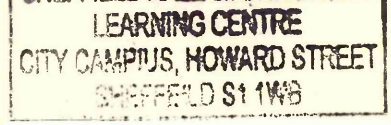
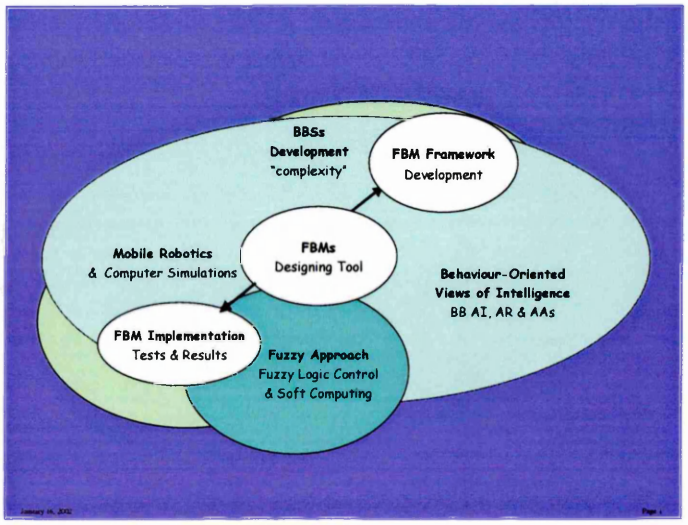


Explorations Into the Behaviour-Oriented Nature of Intelligence: Fuzzy Behavioural Maps



By

Ana María González de Miguel

Computing and Managament Sciences

SHU, UK

Acknowledgements

To my supervisors

I thank Peter Collingwood for being a great supervisor. I am grateful for all his advice, enormous patience and continuing support.

Many thanks to Pascale Vacher and Rick Osborn for their research-related, thesis-related and, life-related advice during all my years in Sheffield.

Explorations Into the Behaviour-Oriented Nature of Intelligence: Fuzzy Behavioural Maps

by Ana María González de Miguel

Abstract

This thesis explores the behaviour-oriented nature of intelligence and presents the definition and use of **Fuzzy Behavioural Maps** (FBMs) as a flexible development framework for providing complex autonomous agent behaviour.

The explorations into the behaviour-oriented nature of intelligence provide an overall view of what has been achieved and, what remains to be done to prove the validity of the **Behaviour-Based** (BB) **approach to Artificial Intelligence**. The thesis elaborates the fundamentals of this field, identifies many contributions (ideas, tendencies and, perspectives) from **Autonomous Agents** and, summarises recent implications and research issues around the realisation of truly artificial autonomous creatures. Further, it concentrates on the **development problem**, describes current approaches to develop **BB Systems** and, discusses some technical issues found with the application of these various approaches to construct **complex BB models**.

FBMs are appropriate tools for **prototyping BB models**. The thesis explores numerous **interpretations and uses of FBMs**. It describes several ways to interpret the design of FBMs, addresses their overall performance (using some computing algorithms) and, provides a number of examples with varying degrees of complexity. However, it is important to underline the fact that this thesis does not define (or decide for) any particular technique to implement FBMs. It is not a thesis on, for example, Fuzzy Logic Control techniques. It only provides some **general guidelines** and **experimental results** on the use of these.

A **simple FBM** draws pictures about the potential behaviours of an agent (nodes) and their causal interactions (edges). This BB network can be designed using fuzzy numerical information (degrees of membership functions of fuzzy sets) and, fuzzy words (linguistic variables and fuzzy edges) to mean dual, intrinsic characterisation of fuzzy sets and fuzzy systems. **More complex FBMs** incorporate feedback loops. These FBMs are designed with recurrent topology networks that make them able to operate dynamically and, to converge towards possible asymptotic behaviours. Their behaviour nodes can evolve (over time of interaction) until some terminating condition has been reached.

The most **generic FBMs** allow extending and/or refining the number and type of BB components as to consider some possible external modules (relative to the activities, morphological set-up and, environmental conditions of the agent) that influence (and/or become causally affected) by the internal behaviour nodes.

The thesis provides a **proof-of-concept for simple FBMs**, including some experimental results in Mobile Robotics and Fuzzy Logic Control. This practical work shows the design of a **collision avoidance behaviour** (of a mobile robot) using a simple FBM and, the implementation of this using a **Fuzzy Logic Controller** (FLC). The FBM incorporates three causally related sensorimotor activities (moving around, perceiving obstacles and, varying speed). This **Collision Avoidance FBM** is designed (in more detail) using fuzzy relations (between levels of perception, motion and variation of speed) in the form of fuzzy control rules. The FLC stores and manipulates these fuzzy control (FBM) rules using fuzzy inference mechanisms and other related implementation parameters (fuzzy sets and fuzzy logic operators). The resulting FBM-FLC architecture controls the behaviour patterns of the agent. Its fuzzy inference mechanisms determine the level of activation of each FBM node while driving appropriate control actions over the creature's motors. The thesis validates (demonstrates the general fitness of) this control architecture through various **pilot tests** (computer simulations). This practical work also serves to emphasise some **benefits in the use of FLC techniques** to implement FBMs (e.g. flexibility of the fuzzy aggregation methods and fuzzy granularity).

More generally, the thesis presents and validates a **FBM Framework** to develop more complex autonomous agent behaviour. This framework represents a top-down approach to derive the BB models using generic FBMs, **levels of abstraction** and **refinement stages**. Its major scope is to capture and model behavioural dynamics at different levels of abstraction (through different levels of refinement). Most obviously, the framework maps some required behaviours into connection structures of behaviour-producing modules that are causally related. But the main idea is following as many refinement stages as required to complete the development process. These refinement stages help to identify lower design parameters (i.e. control actions) rather than linguistic variables, fuzzy sets or, fuzzy inference mechanisms. They facilitate the definition of the behaviours selected from first levels of abstraction. Further, the thesis proposes taking the FBM Framework into the implementation levels that are required to build BB control architecture and provides and **application case study**. This describes how to develop a complex, non-hierarchical, multi-agent behaviour system using the refinement capabilities of the FBM Framework. Finally, the thesis introduces some more general ideas about the use of this framework to cope with some, current complexity issues around the behaviour-oriented nature of intelligence.

