, vol. 27, no. 8, pp. 76–82, 1992.

[85] S. Mitra, "Object-oriented specification in vdm++," in **Object Oriented Specification Case Studies**, K. Lano and H. Haughton, Eds., pp. 130–136. Prentice Hall International (UK) Ltd., 1994.

[86] A. C. Wills, "Specification in Fresco," in **Object Orientation in Z**, Stepney et al, Ed. 1992, Springer-Verlag Workshops Series.

[87] J.-R. Abrial, "The B Tool," **VDM Europe**, pp. 86–87, 1988.

[88] M. Butler, "csp2B: A Practical approach to combining CSP and B," **FM'99**, pp. 490–508, 1999.

[89] H. Treharne and Sc. Schneider, "Using a process algebra to control B operations," **IFM'99**, pp. 437–456, 1999.

[90] ISO, "Information processing systems — Open systems interconnection — Estelle — A formal description technique based on an extended state transition model," 1989.

[91] ISO/IEC, "Open distributed processing reference model, part 1: Overview," ITU-T Rec. X.901 — ISO/IEC 10746-1, ISO/IEC, 1995.

[92] C. C. Morgan, **Programming from Specifications**, Prentice Hall International Series in Computer Science, 2nd edition, 1994.

[93] J. Sinclair B. Potter and D. Till, **An Introduction to Formal Specification and Z**, Prentice Hall International Series in Computer Science, 2nd edition, 1991.

[94] J. C. P. Woodcock and J. Davies, **Using Z: Specification, Refinement and Proof**, Prentice Hall, 1996.

[95] C. Fidge, "A Comparative Introduction to CSP, CCS and LOTOS," Tech. Rep. 93-24, University of Queensland, 1994.

[96] C. Fischer and H. Wehrheim, "Failure-Divergence Semantics as a Formal Basis for an Object-Oriented Integrated Formal Method," in **Current Trends in Theoretical Computer Science: The Challenge of the New Century**, G. Rozenberg G. Pavin and A. Salomaa, Eds. 2004, vol. 2, World Scientific.

[97] M. Minksy, **The Society of Mind**, Simon & Schuster, Inc., Rockefeller Center, 1230 Avenue of the Americas, NY, 1986.

[98] The Mathworks, "MATLAB: The Language of Technical Computing," **http://www.mathworks.com**, Last accessed on September 16 2005.

[99] Y. Keren, "Data Structures & Algorithms Toolbox ," **Matlab Central**, Last accessed on June 10 2005.

[100] N. Zolotykh, "MATLAB Pointers Library," **Matlab Central**, Last accessed on June 10 2005.

[101] R. Abielmona et al., "Wireless Intelligent Sensor Network for Environment Monitoring," **CITO 2002 Knowledge Network Conference**, October 2002.

[102] Altera Corp., "Nios Development Kit (Stratix Professional Edition)," **http://www.altera.com/products/devkits/altera/kit-nios_1S10.html**, Last accessed on August 10 2004.

[103] Technological Arts, "9S12 C Microcontroller Family Overview," **http://www.technologicalarts.ca/catalog/index.php/cPath/50_75**, Last accessed on October 28 2004.

[104] R. Abielmona, "Sensor Networks: Research Challenges in Practical Implementations, Physical Characteristics and Applications," **IEEE CommSoc**, November 2003.

[105] IEEE Ottawa Section, "IEEE Ottawa 4th Annual Robotics Competition," May 2006, http://www.ottawaroboticscompetition.org.

[106] Acroname Inc., "Sharp GP2D02 distance measuring sensor datasheet," **www.acroname.com/robotics/parts/SharpGP2D12-15.pdf**, Last seen on September 15 2005.

[107] L. Barello, "Interfacing with GP2D02 sensors," **http://www.barello.net/Papers/GP2D02/index.htm**, Last seen on January 15 2005.

[108] J. Labrosse, **MicroC/OS-II**, CMP Books, second edition, 2002.

[109] A. Dunkels, "Minimal TCP/IP implementation with proxy support," M.S. thesis, SICS - Swedish Institute of Computer Science, 2001.

[110] M. El-Kadri, V. Groza, R. Abielmona, and M. Assaf, "A Just-in-Time Compiler for a Reconfigurable Testing Platform," **IMTC 2006 - Instrumentation and Measurement Technology Conference**, April 2006.

[111] E.M. Petriu, N.D. Georganas, D.C. Petriu, D. Makrakis, and V.Z. Groza, "Sensor-Based Information Appliances," **IEEE Instrum. Meas. Mag.**, vol. 3, no. 4, pp. 31–35, 2000.

[112] Abacom Technologies, "Welcome to ABACOM Technologies," **http://www.abacom-tech.com/**, Last accessed on June 10 2004.

[113] Laipac Tech, "GPS Tracking Systems-GPS Receiver-GPS Unit-GPS OEM-GPS Manufacturer-GPS Devices-GPS Provider," **http://www.laipac.com/**, Last accessed on June 10 2004.

[114] Radiotronix, "Life Without Wires," **http://www.radiotronix.com/**, Last accessed on June 10 2004.

[115] Aerocomm, "ConnexLink Devices," **http://www.aerocomm.com/Devices/link.htm/**, Last accessed on December 05 2005.

[116] R. Brooks, **Flesh and Machines: How Robots will Change Us**, Pantheon Books, 2002.

[117] H. Moravec, **Mind Children: The Future of Human and Robot Intelligence**, Harvard University Press, 1988.

[118] H. Moravec, **Robot: Mere Machine to Transcendent Mind**, Oxford University Press Inc., 1998.

[119] Vizx3D, "Real Time 3D Authoring in X3D," Internet, December 2003, http://www.vizx3d.com/.

[120] ParallelGraphics, "VrmlPad," Internet, December 2003, http://www.parallelgraphics.com/products/vrmlpad/.

[121] ParallelGraphics, "Cortona VRML Client," Internet, December 2003, http://www.parallelgraphics.com/products/cortona/.

[122] ISO/IEC, "Information technology Computer graphics and image processing Extensible 3D (X3D)," 2004.

[123] Sun Microsystems, "Java Bindings for OpenGL API Project," June 2005, https://joal.dev.java.net.

[124] Sun Microsystems, "Java3D API," May 2006, http://java.sun.com/products/java-media/3D.

[125] Sun Microsystems, "Java Bindings for OpenAL/Sound3D Toolkit Project," June 2005, https://joal.dev.java.net.

[126] Sun Microsystems, "Java Input API Project," May 2006, https://jinput.dev.java.net.

[127] Web3D Consortium, ," Internet, April 2006, http://www.xj3d.org.

[128] M. Robinson and P. Vorobiev, **Swing**, Manning Publications Company, February 2003.

[129] Sun Developer Network, "Java Web Start Technology," April 2006, http://java.sun.com/products/javawebstart.

[130] ISO/IEC, "Information technology Computer graphics and image processing Extensible 3D (X3D) language bindings Part 2: Java," 2005.

[131] E.M. Petriu, G.G. Patry, T.E. Whalen, A.H. Al-Dhaher, and V. Z. Groza, "Intelligent Robotic Sensor Agents for Environment Monitoring," **Proc. VIMS 2002 - IEEE International Symposium on Virtual and Intelligent Measurement Systems**, pp. 14–19, May 2002.

[132] Mentor Graphics, "ModelSim: From Creation to Realization," April 2005, http://www.model.com/.

[133] E. Casas, "Integrated Microelectronics Engineering, Module 1," November 1998, http://www.ece.ubc.ca/ edc.

[134] Altera Corp., "Stratix II Device Family Data Sheet," **http://www.altera.com/literature/hb/stx2/stx2_sii5v1_01.pdf**, Last accessed on May 15 2006.