Table of Contents

INTRODUCTION: AN INTRODUCTION TO THE DISSERTATION PRESENTING THE THESIS 1

Thesis Overview 1

A. Fuzzy Behavioural Maps 2

B. The Development of a FBM in Mobile Robotics 4

C. The Fuzzy Behavioural Maps Framework 8

Thesis Outline 10

Chapter 1: Research Into Behaviour-Oriented Views of Intelligence 11

Chapter 2: Fuzzy Behavioural Maps 13

Chapter 3: The Development of a FBM for a Mobile Robot 14

Chapter 4: The FBMs Framework 15

Chapter 5: Conclusions 15

Chapter 6: Future Research Work 15

CHAPTER 1: RESEARCH INTO BEHAVIOUR-ORIENTED VIEWS OF INTELLIGENCE 16

1.1. Behaviour-Based Artificial Intelligence 17

1.1.1. Points of Departure 18

1.1.1.1. Modelling and Building Behaviour-Based Forms of Intelligence 18

1.1.1.2. Not Using Conventional Symbolic-Processing Architectures 19

1.1.1.3. Avoiding World Modelling Techniques 21

1.1.2. Autonomous Robotics: The Beginning 22

1.1.2.1. The Pioneering Work of Rodney A. Brooks 22

1.1.2.2. Recent Generations of Mobile Robots 23

1.1.2.3. Some Explorations with Autonomous Robots 24

1.1.3. Major Insights 25

1.1.3.1. Objectives and Tasks; Models, Mechanisms and Systems 26

1.1.3.2. Slogan: Understanding Behaviour by Building Artificial Creatures 26

1.1.3.3. Claim: Biological Fidelity of the Approach 27

1.2. Autonomous Agents 27

1.2.1. What is An Autonomous Agent? 30

1.2.2. Understanding Behaviour-Oriented Views of Intelligence 31

1.2.2.1. Understanding Behaviours 31

1.2.2.2. Robots as Animals and Animals as Robots 32

1.2.2.3. Brain-Like Models 33

1.2.2.4. Genetic Side 35

1.2.2.5. Technological Perspectives 36

1.3. Implications and Research Issues 42

1.4. The Development of Behaviour-Based Systems 43

1.4.1. General Steps 43

1.4.1.1. Selecting and Describing Behaviours 44

1.4.1.2. Designing and Implementing the Behaviour-Producing Modules 45

1.4.1.3. Organising and Aggregating the Behaviour-Producing Modules 47

1.4.1.4. Hardware and Software Platforms 48

1.4.1. A Classification of Approaches 48

1.4.1.1. Subsumption Architectures 49

1.4.1.2. BB Networks 51

1.4.1.3. Dynamical BBSs 54

1.4.1.4. Evolutionary Architectures 56

1.4.1.5. Hybrid Paradigms 57

1.4.2. The Technical Issues 57

CHAPTER 2: FUZZY BEHAVIOURAL MAPS 61

2.1. Influences and Motivations 62

2.1.1. Fuzzy Cognitive Maps 62

2.1.2. Behaviour-Based Systems 69

2.2. Main Features 69

2.2.1. FBMs and FCMs 70

2.2.2. BB Aspects 75

2.3. Examples 79

2.3.1. Simple FBMs 79

2.3.2. More Complex FBMs 82

2.4. General Intuitions 88

CHAPTER 3: THE DEVELOPMENT OF A FBM FOR A MOBILE ROBOT 90

3.1. Mobile Robots and Control Systems 91

3.2. Perception, Motion and Collision Avoidance 93

3.3. The Collision Avoidance FBM 95

3.4. The FLC Architecture 98

3.4.1. Structure and Operations 100

3.4.2. Components and Implementation Parameters 101

3.4.2.1. Component 1: Fuzzification Interface 102

3.4.2.2. Component 2: Data Base 103

3.4.2.3. Component 3: Rule Base 105

3.4.2.4. Component 4: Decision Making Logic Unit 108

3.4.2.5. Component 5: Defuzzification Interface 112

3.5. The FLC-Based Implementation of the Collision Avoidance FBM 114

3.5.1. Fuzzification Interface 116

3.5.2. Data Base 117

3.5.3. Rule Base 119

3.5.3.1. Sub-Rule Base R1: Levels of Perception 119

3.5.3.2. Sub-Rule Base R2: Avoidance Control Policy 122

3.5.4. Decision Making Logic Unit 125

3.5.4.1. The Fuzzy Inference Mechanism 125

3.5.4.2. Weighting Factors and Fuzzy Inference 128

3.5.5. Defuzzification Interface 130

3.6. Pilot Tests and Results 132

3.6.1. Khepera Simulator 132

3.6.2. The FBM-FLC Program 133

3.6.2.1. Fuzzy Logic Source Files 134

3.6.2.2. User Source Files 138

3.6.2.3. Makefile 142

3.6.3. Computer Simulations 144

3.6.4. Experimental Results 149

CHAPTER 4: THE FBMs FRAMEWORK 157

4.1. Generic FBMs 158

4.2. The FBMs Framework 165

4.2.1. Levels of Abstraction of FBMs 166

4.2.2. The Refinement of FBMs 167

4.3. An Application Case Study 169

4.4. Complexity Considerations 173

4.4.1. Abstraction and Description of Behaviours 174

4.4.2. Organisation, Activity and Execution of Behaviours 177

4.4.3. Evaluation and Optimisations 179

4.4.4. Technical Aspects to Design and Implement the Systems 180

CHAPTER 5: CONCLUSIONS 181

The Investigation Into the Behaviour-Oriented Views of Intelligence 181

The Definition, Exploration and Evaluation of FBMs 182

The Collision Avoidance FBM: Development, Pilot Tests and Results 183

The FBMs Framework: Definition and Application Case Study 185

CHAPTER 6: FUTURE RESEARCH WORK 187

APPENDIX A: FUZZY SETS AND FUZZY RELATIONS 190

APPENDIX B: FUZZY LOGIC CONTROL 199

APPENDIX C: THE FBM-FLC PROGRAM 204

BIBLIOGRAPHY 300

List of Figures

Figure I.1: Thesis Overview. Investigations and Contributions to Knowledge.

Figure I.2: Thesis Overview. Fuzzy Behavioural Maps.

Figure I.3: Thesis Overview. The Development of a FBM.

Figure I.4: Thesis Overview. The FBM Framework.

Figure 2.1: Labels of edges in FCMs.

Figure 2.2: An Example of a FCM found in [Kosko-86a].

Figure 2.3: Example of a FCM; Edge Matrix and Nodes String [Kosko-94].

Figure 2.4: Example of an Augmented FCM [Taber-91].

Figure 2.5: Simple FBM (1).

Figure 2.6: Simple FBM (2).

Figure 2.7: More Complex FBMs. Predator & Food.

Figure 2.8: Turn Movements with FBMs.

Figure 2.9: FBMs with Co-operation (edge > 0) and Competition (edge < 0) of Behaviours.

Figure 2.10: Co-operation among FBMs.

Figure 2.11: A Hierarchy of Behaviours inspired by [Tyrrell-92].

Figure 3.1: A Top View of the Mobile Robot Khepera.

Figure 3.2: The Collision Avoidance FBM.

Figure 3.3: Perception and Motion.

Figure 3.4: Basic Structure of a FLC.

Figure 3.5: The Khepera Simulator, the Collision Avoidance FBM and the FLC.

Figure 3.6: Membership Functions of Primary Fuzzy Sets of the FBM-FLC.

Figure 3.7: Flow Diagram of Input Data to the Inference Mechanism of the FBM-FLC.

Figure 3.8: Defuzzification Strategy of the FBM-FLC.

Figure 3.9: Running Khepera, Simulated Environment and Initialised I/O Fuzzy Data of the FBM-FLC.

Figure 3.10: Running Khepera in an empty world (1).

Figure 3.11: Running Khepera in an empty world (2).

Figure 3.12: Running Khepera in an empty world (3).

Figure 3.13: Running Khepera in an empty world (4).

Figure 3.14: Running Khepera in an empty world (5).

Figure 3.15: Firing Strengths of the FLC with Mamdani's and Larsen's Fuzzy Implications.

Figure 3.16: Paths of the Khepera robot in corridor A for the Mamdani's and Larsen's fuzzy inference .techniques.

Figure 3.17: Paths of the Khepera robot in corridor B for the Mamdani's and Larsen's fuzzy inference technique.

Figure 3.18: Path of Khepera robot passing through the T-corridor

Figure 3.19: Path of Khepera robot negotiating the T-corridor (1).

Figure 3.20: Path of Khepera robot negotiating the T-corridor (2).

Figure 4.1: "Generic FBM": External Effectors, Internal Flow and Consequences of Behaviours.

Figure 4.2: A Top-View Model of Interaction of a FBM.

Figure 4.3: A Low-View Model of Interaction of a FBM.

Figure 4.4: FBM Framework. Levels of Abstraction and Refinement.

Figure 4.5: First Level of Abstraction.

Figure 4.6: Second Level of Abstraction.

Figure 4.7: Third Level of Abstraction.

Figure A.1: Diagrammatic Representation of Functional fuzzy sets.

List of Tables

Table 3.1. Table of Components and Implementation Parameters of a FLC.

Table 3.2. Prototype of R1. Levels of Perception.

Table 3.3. Prototype of R2. Collision Avoidance Control Policy.

Table 3.4. Pilot Tests in Corridor B.

Table A.1. Membership Functions of (a) Triangular Norms, (b) Triangular Co-Norms.

Table A.2. Fuzzy Implications.

Acronyms

AA	Autonomous Agent
ANN	Artificial Neural Network
BB AI	Behaviour-Based Artificial Intelligence
BBS	Behaviour-Based System
FANN	Feedforward ANN
FBM	Fuzzy Behavioural Map
FL	Fuzzy Logic
FLC	Fuzzy Logic Controller
FS	Fuzzy System
KB AI	Knowledge-Based AI
ES	Expert System
RANN	Recurrent ANN
SA	Subsumption Architecture
SMPAS	Sense Model Plan Act System
UL	Unsupervised Learning

Introduction

An Introduction to the Dissertation Presenting the Thesis

Thesis Overview

This dissertation presents a thesis on the definition and use of *Fuzzy Behavioural Maps* (FBMs) within the context of the modelling and building of *behaviour-oriented forms of intelligence* with *Autonomous Agents*. More particularly, this document provides the following main contributions to knowledge (see Figure I.1):

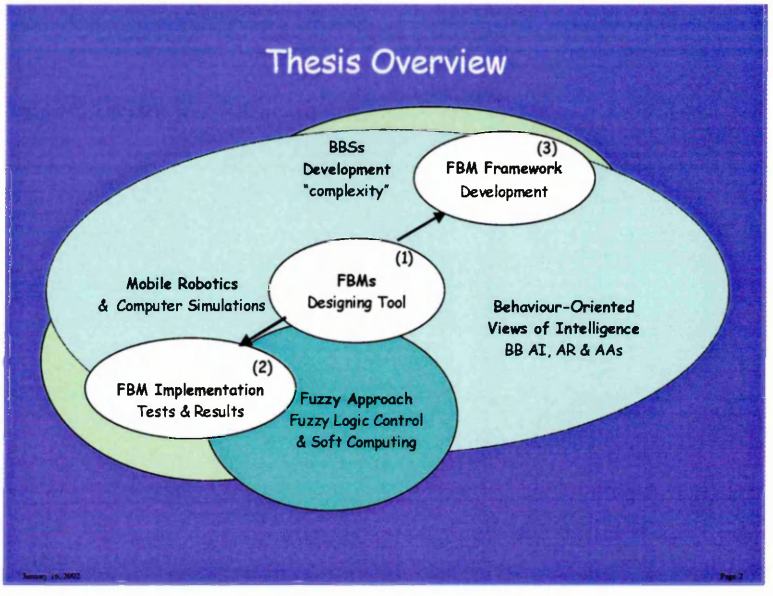


Figure I.1: Thesis Overview. Investigations and Contributions to Knowledge.

1. The presentation of FBMs as *a new approach to design behaviour-based models.*

2. The *development of a FBM in Mobile Robotics* i.e. the design, implementation and testing of a Collision Avoidance FBM for a simulated mobile robot.

3. The extension of all this work into the *definition and use of a FBM Framework* based on some features of the current developments of *Behaviour-Based Systems* and related complexity research issues.

A. Fuzzy Behavioural Maps

A FBM is a network with a graph topology where the nodes represent *behaviours* and the edges represent their *causal interactions*. It is a concept fundamentally motivated by:

- Fuzzy Cognitive Maps [Kosko-86a; Kosko-86c; Kosko-87; Kandel-87; Kosko-88; Taber-91; Kosko-94],
- other Cognitive Mapping Techniques [Tolman-84; Sholl-87; Levenick-91; Branback & Malaske-95],
- Non-Hierarchical Behaviour Decompositions [Brooks-86; Brooks-91a; Brooks-91b; Mataric-94a; Mataric-94b],
- Behaviour-Based Networks [Maes-89b; Tyrrell-92] and,
- Dynamical Behaviour-Based Systems [Smithers-94; Smithers-95; Steels-94b; Steels-95])

The overall performance of a FBM results from driving *causal interactions* (cause-effect flows) among the levels of activation of the behaviours that it represents. Further, it is possible being flexible in the way that the *design* of this map is addressed and, the way that the terms *behaviour* and *cause-effect interaction* are interpreted using this.

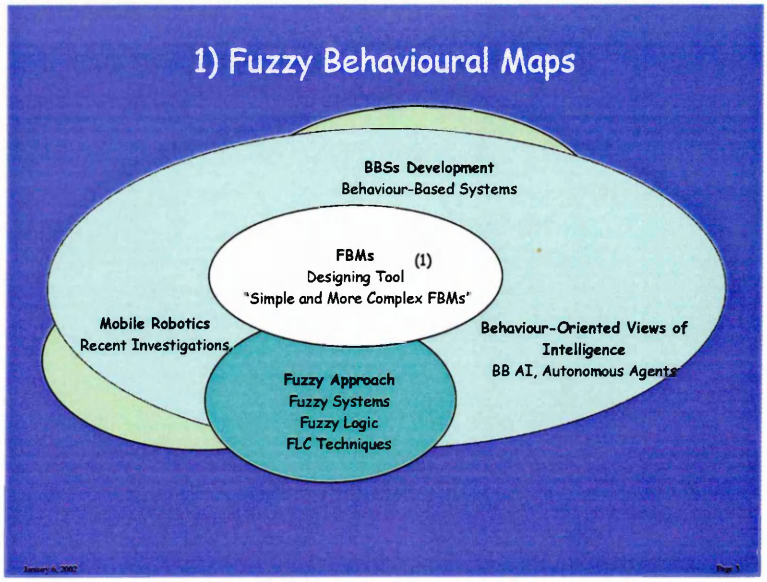


Figure I.2: Thesis Overview. Fuzzy Behavioural Maps.

A simple FBM is a directed graph where the nodes represent *behaviour-producing modules* the designer is hoping to implement within an Autonomous Agent. It is a network that draws pictures about the behaviours of the agent and how the firing degrees of these can be causally related with *symbolic edges* showing either positive or negative cause-effect relation. These cause-effect connections provide a view of interaction between the behaviour-oriented nodes of the map. But the idea is that both the levels of activation of the behaviour nodes and the inter-connections among these should be expressed in terms of Fuzzy Logic using, for example, *linguistic variables* and *fuzzy sets* or, *fuzzy inference mechanisms* to determine the level of activation of a particular node of a FBM.

Complex FBMs incorporate recurrent feedback loops. These maps are dynamic systems that can be *switch on* with *behaviour-activation patterns* being propagated until some terminating condition (e.g., timing out or, maximum level of activation of behaviour/s) has been reached. Further, we can achieve behaviour co-operation (or, behaviour competition) by means of combining complex FBMs. It is possible aggregating a number of low-level FBMs so that these maps modify the levels of activity of each other.