Appendix A

Multi-Agent Platform Simulation Environment Design - UML Diagrams

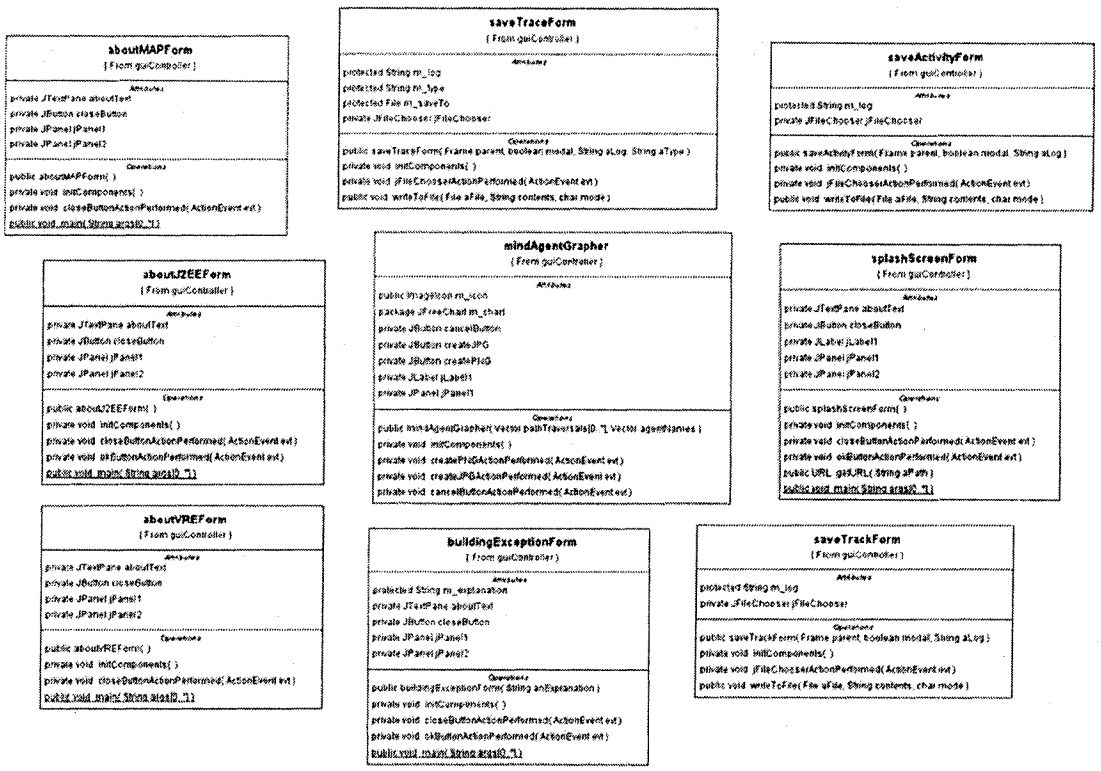


Figure A.1: MAPSE standalone form classes

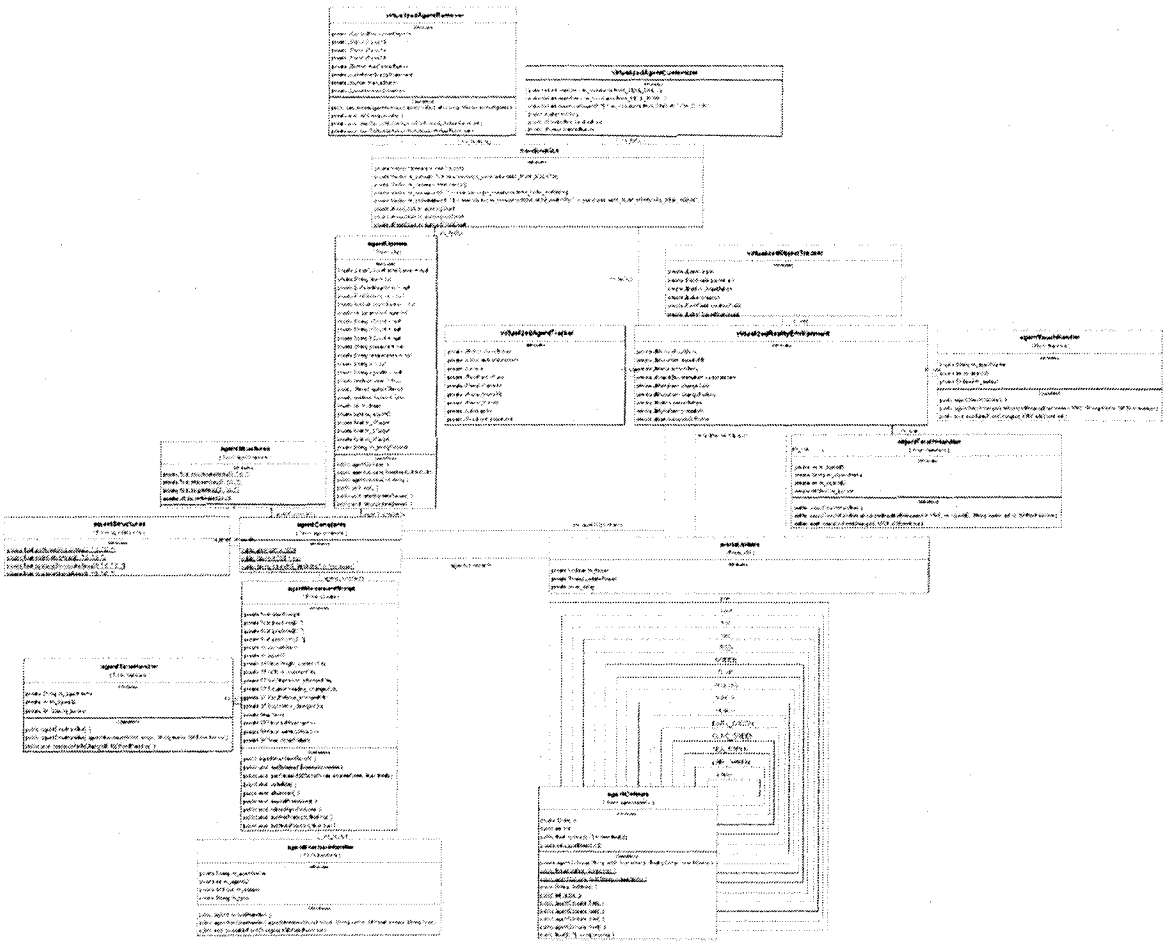


Figure A.2: VRME main classes, including *vre* and *agentUpdateServer* and *worldUpdateServer*

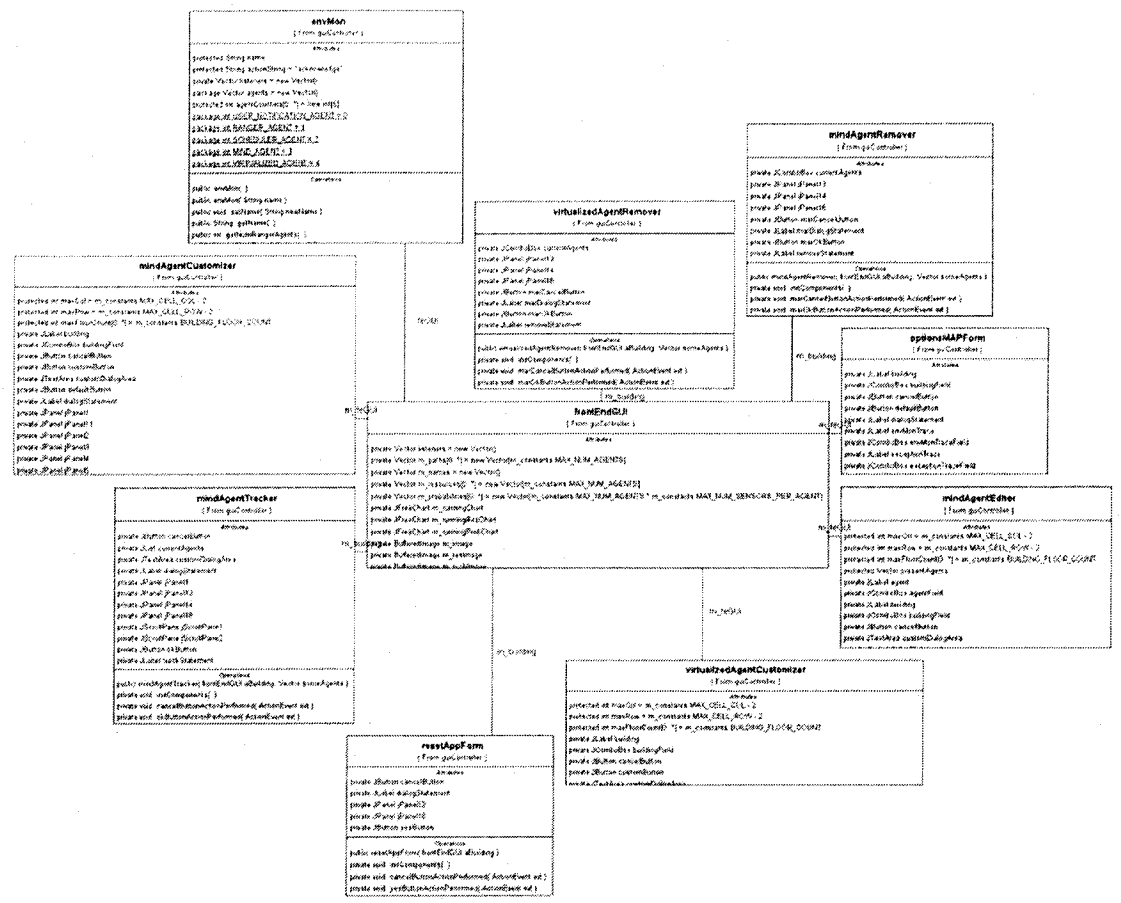


Figure A.3: MAPSE main classes including *envMon* and *feGUI*

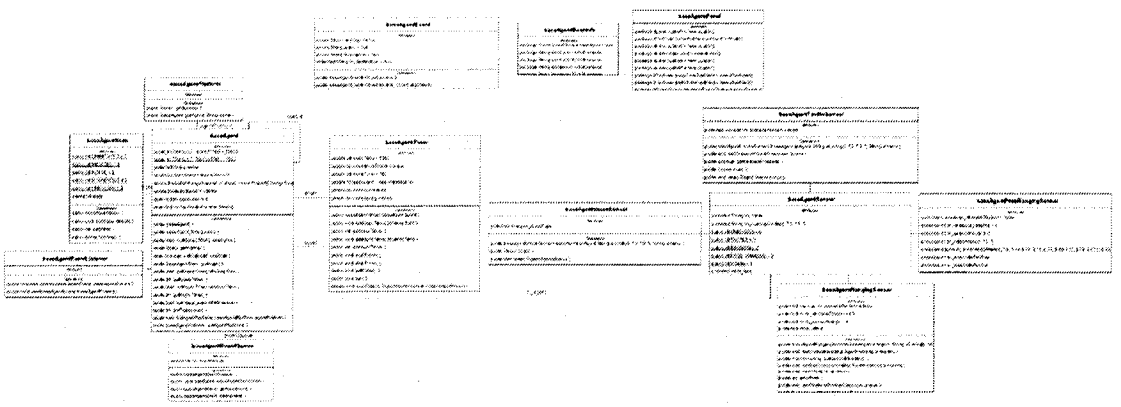


Figure A.4: Base agent classes, including *baseAgent* and *baseAgentListener* and *baseAgent-Timer* and *baseAgentSensor*

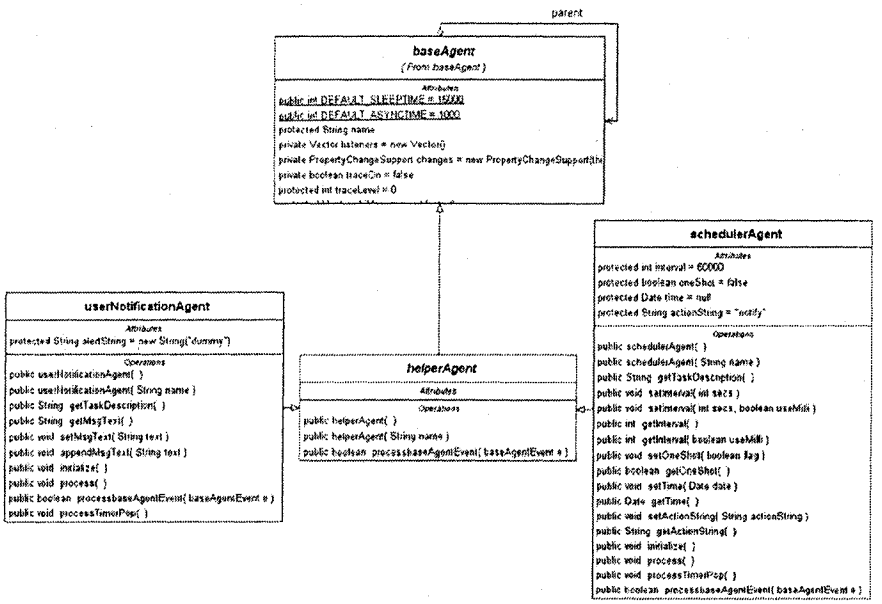


Figure A.5: Helper agent classes, including *helperAgent* and *userNotificationAgent* and *schedulerAgent*

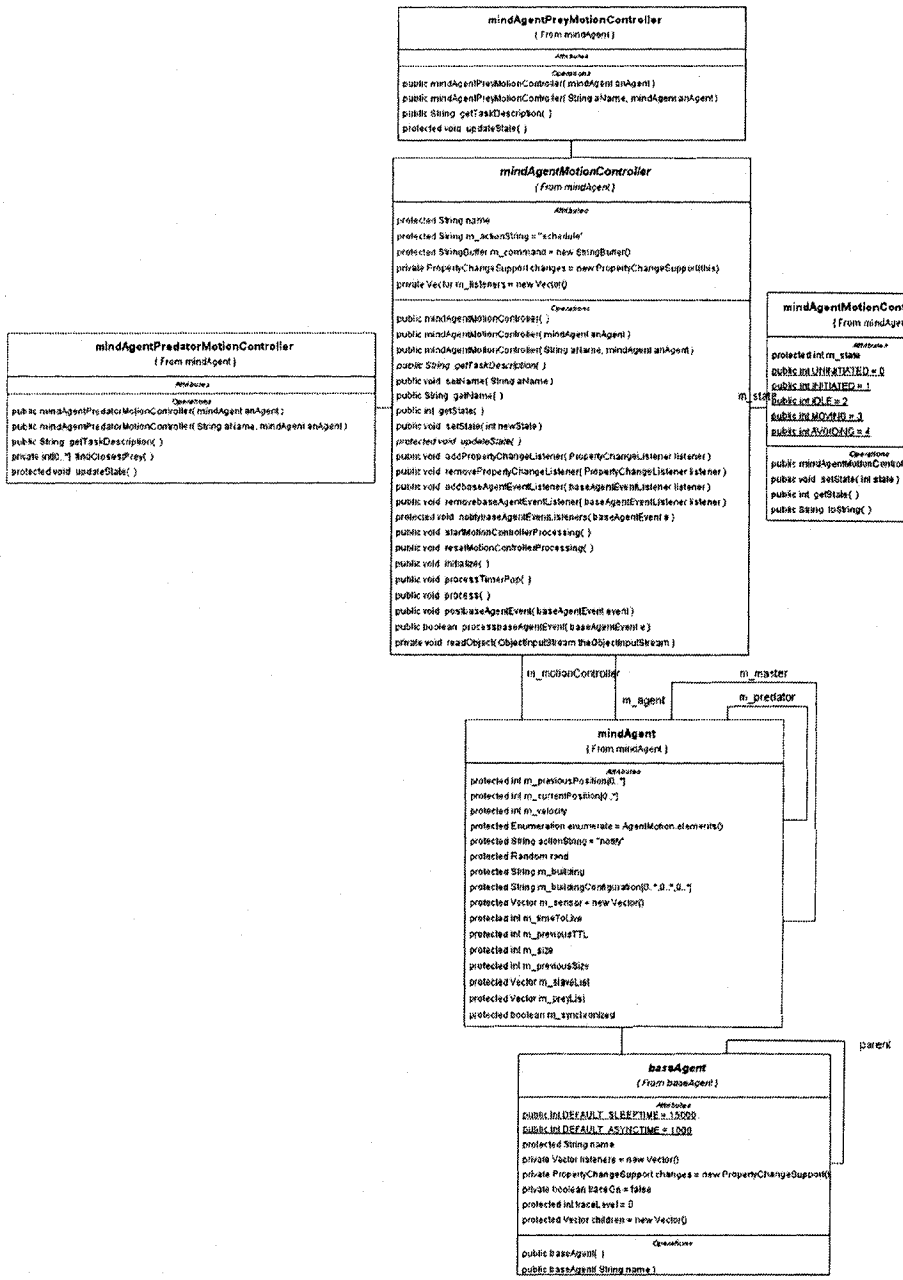


Figure A.6: Mind agent classes, including *mindAgent* and *mindAgentMotionController*

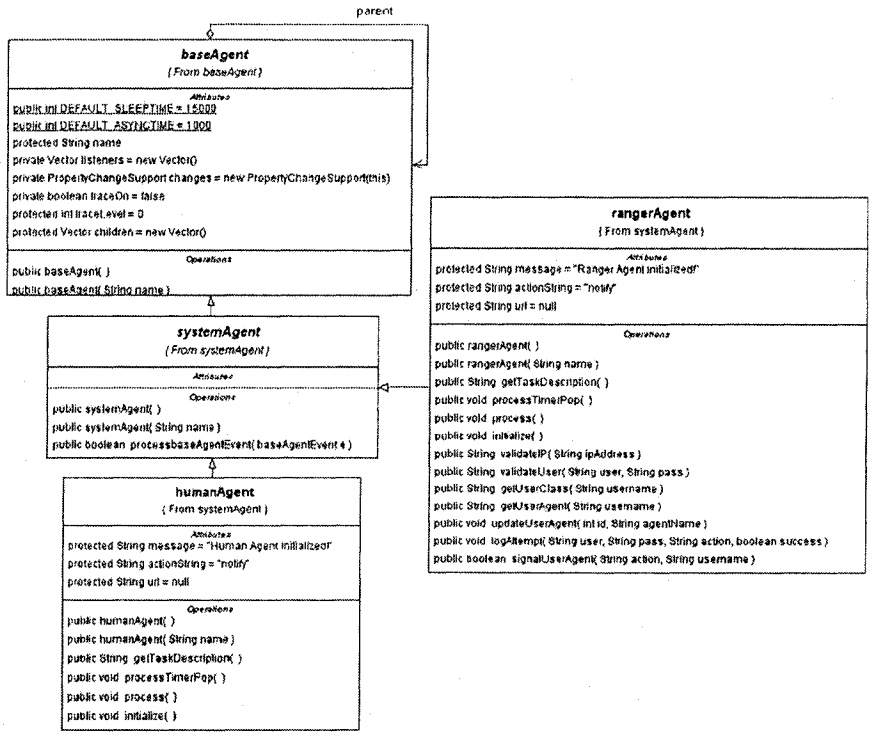


Figure A.7: System agent classes, including *systemAgent* and *humanAgent* and *rangerAgent*

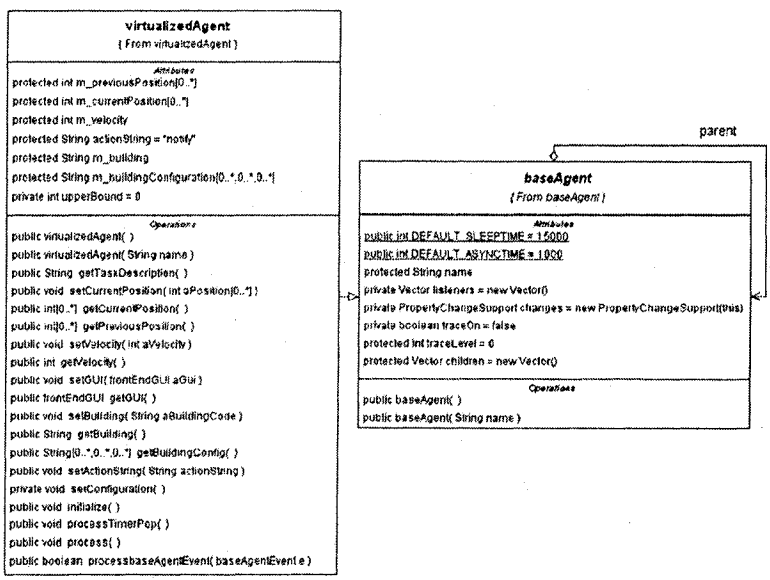


Figure A.8: Virtualized agent classes, including *virtualizedAgent*

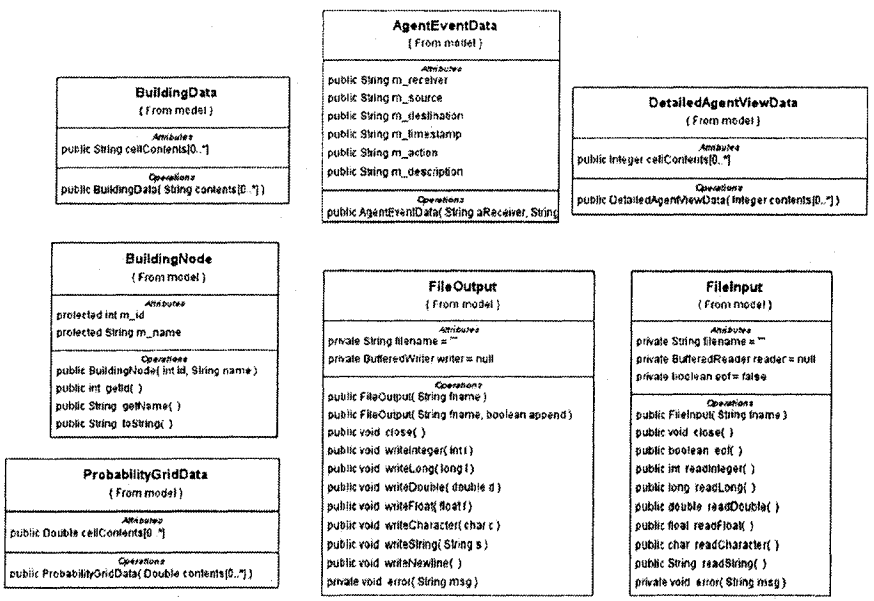


Figure A.9: MAPSE helper classes, including *BuildingData* and *BuildingNode* and *FileOutput* and *FileInput*

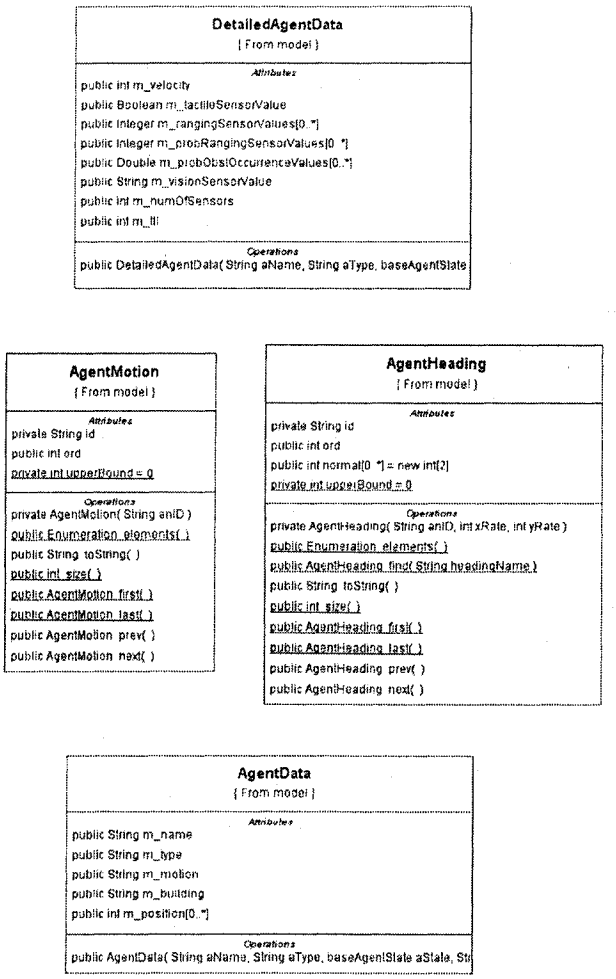


Figure A.10: MAPSE constant classes, including *AgentMotion* and *AgentHeading* and *AgentData* and *DetailedAgentData*

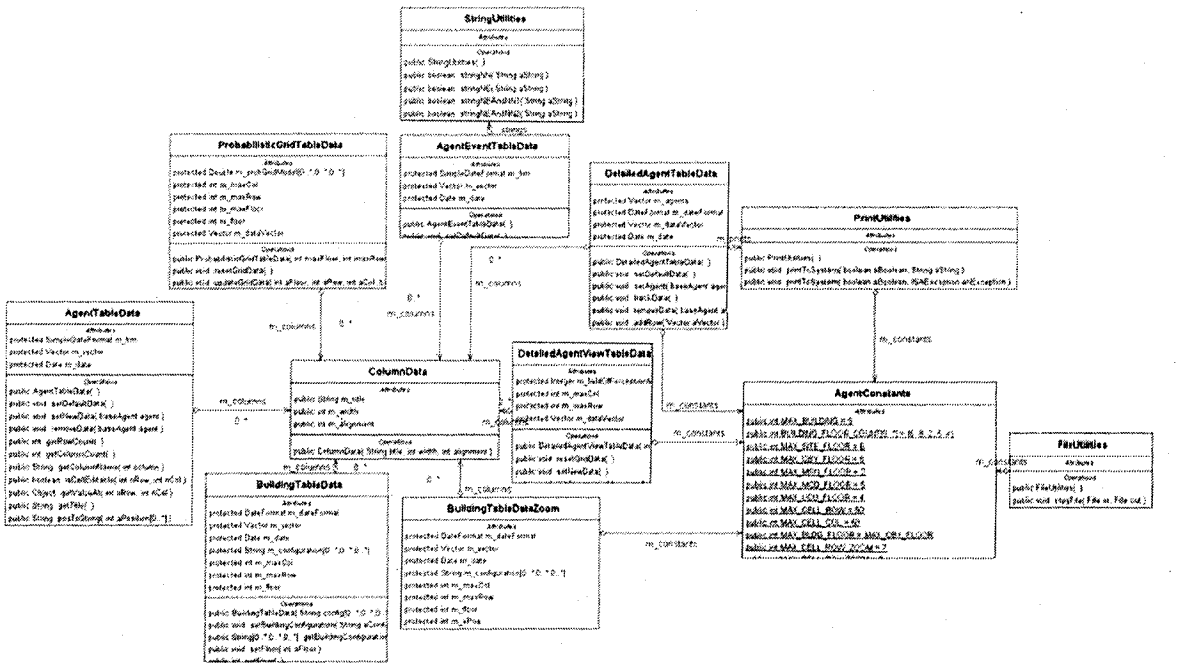


Figure A.11: MAPSE utilities classes, including *StringUtilities* and *PrintUtilities* and *FileUtilities*