With these definitions, this dissertation argues that FBMs are *flexible designing tools* in the sense that their use underlines *several interpretations and uses*. Indeed, one of the major statements of this thesis is that we can use FBMs with:

- nodes as *physical resources* of an agent (e.g., infrared sensors of a mobile robot) or, other more *abstract behaviour views* (e.g., hunger awareness of an animal) and,
- edges representing either *physical communication links* or, other more *abstract interactions* such as *co-operation* and (or) *competition* of behaviour-producing modules.

B. The Development of a FBM in Mobile Robotics

In recent years, there has been a resurgence of interest in the field of Robotics inspired, largely, by the Behaviour-Based (BB) viewpoint [Brooks-91a; Brooks-91b; Maes-94a; Maes-94b; Mataric-94a] that this thesis explores. From the research carried out so far (overviewed in chapter 1), it is clear that, in order to produce effective BB Systems, mobile robots will need to acquire and develop *their own knowledge* in order to adapt to the demands of a complex and changing environment. A number of techniques have been proposed and explored to address this issue (explained, for example, in [Brooks-86; Cliff *et al*-93a; Smithers-95; Steels-94a; Mataric-94a; Maes-89b; Rosenblant & Payton-89]).

Whilst successful, this approach to BB Robotics (also called *Autonomous Robotics*) becomes unwieldy as the number and the complexity of the behaviours and interactions involved increase. This leads to difficulties in the *practical design of BB Systems for mobile robots. Artificial Neural Networks* have been proposed as providing a mechanism to address this problem and some applications of these control systems do appear to be able to implement basic BB architectures.

However, the design of *composed behavioural networks* (BB Networks) also become more and more complex with the desired high level groups of behaviours to be exhibited by a mobile robot (see, for example, [Smithers-94]).

The difficulties in applying this technology are similar to those encountered in the development of *complex, large-scale BB Systems*. For example, as with the *Subsumption Architecture* [Brooks-86], the key problem in applying Feedforward Artificial Neural Networks is the need to implicitly design the control process in terms of network units and inflexible inter-unit connections (as described in, for example, [Takagi *et al*-92]). In addition other authors argue that, however detailed the inter-unit relations are, and however large the number of processors (neurones) may be, there are always unpredictable situations which complicate the design process by presenting circumstances which are difficult to foresee [Smithers-94].

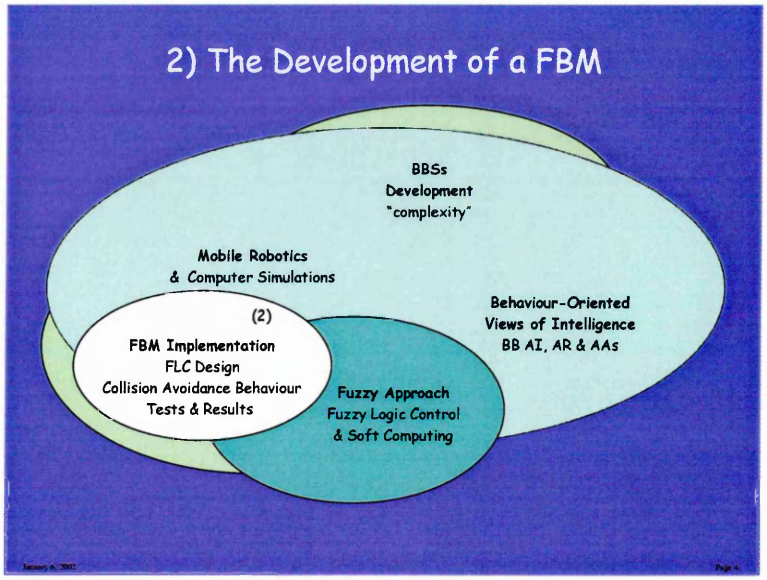


Figure I.3: Thesis Overview. The Development of a FBM.

The thesis presents the FBMs as simple designing tools *to try to handle this complexity more effectively*, exploiting ideas from *Fuzzy Systems Theories*, *Fuzzy Logic Control* (FLC) techniques and, some *current approaches to develop BB Systems.*

In this dissertation, we summarises the results of our initial investigations with FBMs. This work to date concentrates on the design and control of a mobile robot's behaviour using simple FBMs and FLC implementation techniques (see Figure I.3). We provide a *proof-of-concept of simple FBMs* through the design, implementation and testing of a FBM for a mobile robot to exhibit *collision avoidance behaviour.*

More particularly, this part of the thesis brings:

- the *study* of Fuzzy Systems and FLC techniques (providing a generic FLC architecture) to control sensorimotor behaviours of a mobile robot,
- the *design* of a collision avoidance behaviour-producing module using a FBM (a Collision Avoidance FBM) and some preliminary analysis of the avoidance strategy,
- the *implementation* process of an FLC architecture,
- the *mapping* of the Collision Avoidance FBM onto the FLC architecture to control the final activation of the avoidance behaviour (FBM global node) using fuzzy inference mechanisms and,
- *some pilot tests of the resulting FBM-based architecture* using different types of fuzzy inference mechanisms (to control the activation of the avoidance behaviour) and the corresponding analysis of the behaviour displayed by the mobile robot.

In general, the use of *Fuzzy Systems* (FSs) has been advocated because of their abilities to handle uncertain information using fuzzy sets, fuzzy weightings and other operations related to the aggregation and comparison of fuzzy sets of objects [Zadeh-65; Bellman & Zadeh-70; Kandel-87; Zadeh & Kacprzyk-92;

Foulloy-93; Safiotti-97; Gorrini & Bersini-94; McNeill & Thro-94]. However, our main reason for believing that these systems will be effective for developing BB Systems is due to a specific feature of fuzzy logic processing, namely that it provides a changeable output rather than the (straightforward) weighted sum of other controllers such as those based on ANNs.

Even though this functionality appears to be suitable for designing BB Systems, FSs do not seem to have received much attention in this research area. The only examples found in the literature refer to the development of *Hybrid* or, *Fuzzy Neural Nets* [Narazaki & Ralescu-92; Keller *et al*-92; Goode & Chow-94; Goonatilake & Khebbal-95; Hercock & Barnes-96; Pipe & Winfield-96a; Pipe *et al*-96b; Safiotti-97] that incorporate special network propagation algorithms with computations driven by Fuzzy Logic inference. Though, from our point of view, these examples do not involve direct applications of Fuzzy Logic such as the FLCs based on input-output fuzzy relations and fuzzy inference mechanisms [Mamdani-76; Wang & Loe-93].

The analysis of the collision avoidance strategy discussed in this dissertation begins with a simple linguistic algorithm similar to one used by [Braitenberg-84]. Then we improve this strategy designing a Collision Avoidance FBM (that exploits some features of the fuzzy approach) and implement it using a FLC architecture that includes five functional components (Fuzzification Interface, Data Base, Rule Base, Decision Making Logic Unit and Defuzzification Interface).

Basically, the implementation of the Collision Avoidance FBM using the FLC architecture consist of converting the nodes and edges of the map onto fuzzy logic control rules. We have implemented this heuristical verbalisation of the Collision Avoidance FBM within the FLC Rule Base so that the fuzzy inference mechanisms (driven by the Decision Making Logic Unit) activate the avoidance behaviour of the mobile robot. In doing so, we demonstrate that simple FBMs can be implemented using Fuzzy Logic parameters such as fuzzy sets, linguistic variables and fuzzy implication methods.