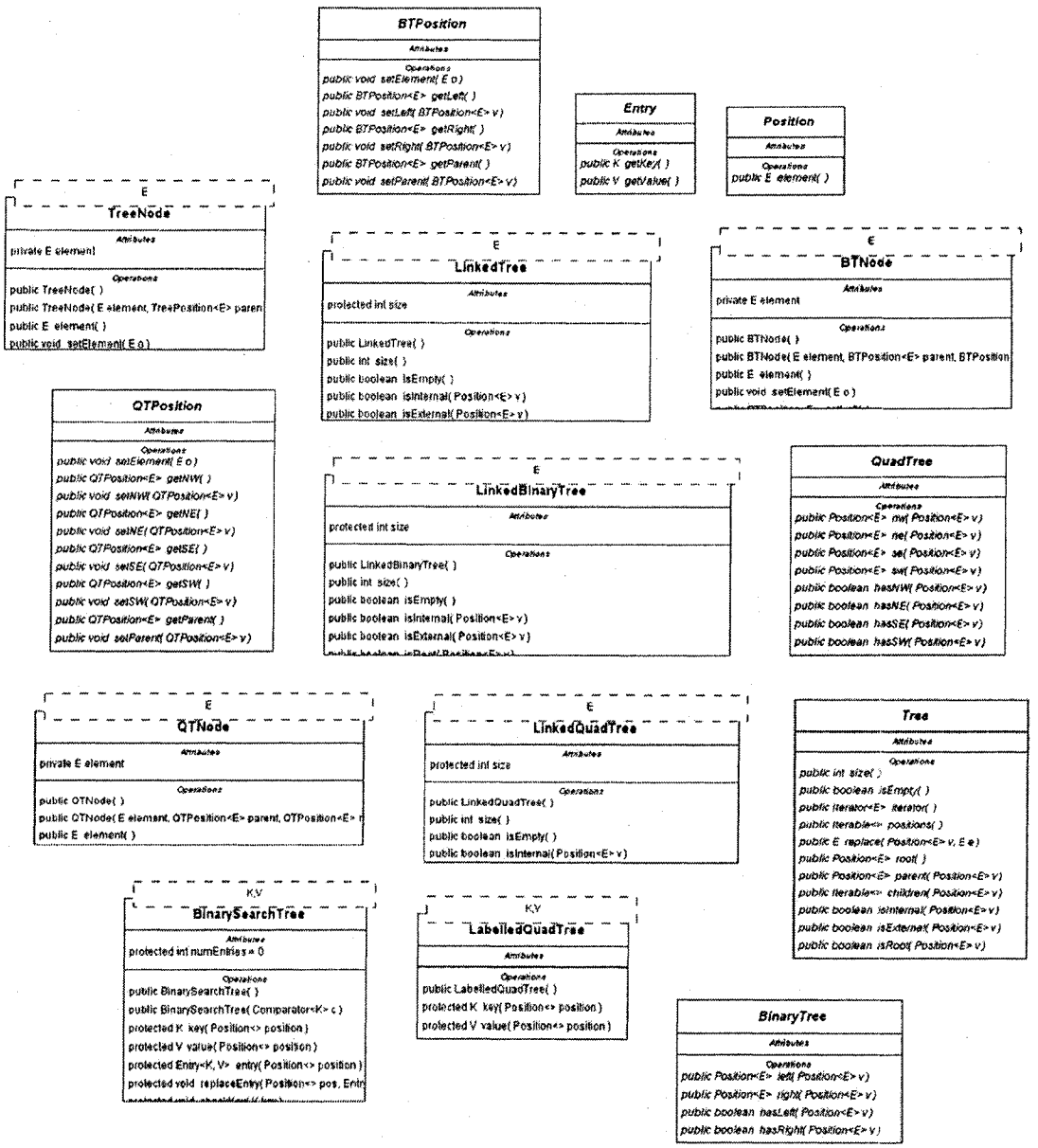


Figure A.12: Labelled quadtree data structure classes, including *QuadTree* and *LinkedQuadTree* and *LabelledQuadTree*

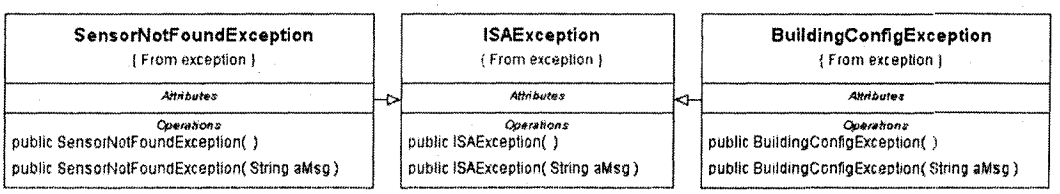


Figure A.13: MAPSE exception classes, including *ISAException* and *SensorNotFoundException* and *BuildingConfigException*

Appendix B

System Architecture

In this appendix, we shall present the open-core design and development that was done to better assimilate the sensors and actuators within the mobile agent's computational resources. These robotic intellectual property (IP) cores have been handed over to our University's Technology Transfer and Business Enterprise office for inclusion as freely available cores, along with their documentation. The functional simulations for the cores are presented in appendix C. Mentor Graphics' Modelsim [132] tool was used during both functional and timing simulation of all of these cores.

B.1 Robotic IP Cores

The designs of each of the following drivers are presented as Altera Avalon bus slave peripherals, such that they can be easily integrated as a component in the Altera SOPC Builder tool: servo motor PWM, IR sensor, wheel encoder sensor, compass sensor and sonar sensor.

B.2 Hardware Development

The hardware development of these peripherals closely follows that presented in application note #333 by Altera Corporation, entitled *Developing Peripherals for SOPC Builder*, where a pulse width modulator (PWM) design is presented as a simple Avalon slave peripheral, as well as a VGA video peripheral as a streaming Avalon slave peripheral. According to the application note, the design of an Avalon slave peripheral is divided into two blocks: the peripheral

task logic and the register file. The former outlines how the peripheral performs its functional operation, while the latter presents the *programmer's view* of the peripheral, with which input is presented to and output is extracted from the peripheral. Both the task logic and the register file, combined with the software drivers, make up a typical Avalon slave peripheral.

Particularly, in this work, the target board is the Nios development board, Stratix-II edition, from Altera; however, the design is modular enough to be synthesized onto other Altera products, as well as other vendors' products. The Stratix-II board has a EP2S60F672C5ES device, with 24,176 adaptive logic modules (ALM) and 2,544,192 bits of on-chip memory.

B.2.1 Servo Motor PWM

A robot is a system that can automate some tasks called *freedoms*. An actuator is a device that makes those freedoms possible. The most important and popular actuator is a motor, that allows the robot to control a wheel, a switch, a joint, or even an arm. We consider here three types of motors: direct current (DC) motors, alternating current (AC) motors and inductive motors.

Motors also have different ways to control them:

- Increasing or decreasing the voltage (DC motors)
- Slowing or speeding up the motor using a feedback loop (servo motors)

In this section, we will mostly concentrate on servo motors. A popular control method for servos is the *pulse width modulation (PWM)* scheme. By varying the pulse width, we can: increase/decrease the speed of the motor, change the direction of the motor or stop the motor. Servo motors require a periodic control signal, that possesses a constant high time (pulse width), the latter determining the rotation speed and direction. Typically, the servo only goes to 180 degrees. We modify the motor to make it spin 360 degrees.

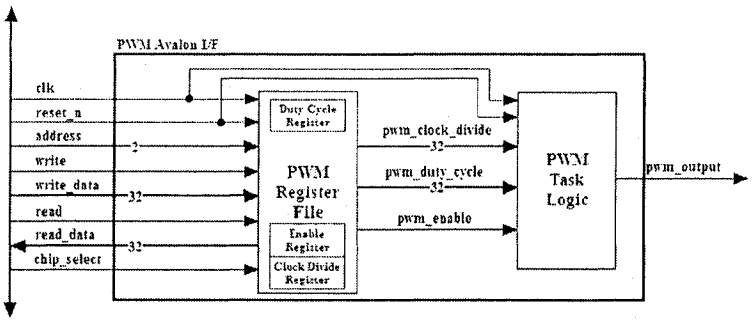


Figure B.1: PWM servo motor peripheral's Avalon interface

B.2.1.1 Peripheral Avalon Interface

Analyzing Figure B.1, we see the peripheral's architecture that, as aforementioned, includes a register file and a task logic. The Avalon interface provides all the signals required to gain access to the register file, as well as to support the operations performed by the task logic. It is up to the designer to connect the task logic to I/O pins on the FPGA, if the peripheral interfaces to an external module. Otherwise, the task logic module's output signals can easily be seen through status registers implemented by the register file.

B.2.1.2 Peripheral Task Logic

Looking at Figure B.2, we notice that the PWM peripheral task logic is simply composed of two 32-bit comparators and one 32-bit up counter. The operation of the task logic is quite simple: a 32-bit comparator compares the clock divider register value to that of the up counter, and when they are equal, controls the counter to reset back to 0, while another 32-bit comparator forces the *pwm_output* signal to be 0 when the counter value is less than or equal to that stored in the duty cycle register, otherwise the signal drives logic value 1. This forces the *pwm_output* signal to be low for *duty_cycle* number of clock cycles, with a period of *clock_divide*. The *clock_enable* signal allows for the enabling (when 1) or disabling (when 0) the PWM output, while the *clk* signal provisions the global synchronizing clock to all internal blocks.

B.2.1.3 Peripheral Register File

The register file provides access to the programmer's view of the peripheral, including data, status and control registers. In this case, as seen in Table B.5, a user can have access to the clock divide value, the duty cycle value and the enable bit. All three registers were given read and write access, allowing software to read back previously written values, and two address bits were decoded. Note that it takes one clock cycle to read and write to the registers, hence no wait-states are used on the Avalon bus in the access of the PWM peripheral.

Register Name	Offset	Access	Description
clock_divide	00	Read/Write	The number of clock cycles counted during one cycle of the PWM output.
duty_cycle	01	Read/Write	The number of clock cycles in which the PWM output will be low.
enable	10	Read/Write	Enables/disables the PWM output. Setting bit 0 to 1 enables the PWM.
Reserved	11	N/A	N/A

Table B.1: PWM servo motor register file and address mapping

B.2.2 IR Sensor

A robot also requires sensors to situate itself in its environment. A sensor is used to obtain characteristics about the environment that feed the robot's internal model. A sensor that is used to inform the robot how far objects are from itself is the triangulation ranging sensor. There are two types of such sensors:

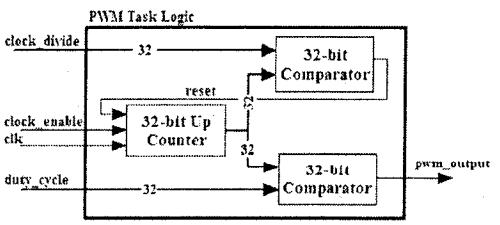


Figure B.2: PWM servo motor peripheral's task logic

(1) Active, radiating some form of energy into the field

(2) Passive, relying on energy emitted by the various objects in the field

In this section, we will mostly concentrate on interfacing to the Sharp family of infrared (IR) sensors, which in this case, fall under the active triangulation system, hence does not require special environment conditions or ambient lighting, nor suffers from the *correspondance problem*, defined as the problem of matching points viewed by one sensor with those of another. However, active triangulation systems suffer from the problems of specular reflection, described as the redirection of incident light at the mirror angle, and surface absorption of the emitted light.

B.2.2.1 Peripheral Avalon Interface

Analyzing Figure B.3, we see the peripheral's architecture that, as aforementioned, includes a register file and a task logic. The Avalon interface provides all the signals required to gain access to the register file, as well as support the operations performed by the task logic. It is up to the designer to connect the task logic to I/O pins on the FPGA, if the peripheral interfaces to an external module. Otherwise, the task logic module's output signals can easily be seen through status registers implemented by the register file. In the IR sensor peripheral, an input signal is read from the IR sensor itself, and an output signal is delivered to it. As well, the peripheral is interrupt based, hence delivers an interrupt signal, *ir_interrupt*, through the Avalon bus.

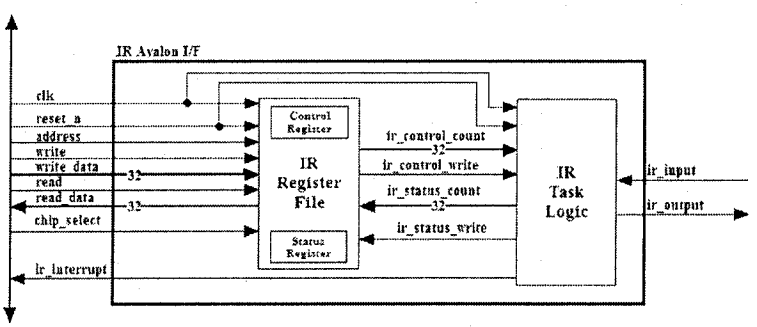


Figure B.3: IR sensor peripheral's Avalon interface

B.2.2.2 Peripheral Task Logic

Looking at Figure B.4, we notice that the IR sensor peripheral task logic is composed of a clock divider, an FSM controller, an 8-bit comparator and interrupt logic. The clock divider divides and synchronizes the FSM controller to a 10 kHz clock. The latter interfaces to the physical IR sensor and extracts the 8-bit distance value, that is compared to the control register's trigger value, and if it is greater than or equal to the latter, and the interrupt enable bit *ir_control_count(31)* is set, and the distance is a valid one, then an interrupt is generated by setting the SR-latch. If a software read occurs on the control register subsequently, then the register file asserts the *ir_control_write* signal, that effectively resets the interrupt line (*ir_interrupt*). The *valid_distance* signal also indicates to the register file that a new status value can be written into the status register, through the write enable signal, *ir_status_write*.