Further, the results of the pilot tests presented here demonstrate that the resulting FBM-based architecture combines the designing features of the proposed FBMs with some important capabilities of the fuzzy approach. These pilot tests show that in simulation and, using Mamdani and Larsen's fuzzy inference techniques [Mamdani-76; Wang & Loe-93], the mobile robot follows different patterns of collision avoidance behaviour. It displays one behaviour pattern or another depending on the fuzzy inference technique that is used (Mamdami or, Larsen's).

C. The Fuzzy Behavioural Maps Framework

The most generic type of FBM that this thesis introduces refers to *FBM-based architectures*. This dissertation demonstrates that it is possible to extend and/or refine the number and type of components of the behaviour producing modules of an FBM so that some of these sub-modules become causally affected by the changes of others that might include (but are not limited to) *environmental stimuli*, *internal components / mechanisms of the AA* or, external *resources*.

A *generic FBM* can incorporate as many nodes (sub-nodes) and interactions (sub-links) as the designer needs in order to design a set of behaviour producing modules and their interactions. The FBM-based architectures incorporate different types of behaviour-oriented modules. They include both internal and external modules interacting with each other. Some external modules or, their changes (if identified) can directly cause the activation of an FBM (and so, engage the agent to perform the behaviour it represents) or, increase / decrease the levels of activation of some of its internal behaviour nodes (e.g., make the agent to start performing some other actions). Similarly, the level of activation of the external nodes can be increased / decreased by the effect of the internal ones. Indeed, the flow of an FBM can include and drive many feedback connections (and related computations), to allow changes on the components that influence this map, as to implement many possible consequences of behaviours.

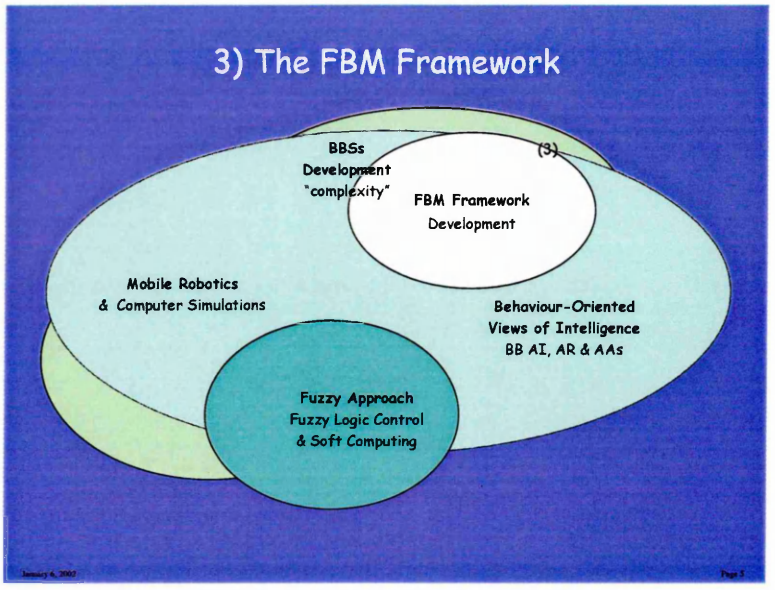


Figure I.4: Thesis Overview. (3) The FBM Framework.

In essence, this dissertation demonstrates that it is possible using many different types of FBMs to design behaviour-based models and systems. The nodes, edges and external components (or mechanisms) of an FBM-based architecture let the designer establishing a number of behaviour-oriented processes and implementing different sort of BB Systems. A *more generic FBM* extends a basic one incorporating new FBM modules (nodes and edges). The final set up (design) of the map depends on many design aspects such as the number and complexity of the behaviour producing modules that are included, how these behaviours have been analysed, the morphological set up of the agent(s), the behavioural performance that has to be achieved, etc. Using these generic FBMs, the dissertation describes *how to derive a non-hierarchical Action Selection Mechanism.*

More generally, this dissertation proposes the FBM approach as a tool for trying to address some *technical complexity issues* that the thesis identifies in the current approaches to develop BB Systems. This document presents a FBM Framework exploring and evaluating the development of complex behaviour-oriented structures using FBMs. It discusses how to develop large-scale BB Systems using FBMs.

The keywords explored here are: *levels of abstraction* and *refinement*. These extend the mapping of FBMs onto fuzzy architectures considering that it is possible designing FBMs at different levels of abstraction and, refining these by identifying lower behaviour-oriented design parameters rather than linguistic variables, fuzzy sets or, fuzzy inference mechanisms.

The thesis presents the *FBM Framework* in comparison to other current techniques to develop large-scale behaviour-oriented models. It describes the key initial features of this framework to assess the advantages of this technique. Finally, the dissertation presents an example about how to design a large-scale BB System using the FBM Framework. This work is largely inspired by the "Interaction and Intelligent Behaviour" presented in [Mataric-92b] and represents our initial work towards the development of more complex and more natural behaviour systems.

Dissertation Outline

Before moving into the presentation of these contributions to knowledge around the definition and use of FBMs, this dissertation surveys our investigations into the behaviour-oriented nature of intelligence concentrating on the current approaches to develop BB Systems and research issues that have motivated the thesis. In doing so, we pursue four main objectives:

1. Presenting an overall view of what has been achieved and what remains to be done to prove the validity of the behaviour-oriented viewpoint in Artificial Intelligence (AI).

2. Elaborating the fundamentals of this field by identifying different contributions (e.g. ideas, solutions, theories, etc.) made by research groups that, falling into cognitive, technical and biological tendencies, study either natural or artificial creatures and establish some scientific basis to the behaviour-based models.

3. Underlying the role played by Computer Science providing useful formal frameworks (e.g., fuzzy frameworks, neural nets and soft computing paradigms) together with other more general technological perspectives used to build models through the development of artificial systems (i.e., moving from the selected forms of intelligence to the physical or, computational systems).

4. Exploring and evaluating the potential of the proposed FBMs to design behaviour-based models.

Chapter 1: Research Into Behaviour-Oriented Views of Intelligence

Chapter 1 presents our overall view of the behaviour-oriented nature of intelligence from the foundations of three recent research areas (represented in Figures I.1 to I.4):

- a relatively new *Behaviour-Based Approach to Artificial Intelligence* (BB AI, founded around 1985) that firstly focused on behaviour-oriented forms of intelligence considering these were "more realistic" than using earlier knowledge-based ones,

- a school called *Autonomous Robotics*, mainly dedicated to design and build behaviour-based robot control systems and,

- a multi-disciplinary group of research communities known as *Autonomous Agents*, offering a great diversity of ideas and methods to study, model and build different sorts of creatures (not only robots) able to act autonomously.

The overview of the BB AI fundamentals (concepts, ideas, techniques and, methods) presented in this dissertation includes three subsections. The first one (1.1.1) shows definitive points of departure that express the shift made from an early Knowledge-Based approach to AI (KB AI also called "the conventional approach" or, "good-old-fashioned AI") to BB AI. These are:

- modelling and building behaviour-based forms of intelligence,
- not using conventional symbolic-processing architectures and,
- avoiding world modelling techniques.