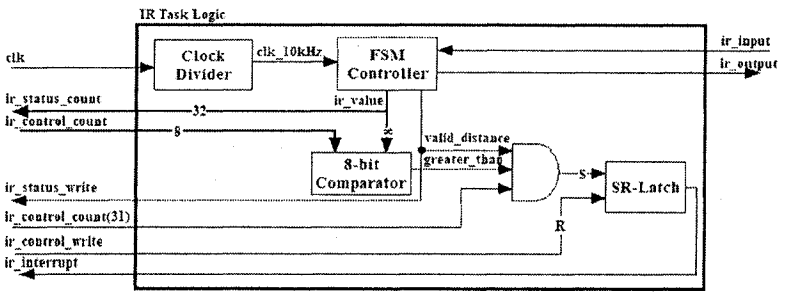


Figure B.4: IR sensor peripheral's task logic

Let us analyze Figure B.5, showing the IR sensor's task logic FSM controller. It is initialized into the *Init* state, where the IR output line, called *Vin*, is set to a logic high. At the next clock cycle, the controller transitions to the *Power Down* state, that checks for the sensor's power down control line to go high (*measure* = '1') and for the inter-measurement delay time (*count* = 1100). If those two events occur, then a transition to the *Start Measure* state occurs, that sets *Vin* to low and waits for a specified delay before jumping to the *Wait Ack* state. In the latter, the machine awaits the IR input line, called *Vout*, to be asserted. If that occurs, a transition to the *Got Ack* state occurs, where *Vin* is de-asserted back to 1. At the next clock cycle, the controller

transitions to the *Send Data* state and then to the *Clock Data* state, alternating back and forth by oscillating *Vin* high and low, while clocking in the data from the IR sensor representing the current distance seen by the physical sensor. After receiving 8 such bits, the controller reverts back to the *Power Down* state to await a new measurement.

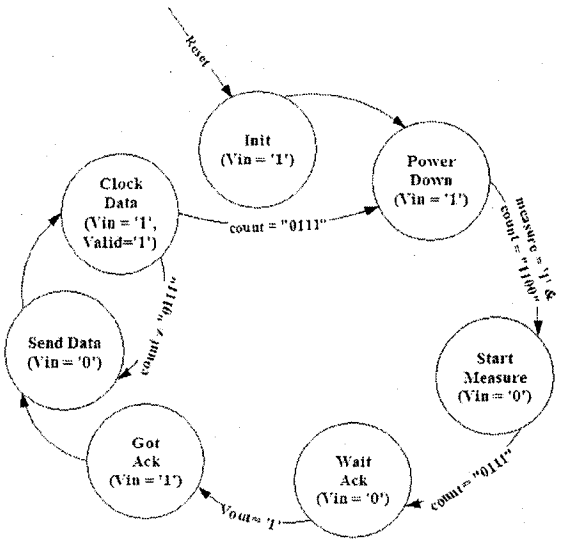


Figure B.5: IR sensor peripheral's FSM controller

B.2.2.3 Peripheral Register File

The register file provides access to the programmer's view of the peripheral, including data, status and control registers. In this case, as seen in Table B.5, a user can have access to the control and status registers. The former is given read and write access but the latter only requires read access, saving on-chip logical resources. One address bit was decoded. Note that it takes one clock cycle to read and write to the registers, hence no wait-states are used on the Avalon bus in the access of the IR peripheral. The interrupt trigger value in the control register allows software to set a 8-bit value, above which, the peripheral generates an interrupt, indicating that the obstacle is within a specific distance away from the sensor.

Register Name	Offset	Access	Description
control_register	0	Read/Write	Bit 31 is the interrupt enable bit, while LSB is the interrupt trigger value.
status_register	1	Read	LSB is current IR value.

Table B.2: IR sensor register file and address mapping

B.2.3 Wheel Encoder Sensor

A sensor that is used to inform the robot of its wheels' angular speeds is the optical wheel encoder. From those values, one can deduce how far and where the robot has travelled. There are two types of such sensors:

(1) Incremental, measuring rotational velocity and inferring relative position

(2) Absolute, measuring angular position and inferring velocity

In this section, we will mostly concentrate on interfacing to the optical shaft encoder sensor, that in this case, falls under the incremental system, hence possesses a simple interface, is low cost, reliable and immune to noise. Absolute optical encoders, on the other hand, possess a more complex interface. They typically use coding schemes such as Gray code, natural binary or BCD. Absolute encoders are better suited for slow and/or infrequent rotations such as steering angle encodings, as opposed to measuring high-speed continuous rotations using incremental encoders.

B.2.3.1 Peripheral Avalon Interface

Analyzing Figure B.6, we see the peripheral's architecture that, as aforementioned, includes a register file and a task logic. The Avalon interface provides all the signals required to gain access to the register file, as well as support the operations performed by the task logic. It is up to the designer to connect the task logic to I/O pins on the FPGA, if the peripheral interfaces to an external module. Otherwise, the task logic module's output signals can easily be seen through status registers implemented by the register file. In the wheel encoder sensor

peripheral, an input signal is read from the shaft encoder sensor itself, indicating whether a transition has occurred from a dark to light surface or vice versa. As well, the peripheral is interrupt based, hence delivers an interrupt signal, *enc_interrupt*, through the Avalon bus.

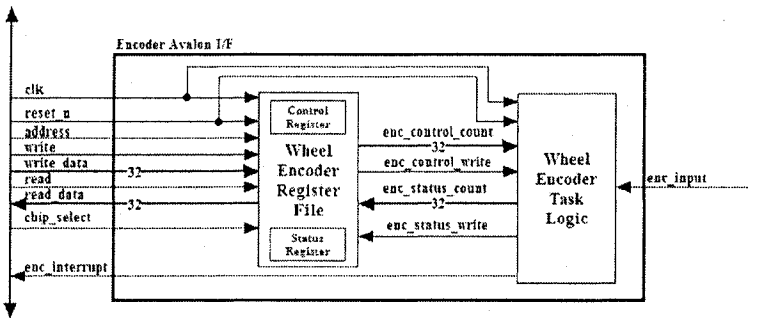


Figure B.6: Wheel encoder sensor peripheral's Avalon interface

B.2.3.2 Peripheral Task Logic

Looking at Figure B.7, we notice that the wheel encoder sensor peripheral task logic is composed of three D flip-flops, a 16-bit counter, a 16-bit comparator and interrupt logic. The first two D flip-flops synchronize the asynchronous *enc_input* signal to the global clock. The third flip-flop, in conjuction with a delayed XOR gate input, is used to detect changes in the actual signal values, indicating that a transition on the wheel encoder has occurred. The event causes a 16-bit counter to increment the current number of detected transitions. The 16-bit encode value is compared to the control register's 16-bit trigger value, and if it is equal to the latter, and the interrupt enable bit *enc_control_count(31)* is set, then an interrupt is generated by asserting the *enc_interrupt* line. If a software read occurs on the control register subsequently, then the register file asserts the *enc_control_write* signal, that effectively resets the interrupt line (*enc_interrupt*). The *sync_change* signal, asserted on each new detected encoder transition, indicates to the register file that a new status value can be written into the status register, through the write enable signal, *enc_status_write*.

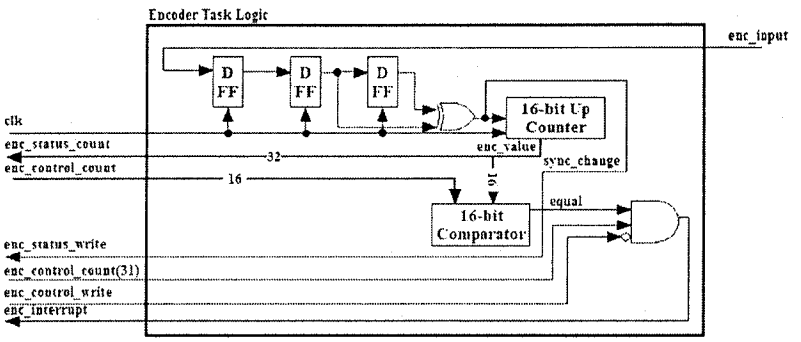


Figure B.7: Wheel encoder sensor peripheral's task logic

B.2.3.3 Peripheral Register File

The register file provides access to the programmer's view of the peripheral, including data, status and control registers. In this case, as seen in Table B.5, a user can have access to the control and status registers. The former is given read and write access but the latter only requires read access, saving on-chip logical resources. One address bit was decoded. Note that it takes one clock cycle to read and write to the registers, hence no wait-states are used on the Avalon bus in the access of the encoder peripheral. The interrupt trigger value in the control register allows software to set a 16-bit value, above which, the peripheral generates an interrupt, indicating that a specified number of encoder counts has been reached.

Register Name	Offset	Access	Description
control_register	0	Read/Write	Bit 31 is the interrupt enable bit, while LS 2 bytes represent the interrupt trigger value.
status_register	1	Read	LS 2 bytes represent current encoder value.

Table B.3: Wheel encoder sensor register file and address mapping

B.2.4 Compass Sensor

A sensor that is used to inform the robot of its heading is the magnetic compass sensor. We consider here six types of such sensors: mechanical magnetic compass, fluxgate compass, magnetoinductive compass, Hall-effect compass, magnetoresistive compass and magentoelastic

compass.

In this section, we will mostly concentrate on interfacing to the CMPS03 compass sensors, that in this case, is a Hall-effect compass, that, in the presence of an external magnetic field, develops a DC voltage across a semiconductor region that is proportional to the magnetic field component at right angles to the direction of current flow. These types of compasses also have simple interface electronics, for example, for our particular compass, both PWM and I2C interfaces are provided to the user.

B.2.4.1 Peripheral Avalon Interface

Analyzing Figure B.8, we see the peripheral's architecture that, as aforementioned, includes a register file and a task logic. The Avalon interface provides all the signals required to gain access to the register file, as well as support the operations performed by the task logic. It is up to the designer to connect the task logic to I/O pins on the FPGA, if the peripheral interfaces to an external module. Otherwise, the task logic module's output signals can easily be seen through status registers implemented by the register file. In the compass sensor peripheral, an input signal is read from the compass sensor itself, indicating the relative heading of the robot, through a PWM signal. Note that the peripheral does not contain any control information, and only returns the angle relative to the magnetic field of the environment.

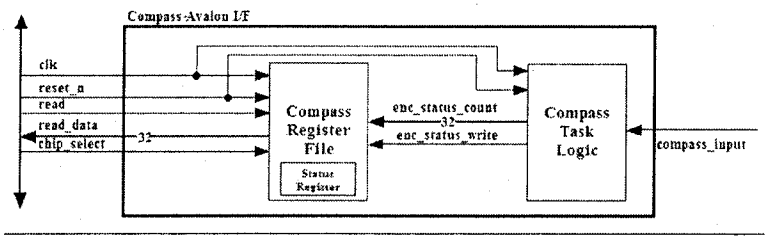


Figure B.8: Compass sensor peripheral's Avalon interface

B.2.4.2 Peripheral Task Logic

Looking at Figure B.9, we notice that the compass sensor peripheral task logic is composed of a clock divider and an FSM controller. The clock divider divides and synchronizes the FSM controller to a 100 kHz clock. The latter interfaces to the physical compass sensor and extracts the 12-bit heading value. The controller also generates a *valid_angle* signal, that when asserted, indicates to the register file that a new status value can be written into the status register, through the write enable signal, *compass_status_write*.