The next subsection (1.1.2) focuses on the beginning of BB AI i.e. the pioneering work of Rodney A. Brooks and the influence of this within the classical field of Robotics. This includes some explorations within the Autonomous Robotics school (building behaviour-based control systems for recent generations of "robots") that help to identify some of the basis of the practical work within BB AI. Finally, the subsection 1.1.3 summarises the most general insights about the behaviour-oriented viewpoint ("objectives and tasks", "slogan" and "claim").

The work within AAs (described in section 1.2) offers a broader view on the behaviour-oriented nature of intelligence. This extends the BB AI fundamentals into a multi-disciplinary groundwork and identifies relevant contributions made by research groups that fall into cognitive, biological and technological perspectives. The subsection 1.2.1 briefly introduces the concept of AA and examines some of its forms. The next subsection (1.2.2) discusses how to understand behaviour-oriented views of intelligence with AAs. This presents an overview of the state-of-art of Autonomous Agents describing the following major tendencies that have motivated the thesis presented in this dissertation:

- understanding behaviours (1.2.2.1),
- robots as animals and animals as robots (1.2.2.2),
- brain-like models (1.2.2.3),
- genetic side (1.2.2.4) and,
- technological perspectives (1.2.2.5).

All these studies include a variety of ideas (e.g., design perspective, mechanisms) that influence and/or motivate the behaviour-based models being studied and used in this dissertation. For example, the technological perspectives discussed in 1.2.2.5 gathers some design aspects about two successful Computational Techniques (Fuzzy Systems and Artificial Neural Networks) while making emphasis on the contributions of the Fuzzy approach.

The next section (1.3) argues for some implications and active research issues. This focuses on the currently required higher levels (large-scale models) with which the thesis starts proving some potential benefits of the FBM approach. Further, this subsection discusses how the most recent work within BB AI and AAs points to the realisation of more realistic artificial autonomous creatures that in turn represents a very complex problem (that cannot be easily resolved). Section 1.4 completes the survey (the research context of the thesis) by concentrating on the development of BB Systems (principal tools of the investigations). The introduction of this section outlines some key aspects about the development of these systems. The subsection 1.4.1 provides the most general steps we have found in the literature to develop BB Systems. The subsection 1.4.2 presents a classification of approaches to the development of these artificial systems (Subsumption Architectures, BB Networks, Dynamical BB Systems, Evolutionary Approaches and Hybrid Paradigms). Finally, the subsection 1.4.3 discusses some technical problems found with the application of these various approaches i.e. the major motivations of the FBM Framework presented in the thesis.

Chapter 2: Fuzzy Behavioural Maps

In chapters 2 to 5, this dissertation presents the work to date exploring and evaluating the role of the FBMs as designing tools of behaviour-based models.

Chapter 2 introduces the main features of FBMs. Section 2.1 describes influences and motivations. The main features of FBMs are described in section 2.2. The next section (2.3) provides examples of simple and more

complex FBMs. The most general intuitions we have about the definition and use of FBMs are summarised in section 2.4.

Chapter 3: The Development of a FBM for a Mobile Robot

The proof-of-concept of FBMs and initial investigations around the use of the Fuzzy Approach to implement these maps is widely described in chapter 3.

Section 3.1 introduces some general features about mobile robots and our initial evaluations of FLC implementation techniques. Section 3.2 summarises the analysis of the avoidance behaviour i.e., our preliminary studies on perception, motion, avoidance and, the use of Fuzzy Logic parameters. In the Appendix A and, for the convenience of the reader, we summarise Fuzzy Sets and Fuzzy Relations definitions used in the rest of the chapter. Similarly, in appendix B, we provide some general features of Fuzzy Logic Control techniques.

The Collision Avoidance FBM (and how to map this graph onto our FLC architecture) is shown in sections 3.3 to 3.5. Section 3.3 provides the design of the FBM. In section 3.4, we identify all the main features, functional components and implementation parameters that are required for implementing a FLC architecture. Section 3.5 shows the FLC-based implementation of the FBM for the mobile robot called Khepera. Finally, section 3.6 describes our computer simulations (3.6.3), including a complete description of the FBM-FLC control program that we have developed and built within Khepera Simulator system (3.6.2). Further, section 3.6 describes the results of our experiments (testing the general fitness of this architecture= and compares the performance of the avoidance behaviour displayed by the mobile robot (variation in the trajectories followed by the robot) when using different types of fuzzy inference techniques (Mamdani and Larsen) within the inference mechanism of the FLC.

Chapter 4: The FBMs Framework

Chapter 4 extends our explorations with FBMs and defines a development framework based on the use of these behaviour-oriented maps. Section 4.1 describes generic FBMs. The next section of this chapter (4.2) outlines the proposed FBM Framework in terms of levels of abstraction, refinement and the most general ideas explored in the thesis to cope with complex BB Systems developments. The next section (4.3) outlines an application case study base on the use of this FBM Framework. Finally, section 4.4 presents an application case study of the proposed FBM Framework based on the development of a complex, multi-agent BB System.

Chapter 5: Conclusions

Chapter 5 summarises our conclusions i.e. the statements of the thesis and contributions to knowledge around the definition and evaluation of FBMs.

Chapter 6: Future Research Work

Finally, chapter 6 describes future research objectives around more definitions and uses of FBMs as possible extensions of the thesis.

Chapter 1

Research Into Behaviour-Oriented Views of Intelligence

Abstract

This chapter surveys the following investigations as the research context of the thesis:

a) The description of BB AI fundamentals to study and use behaviour-based models instead of the early knowledge-based ones (section 1.1).

b) The specification of a broader view on the behaviour-oriented nature of intelligence based on a multi-disciplinary groundwork known as "Autonomous Agents" and, concentrating on cognitive, biological and technological perspectives (section 1.2) that we have been considered to present the FBMs.

c) The identification of active research issues around the currently required higher levels of intelligence (section 1.3) with which the dissertation starts proving the potential benefits of the FBM Framework.

d) The presentation of a classification of approaches to the development of BB Systems (Subsumption Architectures, Behaviour-Based Networks, Dynamical Behaviour-Based Systems, Evolutionary Approaches and Hybrid Paradigms described in section 1.4) based on some development decisions raised with the proposed FBM Framework.

1.1. Behaviour-Based Artificial Intelligence

The goal of AI researchers has always been *the modelling and building of forms of intelligence* [Newell-82; Wilson-85; McEachern-93]. They have done much work to study intelligence by modelling human activities (e.g., reasoning, communication, learning) and applying these models to construct artificial intelligent systems (creatures, robot systems or, computer programs). Since 1986, however, "a general belief is that a new life has emerged in the study of AI" [Brooks-91b]; "a new life is being brought to the field..." [Maes-92b] (see also [Brooks-90b; Brooks-90c; Brooks-91a; Brooks-91e; Resnick-92; Brooks-94; Wilson-91; Mataric-91b; Mataric-94a; Pfeifer-96b]).

The most recent communities of AI investigate an approach called *Behaviour-Based AI* (BB AI) that has challenged conventional *Knowledge-Based AI* (KB AI) [Maes-92b; Pfeifer & Verschure-95; Saunders *et al.*-94; Wilson-91]. Their main objective is studying intelligence by introducing and using non-conventional, behaviour-oriented models and techniques. They investigate new types of problems, adopt new techniques and search for new solutions to design and implement the artificial intelligent systems.