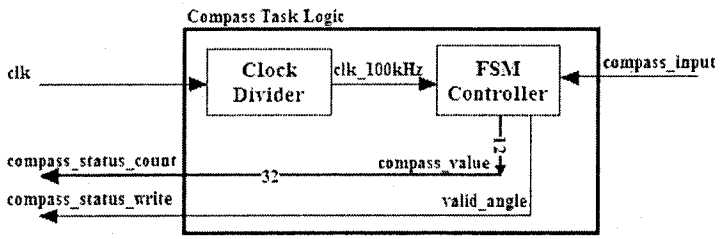


Figure B.9: Compass sensor peripheral's task logic

Let us analyze Figure B.10, showing the compass sensor's task logic FSM controller. It is initialized into the *Waiting For Edge* state, where the PWM clock cycle count is set to 1 and a change in the PWM signal is awaited. Once that occurs, a transition occurs to the *Reading Angle* state, where the count is incremented until the PWM signal goes low again, indicating the length in 100 kHz clock cycles of the high time of the signal. A transition occurs back to the *Waiting for Edge* state, where the count represent the angle value, as being the high time of the read PWM signal (i.e. 100 us multiplied by the number of clock cycles). Once in that state, we await the assertion of the PWM signal to indicate the start of another measurement.

B.2.4.3 Peripheral Register File

The register file provides access to the programmer's view of the peripheral, including data, status and control registers. In this case, as seen in Table B.5, a user can only have access to the status register. The latter is given gead access, saving on-chip logical resources. No

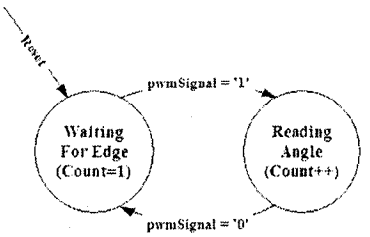


Figure B.10: Compass sensor peripheral's FSM controller

address bits were decoded. Note that it takes one clock cycle to read to the register, hence no wait-states are used on the Avalon bus in the access of the compass peripheral. The compass status value is read as a 32-bit value; however, the least significant 12 bits actually indicate the high time of the PWM signal, and hence the current heading of the robot.

Register Name	Offset	Access	Description
status_register	N/A	Read	LS 12 bits represent current compass value.

Table B.4: Compass sensor register file and address mapping

B.2.5 Sonar Sensor

Another sensor that is used to inform the robot how far objects are from itself is the time-of-flight ranging sensor. We consider here two types of such sensors:

(1) Ultrasonic-based, most popular in mobile robotics

(2) Laser-based, better known as lidar systems

Time-of-flight sensors work by calculating the round-trip distance between the object and the sensor. Note that different signals have different speeds, for example, sound travels at 330 m/s while light travels at 300,000 km/s. In this section, we will mostly concentrate on interfacing to the SRF04 family of sonar sensors, that in this case, fall under the ultrasonic-based ranging system, hence is low cost, and possessed an easy interface. These types of sensors are mostly used for world modeling and collision avoidance, position estimation and motion

detection. Laser-based systems, on the other hand, provide a higher accuracy, typically a few cm over the range of 1-5 meters; however, they require subnanosecond measurements. The latter problem is resolved by a *time-to-amplitude conversion.*

B.2.5.1 Peripheral Avalon Interface

Analyzing Figure B.11, we see the peripheral's architecture that, as aforementioned, includes a register file and a task logic. The Avalon interface provides all the signals required to gain access to the register file, as well as support the operations performed by the task logic. It is up to the designer to connect the task logic to I/O pins on the FPGA, if the peripheral interfaces to an external module. Otherwise, the task logic module's output signals can easily be seen through status registers implemented by the register file. In the sonar sensor peripheral, an input signal is read from the sonar sensor itself, and an output signal is delivered to it. As well, the peripheral is interrupt based, hence delivers an interrupt signal, *sonar_interrupt*, through the Avalon bus.

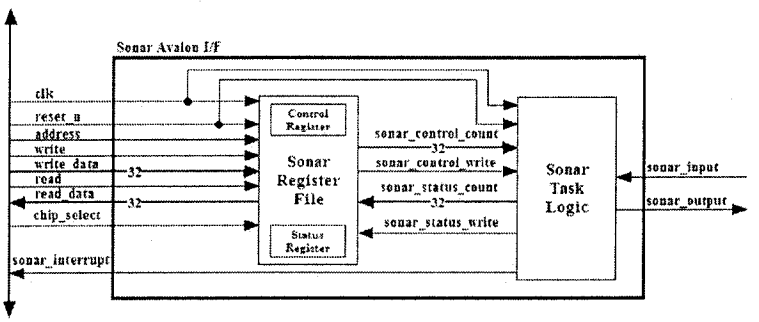


Figure B.11: Sonar sensor peripheral's Avalon interface

B.2.5.2 Peripheral Task Logic

Looking at Figure B.12, we notice that the sonar sensor peripheral task logic is composed of a clock divider, an FSM controller, a 16-bit comparator and interrupt logic. The clock divider divides and synchronizes the FSM controller to a 1 MHz clock. The latter interfaces to the

physical sonar sensor and extracts the 16-bit distance value, that is compared to the control register's trigger value, and if it is greater than or equal to the latter, and the interrupt enable bit *sonar_control_count(31)* is set, and the distance is a valid one, then an interrupt is generated by setting the SR-latch. If a software read occurs on the control register subsequently, then the register file asserts the *sonar_control_write* signal, that effectively resets the interrupt line (*sonar_interrupt*). The *valid_distance* signal also indicates to the register file that a new status value can be written into the status register, through the write enable signal, *sonar_status_write*.

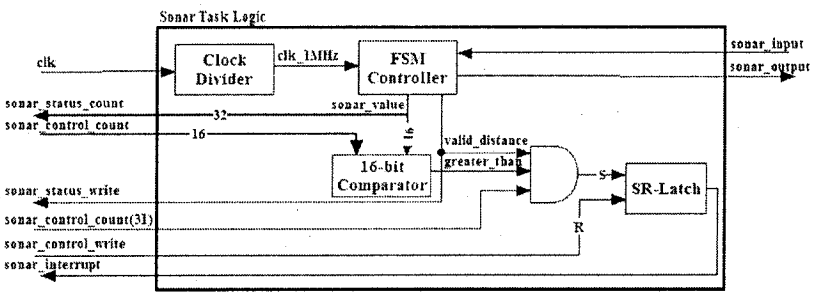


Figure B.12: Sonar sensor peripheral's task logic

Let us analyze Figure B.13, showing the sonar sensor's task logic FSM controller. It is initialized into the *Init* state, where the sonar output line, called *Trig*, is set to a logic low. At the next clock cycle, the controller transitions to the *Power Down* state, that checks for the sensor's power down control line to go high (*measure* = '1') and for the inter-measurement delay time (*count* = 10000), that equates to 10 ms. If those two events occur, then a transition to the *Start Measure* state occurs, that sets *Trig* to high and waits for a specified delay (10 us) before jumping to the *Wait Ack* state. In the latter, the machine awaits the sonar input line, called *echo*, to be asserted. If that occurs, a transition to the *Got Ack* state occurs, where *Trig* is de-asserted back to 0. At the next clock cycle, the controller transitions to the *Send Data* state, where the distance is validated by the assertion of *Valid* and a 16-bit value representing the current distance seen by the physical sensor is sent out of the module. Finally, the controller reverts back to the *Power Down* state to await a new measurement.

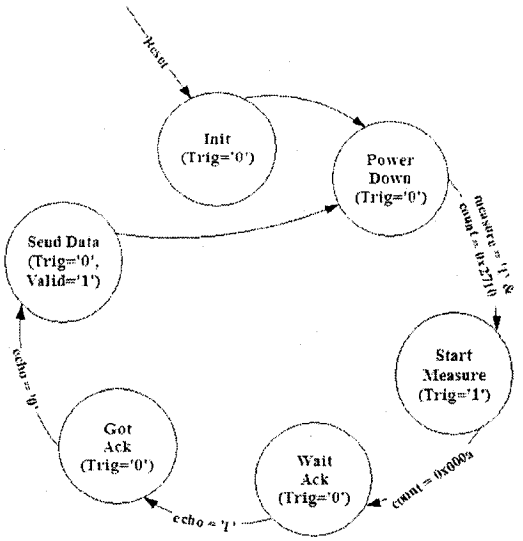


Figure B.13: Sonar sensor peripheral's FSM controller

B.2.5.3 Peripheral Register File

The register file provides access to the programmer's view of the peripheral, including data, status and control registers. In this case, as seen in Table B.5, a user can have access to the control and status registers. The former is given read and write access but the latter only requires read access, saving on-chip logical resources. One address bit was decoded. Note that it takes one clock cycle to read and write to the registers, hence no wait-states are used on the Avalon bus in the access of the sonar peripheral. The interrupt trigger value in the control register allows software to set a 16-bit value, above which, the peripheral generates an interrupt, indicating that the obstacle is within a specific distance away from the sensor.

B.3 Software Development

The software development of these peripherals also closely follows that presented in application note #333 by Altera Corporation, entitled *Developing Peripherals for SOPC Builder*. Once the hardware design has passed functional and timing simulations, it is up to the designer to create a software device driver for the module that includes header files that declare the

Register Name	Offset	Access	Description
control_register	0	Read/Write	Bit 31 is the interrupt enable bit, while LS 2 bytes represent the interrupt trigger value.
status_register	1	Read	LS 2 bytes represent current sonar value.

Table B.5: Sonar sensor register file and address mapping

peripheral-related software structures and constants, as well as the peripheral driver functions to access the peripheral.

B.3.1 Servo Motor PWM

The PWM peripheral's device driver offers the following functions:

int altera_avalon_pwm_init(unsigned int address, unsigned int clock_divide, unsigned int duty_cycle); Initialize the PWM module at *address* with a duty cycle of *duty_cycle* and a clock divide value of *clock_divide*

int altera_avalon_pwm_enable(unsigned int address); Enable the PWM module at *address*

int altera_avalon_pwm_disable(unsigned int address); Disable the PWM module at *address*

int altera_avalon_pwm_change_duty_cycle(unsigned int address, unsigned int duty_cycle); Modify the PWM module at *address* with a duty cycle *duty_cycle*

The PWM peripheral's device driver offers the following macros:

IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER(base) Read the clock divider register at address *base*

IOWR_ALTERA_AVALON_PWM_CLOCK_DIVIDER(base, data) Write to the clock divider register at address *base* a new *data* value

IORD_ALTERA_AVALON_PWM_DUTY_CYCLE(base) Read the duty cycle register at address *base*

IOWR_ALTERA_AVALON_PWM_DUTY_CYCLE(base, data) Write to the duty cycle register at address *base* a new *data* value

IORD_ALTERA_AVALON_PWM_ENABLE(base) Read the enable register at address *base*

IOWR_ALTERA_AVALON_PWM_ENABLE(base, data) Write to the enable register at address *base* a new *data* value

B.3.2 IR Sensor

The IR peripheral's device driver offers the following functions:

int altera_avalon_ir_init(unsigned int address, char interrupt_status, unsigned int interrupt_count); Initialize the IR module at *address* with an *interrupt_count* and check its *interrupt_status*

int altera_avalon_ir_interrupt_enable(unsigned int address); Enable the IR module at *address*

int altera_avalon_ir_interrupt_disable(unsigned int address); Disable the IR module at *address*

int altera_avalon_ir_change_interrupt_count(unsigned int address, unsigned int interrupt_count); Modify the IR module at *address* with a new *interrupt_count*

The IR peripheral's device driver offers the following macros:

IORD_ALTERA_AVALON_IR_CONTROL_REGISTER(base) Read the control register at address *base*

IOWR_ALTERA_AVALON_IR_CONTROL_REGISTER(base, data) Write to the control register at address *base* a new *data* value

IORD_ALTERA_AVALON_IR_STATUS_REGISTER(base) Read the status register at address *base*

B.3.3 Wheel Encoder Sensor

The wheel encoder peripheral's device driver offers the following functions:

int altera_avalon_enc_init(unsigned int address, char interrupt_status, unsigned int interrupt_count); Initialize the wheel encoder module at *address* with an *interrupt_count* and check its *interrupt_status*

int altera_avalon_enc_interrupt_enable(unsigned int address); Enable the wheel encoder module at *address*

int altera_avalon_enc_interrupt_disable(unsigned int address); Disable the wheel encoder module at *address*

int altera_avalon_enc_change_interrupt_count(unsigned int address, unsigned int interrupt_count); Modify the wheel encoder module at *address* with a new *interrupt_count*

The wheel encoder peripheral's device driver offers the following macros:

IORD_ALTERA_AVALON_ENC_CONTROL_REGISTER(base) Read the control register at address *base*

IOWR_ALTERA_AVALON_ENC_CONTROL_REGISTER(base, data) Write to the control register at address *base* a new *data* value

IORD_ALTERA_AVALON_ENC_STATUS_REGISTER(base) Read the status register at address *base*

B.3.4 Compass Sensor

The compass peripheral's device driver offers the following function:

int altera_avalon_comp_init(unsigned int address); Initialize the compass module at *address*

The compass peripheral's device driver offers the following macro:

IORD_ALTERA_AVALON_COMP_STATUS_REGISTER(base) Read the status register at address *base*

B.3.5 Sonar Sensor

The sonar peripheral's device driver offers the following function:

int altera_avalon_sonar_init(unsigned int address, char interrupt_status, unsigned int interrupt_count); Initialize the sonar module at *address* with an *interrupt_count* and check its *interrupt_status*

int altera_avalon_sonar_interrupt_enable(unsigned int address); Enable the sonar module at *address*

int altera_avalon_sonar_interrupt_disable(unsigned int address); Disable the sonar module at *address*

int altera_avalon_sonar_change_interrupt_count(unsigned int address, unsigned int interrupt_count); Modify the sonar module at *address* with a new *interrupt_count*

The sonar peripheral's device driver offers the following macro:

IORD_ALTERA_AVALON_SONAR_CONTROL_REGISTER(base) Read the control register at address *base*

IOWR_ALTERA_AVALON_SONAR_CONTROL_REGISTER(base, data) Write to the control register at address *base* a new *data* value

IORD_ALTERA_AVALON_SONAR_STATUS_REGISTER(base) Read the status register at address *base*

B.4 Mechanical Switch Bounce

To finalize this appendix, let us take a look at an inherent problem when designing for the detection of clean push-button events. As contacts in a switch are made of metal, and since one of the contacts is mobile, contacts will bounce as they close and open a switch, causing what is termed as *switch bounce*. This will occur for a few milliseconds, until the contacts will finally settle in a closed position. During these first few milliseconds, it is imperative for a digital circuit designer to be aware that the sensory readings that are changing must be analyzed only when the contacts come to rest. Imagine if you will, a scheme to count how many times the reset button is pressed on a workstation. If the output was handled without taking care for switch bounce, by the time we reach our tenth reset sequence, the counter might be already at 30 or even higher. Thus, it is necessary to take care of this phenomenon.

Now, let us examine a typical two-contact switch that is opening (refer to Figure B.14). Ignoring for now the fact that there is time delay while charging up the capacitor, it can be seen that a clean transition from low to high exists when the switch opens.

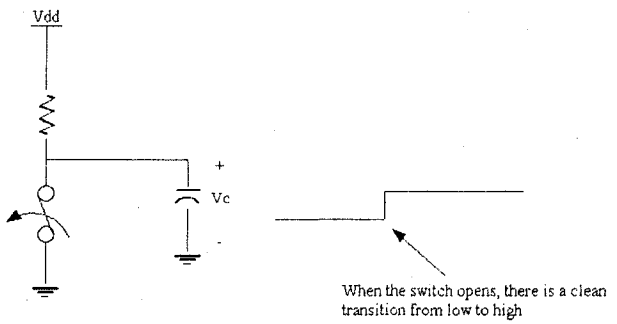


Figure B.14: An open two-contact switch

On the other, let us examine the same circuit, but in this case, the switch is being closed (refer to Figure B.15). In this case, as the switch is closing, there exists a period of output instability, while the switch is bouncing, that causes the output to bounce around the high and low values, until settling on the transient value.

This issue not only exists in two-contact switches, but also in **push-button switches**,

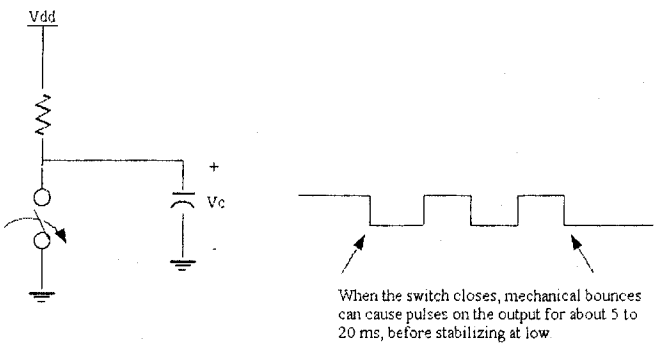


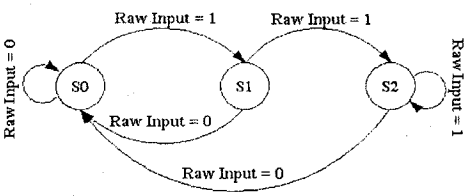
Figure B.15: A closed two-contact switch

toggle switches and electro-mechanical relays.

B.4.1 Realized Solution

Since our Stratix-II board has four main push-button switches, each switch was sent through a debouncer module before feeding the rest of the system. The realization solution is a simple finite state machine (FSM) that effectively drowns out switch bounce, while synchronizing to the system clock [133].

Perusing Figure B.16, we can notice that this particular state machine only has three states. The number of states can be easily calculated as it is a factor of the predicted bounce time and the internal clock frequency. In this case, the clean debounced signal is only true, when the raw input has been detected for at least two edges of the clock. If the input bounces after the first detection (and hence we are in S1), the state machine resets back to S0, and waits for another input transition. From the state transition table, we can observe that the output is only set to high when the state machine actually reaches S2.



State	Output (Debounced Input)
S0 (00)	0
S1 (01)	0
S2 (10)	1
(11)	Undefined

Figure B.16: FSM diagram and state transition table of deboucing solution

Appendix C

Robotic IP Cores - Simulation Results

C.1 IR Sensor

To verify the correct functionality of the hardware design, it is necessary to functionally simulate it. Figure C.1 demonstrates typical operation of the IR sensor peripheral, where a value of 0x00000005 is written into the control register, indicating that the trigger value is 5, while the interrupt is disabled. The control register holds that value. Then, a value of 0x800000C0 is written into the control register switching the trigger value to 192 and enabling the interrupt. A value is then read from the IR sensor (0x000000C5) through its status register, that is greater than the trigger value, causing the interrupt line to be asserted around 1.02 us. The interrupt is de-asserted by a software write to the control register, that occurs at 1.1 us, when a new value of 0x800000B0 is written in.

C.2 Wheel Encoder Sensor

To verify the correct functionality of the hardware design, it is necessary to functionally simulate it. Figure C.2 demonstrates typical operation of the wheel encoder sensor peripheral, where a value of 0x00000005 is written into the control register, indicating that the trigger value is 5, while the interrupt is disabled. The control register holds that value. Then, a value of 0x80000005 is written into the control register switching the trigger value to 5 and enabling the interrupt. Encoder transitions are then detected from the wheel encoder sensor (0x00000005) through its status register, that reach a value that is equal to that of the interrupt trigger value,

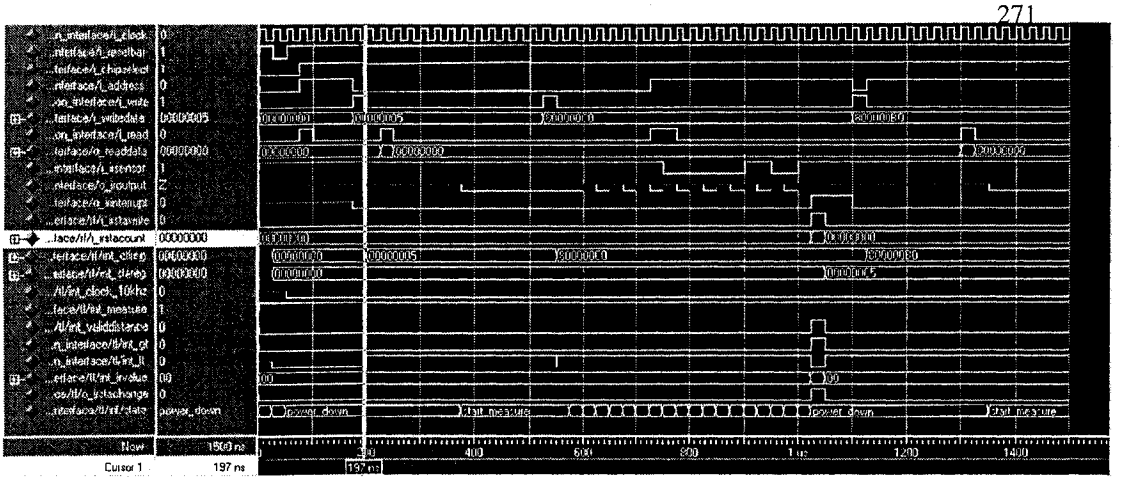


Figure C.1: IR sensor peripheral's functional simulation

causing the interrupt line to be asserted around 650 ns. The interrupt is de-asserted by a software write to the control register, that occurs at 900 ns, when a new value of 0x80000007 is written in. The next interrupt is generated around 1250 ns, when the transition counter indicates a value of 7.

C.3 Compass Sensor

To verify the correct functionality of the hardware design, it is necessary to functionally simulate it. Figure C.3 demonstrates typical operation of the compass sensor peripheral, where as soon as both *chip_select* and *read* are asserted, the peripheral has been accessed, and a measurement begins. The PWM signal high time is measured by calculating the number of clock cycles between the first time it goes high (i.e. around 350 ns) and the next time it goes low (i.e. around 750 ns). That number, 16, is stored in the status register of the peripheral's register file, that is sent out to the read data bus signals of the Avalon bus (*o_read_data*) at 870 ns.

C.4 Sonar Sensor

To verify the correct functionality of the hardware design, it is necessary to functionally simulate it. Figure C.4 demonstrates typical operation of the sonar sensor peripheral, where

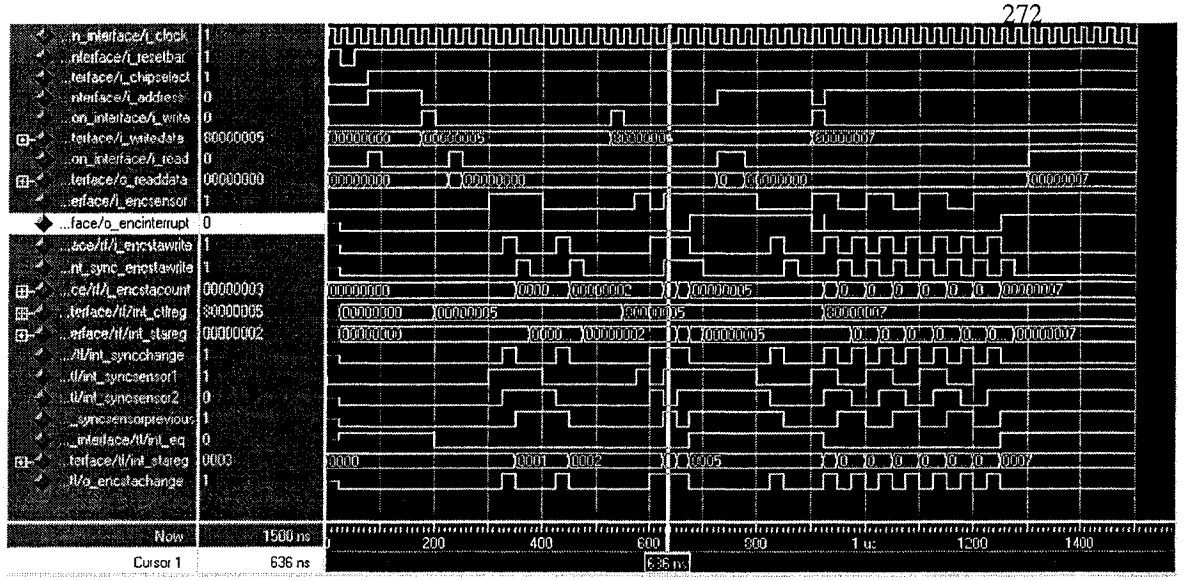


Figure C.2: Wheel encoder sensor peripheral's functional simulation

a value of 0x00000005 is written into the control register, indicating that the trigger value is 5, while the interrupt is disabled. The control register holds that value. Then, a value of 0x80000010 is written into the control register switching the trigger value to 10 and enabling the interrupt. A value is then read from the sonar sensor (0x0000001D) through its status register, that is greater than the trigger value, causing the interrupt line to be asserted around 2.245 us. The interrupt is de-asserted by a software write to the control register, that occurs at 2.45 us, when a new value of 0x800016A8 is written in.

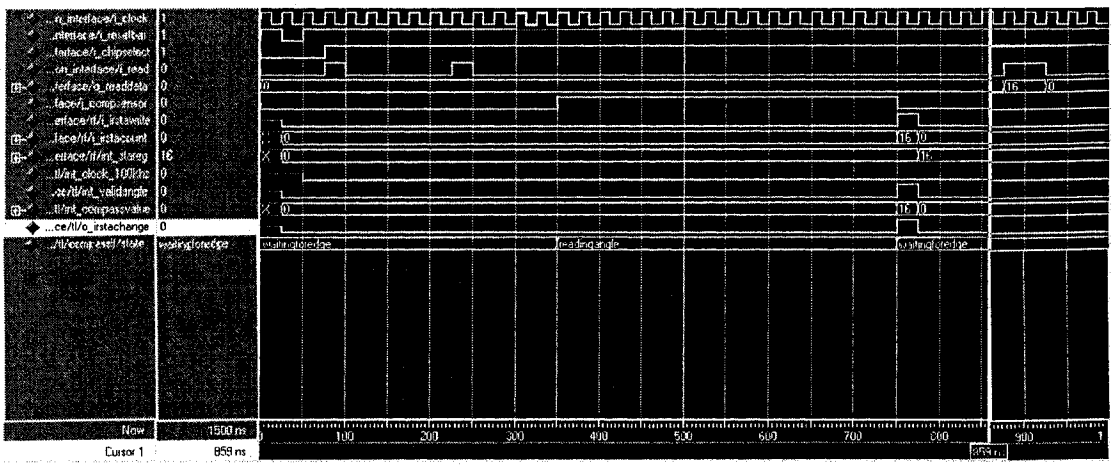


Figure C.3: Compass sensor peripheral's functional simulation

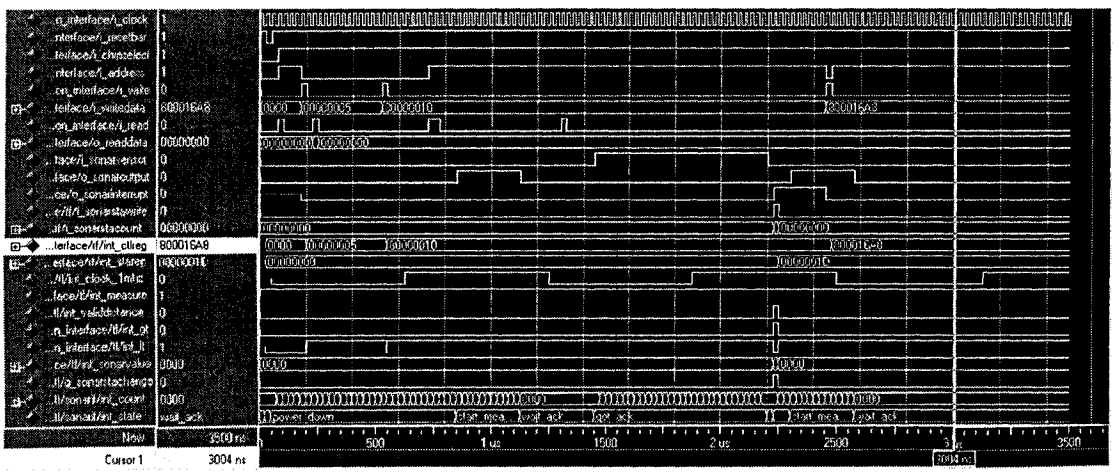


Figure C.4: Sonar sensor peripheral's functional simulation

Appendix D

Agent Hardware/Software Details

Delving into the Stratix-II FPGA, we find numerous components, controllers and cores that together, make up the *System-on-Chip (SoC)*, that makes the agent function. As seen in Figure D.1, the heart of the hardware is the *Altera Nios-II/s soft core processor*. The latter is a 32-bit RISC configurable processor that operates at upwards of 57 Dhrystone MIPS. We have noticed that there is a major speedup when we configure the processor with 4 KB each of instruction and data cache. The controllers that interface to the various external components are all Avalon bus slaves, and hence reside within the processor's address space. Two timers exist in the system, one to provide a high-resolution (1 μs) scheduling tick for the processor's RTOS, while another timer is used for application profiling, and hence has a lower resolution (1 *ms*). Three serial (RS-232) ports were used, one for the wireless modem communications, another for console terminal debugging and a third for the IR beacon and communications. An SPI controller was also realized to interface to a dual-axis accelerometer that was later dropped since it was not needed in our experiments.

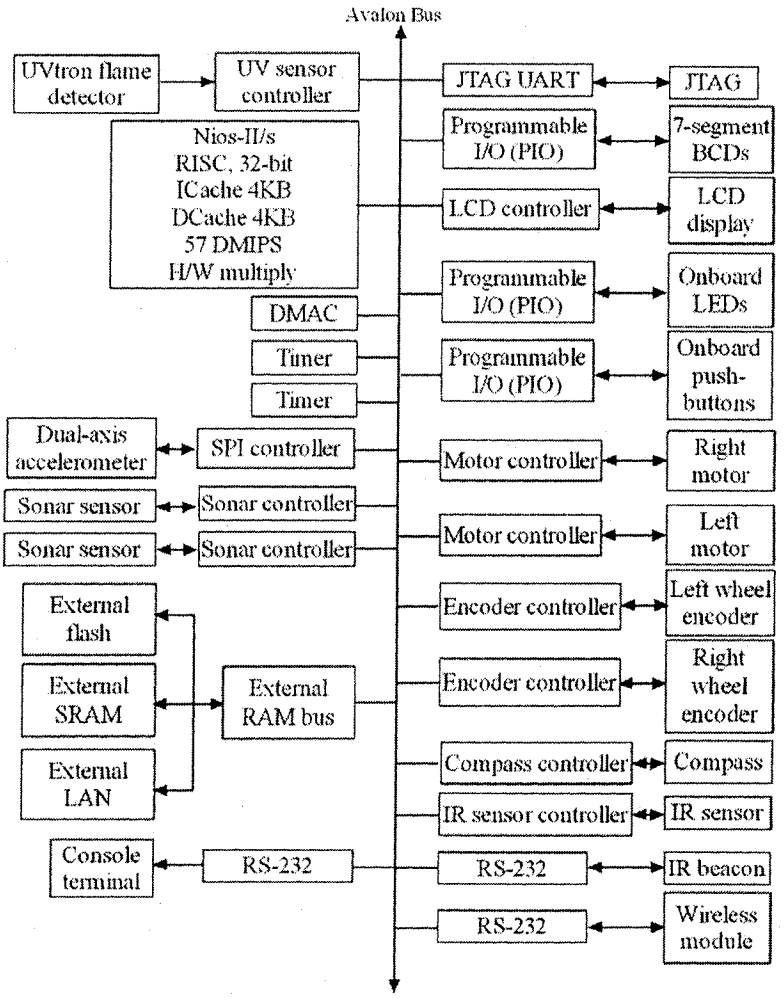


Figure D.1: Altera Nios-II SoC architecture

Our final design report (Figure D.3) shows that we are utilizing 10% of the total adaptive look-up tables (ALUTs), 22% of the total memory bits, 40% of the total pins and 3308 total registers. Considering the functionality and design constraints, we have managed to meet all of our criteria and still have a lot of room in the FPGA for future improvements. Our FPGA layout (Figure D.3) also shows that we are utilizing approximately 25% of the logical area of the FPGA (shown shaded in the figure). That leaves 45,000 logical elements available for future improvements, since each EP2S60 (utilized Stratix-II FPGA) contains 60,440 logical elements [134].

Flow Summary

Flow Status	Successful - Mon Jan 16 16:39:16 2006
Quartus II Version	5.0 Build 148 04/26/2005 SJ Full Version
Revision Name	standard
Top-level Entity Name	roboal
Family	Stratix II
Device	EP2S60F672C5ES
Timing Models	Final
Met timing requirements	Yes
Total ALUTs	5,315 / 48,352 (10 %)
Total registers	3308
Total pins	199 / 493 (40 %)
Total virtual pins	0
Total memory bits	571,136 / 2,544,192 (22 %)
DSP block 9-bit elements	8 / 288 (2 %)
Total PLLs	1 / 6 (16 %)
Total DLLs	0 / 2 (0 %)

Figure D.2: Altera Quartus-II flow summary

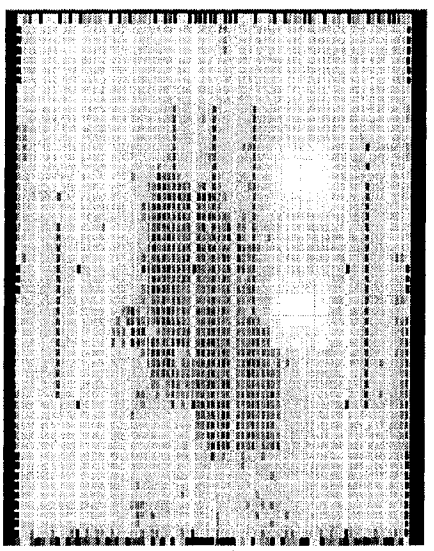


Figure D.3: Altera Quartus-II chip layout

The author ported the RTOS-IP stack combination mentioned above for the Nios-I soft-core processor. The developped port helps users transform their Stratix board, and in particular their Nios-I processor, into a networked real-time device. Using the combination of the uC/OS-II real-time kernel and the lightweight IP stack (lwIP), any designer can easily design real-time systems with internetworking capabilities. The port contained all the files required to generate the Nios processor, along with numerous embedded and networking components.

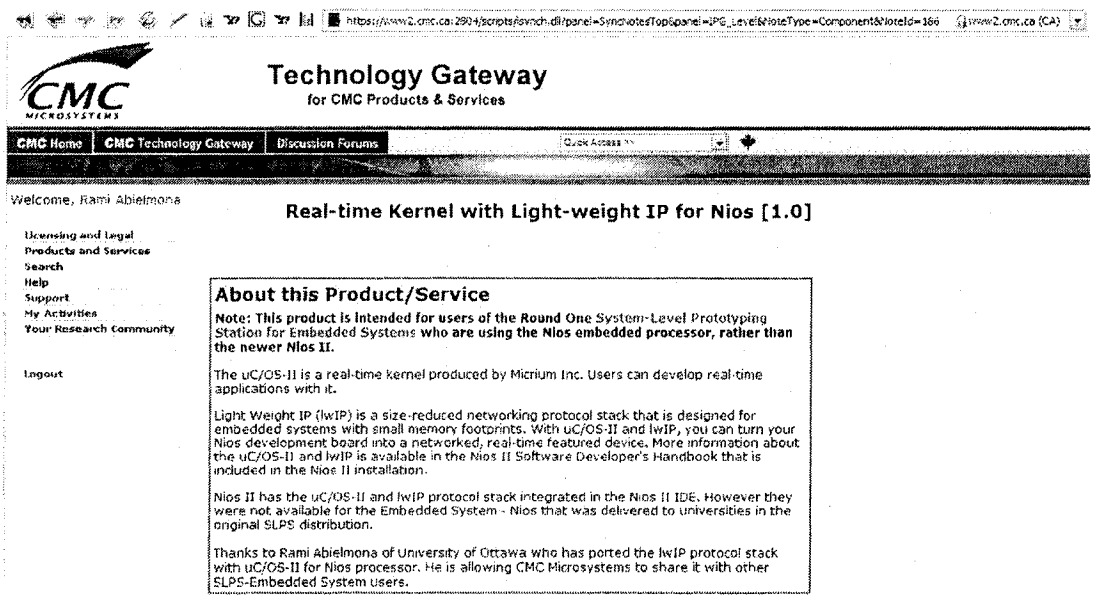


Figure D.4: CMC Microsystems citation for the Nios-I port