More particularly, the research of these recent communities of AI reflects into the field of *Robotics*, a benchmark area of AI for longer than thirty years. Their practical work is being devoted, mainly, to the investigation of new robots (artificial creatures with physical bodies) able to perform some activities within their natural surroundings. Further, their investigations lead to the institution of a new school known as *Autonomous Robotics* that concentrates on the realisation of *autonomous robots* (robots that can perform some actions without much human intervention) while making strong connections between the biological inspiration [Brooks-91e; Brooks-91f; Steels-94a; Brooks-91g] and, the use of the physical basis of *Artificial Neural Networks* [Grossberg-78; Grossberg-80; Bose & Liang-96; Kosko-92; Schmajuk-94; Bezdek-92; Mariano & Morasso-94] and other *Artificial Life* techniques [Langston-90; Beer *et al.*-90; Brooks-91f; Beer-90; Moran *et al.*-95] to develop robot systems.

All these aspects of BB AI are discussed below in more detail.

1.1.1. Points of Departure

There are three significant points of departure from the early KB AI to the current BB AI.

1.1.1.1. Modelling and Building Behaviour-Based Forms of Intelligence

In 1986, the AI paradigm shifted from the study and replication of high level and knowledge-oriented forms of intelligence [Drescher-91; McEachern-93] to the analysis and synthesis of lower level and behaviour-oriented ones [Brooks-91a; Brooks-91d; Wilson-91; McFarland & Bosser-94; McFarland-95; Maes-92b; Mataric-92b]). Indeed, the literature reflects that, while the researchers of the "good-old-fashioned" KB AI focused on *knowledge-based models*, the recent communities of BB AI study *behaviour-based models*.

The researchers of the early KB AI associated the concept *intelligence* to the term *knowledge* [Newell-82]. Their work focused on the study of forms of intelligence such as reasoning or, problem solving [McEachern-93; Hendler-88], motivated by the fact that computers could mimic human thoughts [Minsky-86; Grossberg-78; Grossberg-82; Grossberg-86; Bond & Gasser-88]. They built artificial intelligent systems by identifying, formalising and representing some specific knowledge [Minsky-86]. They had many successful applications in areas where the level of machine intelligence could be measured in terms of the amount and significance of knowledge that the systems were able to manipulate (e.g. Game Playing or Reasoning,).

The most recent communities of AI use a concept of intelligence that is strongly related to the term "natural behaviour" (behaviour displayed by living organisms) [Brooks-91d; Brooks-91c]. Their work represents a general move towards the study of behaviour-based forms of intelligence such as *adaptation, learning* or, *interaction dynamics*, mainly, because of two general beliefs. First, the KB concept of intelligence is "ill-defined" in the sense that it does permit "viable machine intelligence with clear biological inspiration"

[Brooks-91a]. Second, KB models and systems are not realistic enough to advocate AI (they can only work for simulated toy problems [Maes-92a; Maes-92b]).

Nowadays, humans and other natural beings (animals) can be said to be intelligent not because they can store and process some knowledge but, because they can intelligently respond to their environments [Brooks-91b; Maes-92b; Pfeifer-96b]. They can determine their own representations and investigate new (appropriate) ones [Pfeifer & Verschure-95]. They are observed to adapt their actions to the changes of their environments and learn from this [Brooks-93; Aitken-94; Barto-90; Barto *et al.*-95; Bryson-96; Maes & Brooks-90c; Mataric-90a; Mataric-90b; Maes-92a; Mataric-94b]. "Intelligence is determined by dynamics of interaction with the world" [Brooks-91b]. It is "an emergent property of the interaction of the physically embodied agent with the real world" [Brooks-91a].

Consequently, the recent communities of AI introduce and use new technical solutions to construct behaviour-oriented systems. They use, for example, "bottom-up" development methods (that substitute the classical "top-down" methodologies) [Brooks-91b; Brooks-91f; Brooks-91g] or, "non-centralised" control mechanisms (that clearly differ from conventional mechanisms which structure and functioning depends on a central engine) [Maes-92a; Maes-92b; Brooks-91a]. Other, more general, points of departure from KB to BB AI are described below (see also [Brooks-86; Brooks-94; Mataric-95a; Smithers-95; Pfeifer-96b]).

1.1.1.2. Not Using Conventional Symbolic-Processing Architectures

The classical knowledge-based models were built using *conventional symbolic-processing architectures* such as Sense-Model-Plan-Act (SMPA). These architectures, however, do not seem to be appropriate for building the behaviour-based models.

Building a KB model using a SMPA architecture means constructing a system is such a way that, when it is placed in a determined situation (well-known environment) and, it is equipped with appropriate symbolic representations (goals and copies of the world), it can recognise certain aspects of its surroundings (sensing)

and plan actions for achieving its goals (thinking) before performing these (acting). It is a matter of designing and implementing a system based on three specific-purpose units: a *sense unit* (that collects the symbols about the objective features of the environment), a *planner unit* (or central engine that delivers a *plan* according to the input coming from the sense unit and an internal copy of the environment) and, an *act unit* (that performs the set of actions delivered by the planner). Further, the effort is concentrated on the logical operations of the planner unit. In fact, the system can only perform the required functionality if its planner unit "knows" how to correlate the senor unit's information with its internal, engineered copy of the outside world (see, for example, [Newell-82]).

By contrast, building a BB model means constructing a system in such a way that it can adapt its actions to (or learn from) the real world situations that it encounters (as described, for example, in [Maes-92b; Mataric-92b]). It requires solving issues such as how to design *behaviour-producing modules* (in a non-centralised and self-contained manner) and make these working within the real world situations, not how to symbolically represent and process some symbols to respond to a given (well-known) problem domain [Steels-93a; Steels-94a]. The BB architectures are not intended to plan some actions according to an engineered environment [Maes-92b; Peng & Williams-92; Pfeifer & Verschure-95; Chapman-92; Mataric-91b; Mataric-92a; Smithers-94; Pfeifer-94]. They have to be designed using *open architectures* because the ability of the system to adapt and/or learn depends on its capabilities to dynamically interact with the real world [Brooks-91a; Brooks-91b; Pfeifer-94; Smithers-92; Resnick-92; Smithers-94]. They have to relate mutually to their natural (non-engineered) domains [Maes-92b]. They have to solve many problems at a time [Brooks-91b; Chapman-92; Beer-95]. Their designers need to worry about the "environmental pressures" [Steels-94a].

These and other related behaviour-oriented architectures are discussed in sections 1.1.2 and 1.4.

1.1.1.3. Avoiding World Modelling Techniques

The conventional *world modelling techniques* are not appropriate for building behaviour-based models. These techniques are unrealistic and limit the well performance of the systems [Mataric-91b; Maes-92b; Mataric-92b; Brooks-91b; Pfeifer-94; Steels-93a; Steels-94a].

A *world model* is unrealistic because it is far remote from the real one; because it can hardly reflect the objective reality; because the physical world is too complex to admit of a complete engineered version. Making a complete world model is hard to believe and, very difficult to foresee (as discussed, for example, in [Mataric-91b; Brooks-90b]). We can only achieve *partial models* that suffer from the *uncertainty* and limit the well performance of the architecture since they are time-consuming. Indeed, the time that is required for a system (or component of a system) to correlate some current input values (like sensor readings of a robot) to a large (even moderate) world model can delay the computation of the final outputs (actions) until a point that makes the creature failing its real time performance [Mataric-92b; Mataric-95a].

[Brooks-91e] proposes an alternative solution to world modelling techniques. This consists of trying to "find out what the creature is sensing and how its sensory information can be organised/manipulated so that it accomplishes its tasks successfully". It is not clear, however, whether this work can effectively substitute the use of world models. As we shall see in sections 1.1.2 and 1.4, it is not difficult making the creature to display basic behaviours within some not-modelled situations but, the physical world is too complex and, it is very difficult making the behaviour-oriented system to successfully respond to very unpredictable situations. In section 1.2.2.5, we describe these issues is more detail and discuss how the Fuzzy approach can help to solve the uncertainty problem.

1.1.2. Autonomous Robotics: The Beginning

The project of constructing mobile robots has a history of several centuries. Previously, control systems for such devices varied from simple stimulus/response architectures to more complex ones based upon the Physical Symbol System or, the "sense-think-act" model above described. In 1986, Rodney A. Brooks criticised those traditional approaches considering that they were "unable to deliver real-time control in dynamic worlds".

1.1.2.1. The Pioneering Work of Rodney A. Brooks

In 1986, Rodney A. Brooks presented the *Subsumption Architecture* as a new kind of robot control structure based on simple behaviour producing modules (which could be fully wired from sensors to motor outputs) and, more complex ones added as separate layers (affecting the simple ones by inhibition and suppression mechanisms). Further, he made the following contributions to start solving the operation of robots within real world situations:

- *vertical decompositions* (where every component can combine several functions to contribute to a particular behaviour) instead of horizontal ones (where each component is responsible for a specific function like planning or, world representation) to guarantee real time response when needed (see also [Brooks-91b]),

- *new development decisions and analysis tools* to achieve the complete engineering of autonomous robot systems (see also [Brooks-91a; Brooks-91b; Brooks-91f]),

- *new architectures* to follow *biological inspirations* while achieving *autonomy*, *interaction* and, *reactive behaviours* (see also [Brooks-91c; Brooks-91d; Brooks-91e; Brooks-91f; Brooks-91g; Brooks-93; Brooks-94]).

1.1.2.2. Recent Generations of Mobile Robots

The classical robots were developed as *thoughtful creatures* able to respond to some well-known inputs (as discussed, for example, in [Brooks-85; Brooks-91g; Cliff *et al.*-92; Clancey-95]). By contrast, the recent Autonomous Robotics communities build robots as *reactive and autonomous creatures* able to perform some activities without being completely dependent on their designers [Mataric-91b; Maes-92b; Pfeifer & Verschure-95; Mataric-92a; Smithers-94; Pfeifer-94].

Following the pioneering developments at MIT, these last roboticists demonstrate the new intelligent skills of the robots without using much human intervention. Further, most of their work focus on finding new functional components (e.g. non-centralised control programs [Maes-92a; Mataric94a; Garforth *et al.*-97]) and tools (e.g. programming languages [Brooks-90a; Steels-96]).

Most recent generations of robots are built using *simple mechanisms* that can directly generate some behaviour-oriented activities. These basic mechanisms associate the selected behaviours to low level sensorimotor processes that can control the performance of the physical devices (e.g. wheel motors) without need of perception, planning or, execution units. The main idea is that the architecture has to include complete (non-dependent) behaviour-producing modules incorporating, for example, their own perceptual, modelling and planning requirements. Only some arbitration mechanism are aggregated so the robot can select (or decide) which module controls which part of its physical body, at any given instant in time.

More generally, there are three approaches to tackle the design of the new robot systems.

- *Brook's approach* based on an incremental but, handcrafted design of subsumption schemes with vertical organisations [Brooks-86; Brooks-91g; Mataric-94a].

- A *learning approach* that automates the design of the control systems [Thorton-96; Maes & Brooks90c; Barto-90; Aitken-94; Bryson-96; Barto *et al.*-95; Millan-94; Pipe *et al.*-94; Schmajuk-94]. Here the

computer sets up, at least, some control parameters that help to automatically generate the behaviour specification when the algorithm being used has a clear demonstrable convergence.

- An *evolutionary approach* that concentrates on evolving the behaviour-based control architectures using Genetic Programming techniques [Goldberg-89; Koza-92; Colombetti & Dorigo-92; Dorigo & Bersini-94; Reynolds-94; Beer-96]. Here the control system is evolved instead of being designed or, programmed by hand [Cliff *et al.*-92].

1.1.2.3. Some Explorations with Autonomous Robots

The literature reflects that most explorations with autonomous robots focus on:

- how to develop the systems,
- how to cope with the uncertainties of the world,
- what behaviours should be displayed and, how to solve behaviour-related processes.

The developments seem to be carried out so that there is no need to isolate the physical embodiments from the computational components [Brooks-91e]. The aspects of uncertainty that are investigated refer to: the uncertain values that are delivered by the sensors, the sensor readings themselves and, the uncertain effects of control/action executions. It is possible trying to avoid some negative effects of these uncertainties by designing tight couplings between the robot and its environment (e.g. using low level *sensing-acting feedback loops* [Brooks-91f; Horswill & Brooks-88; Horswill-96]) or, using the Fuzzy Approach [Zadeh-83; Bohner-95; Hercock & Barnes-96; Pipe & Winfield-96a; Saffiotti-97].

Behaviours such as *obstacle avoidance*, *wall following* or, *target-seeking* have a considerable remark in this area (see, for example [Brooks-86; Mataric-94a; Cliff *et al.*-93a; Schmajuk-94; Duchon-96]). Other investigations refer to the behaviour-oriented processes such as: *adaptation*, *robustness*, *autonomy* or,

emergence. Some behaviour-producing modules are described as being adaptive and/or robust because the robots that are equipped with these perform well under different environmental conditions (e.g., they improve their movements [Holland & Melhuish-96; Maes-91; Kaelbling-95; Horswill-95]).

The most adaptive and robust behaviour modules can be obtained, for example, identifying the right amount and type of control actions that have a global impact on the surroundings of the robot (see, e.g. [Mataric-94a]). In other words, *adaptive behaviour* is very close to *intelligent behaviour* [Beer-90; Maes-91; Smithers-92; McFarland & Bosser-93; Barto *et al.*-95]; it is the result of *an optimisation process* [Verschure & Pfeifer-92]. The concept of autonomy describes the interaction between the robot and its environment [Beer-95] or, *self-controlling* [Steels-89; Steels-95]. Finally, the emergence of behaviours (what is not designed) has negative and positive side effects. It has negative effects because "it is not predictable" and, it has positive effects because it means "less human intervention" [Steels-94a].

1.1.3. Major Insights

There exists strong ties between Autonomous Robotics and BB AI but, as [Steels-94a] suggests, they "should not be equated". The school represents, without any doubt, the main community working with practical solutions. Its major goal is the analysis, design and synthesis of behaviour-based control architectures for the robot systems to behave more autonomously than classical ones (see e.g. [Brooks-91b; Mataric-94a]).

The BB AI researchers, on the other hand, work on general ideas about the study of the behaviour-oriented models. They try to formulate how to replicate behaviour-based models. They also propose philosophical statements regarding the potential success of the behaviour viewpoint (see e.g. [Varela & Bourgine-92]). Further, they open up "an artificial life route to artificial intelligence" [Steels-93a; Steels-93b; Steels-95; Wilson-85; Wilson-91; Meyer & Guillot-94; Smithers-95; McFarland-92] while contributing to the following insights.

1.1.3.1. Objectives and Tasks; Models, Mechanisms and Systems

The main objective is building creatures that can truly sense their environment and physically act upon this [Brooks-91e], "artefacts which are 'really' intelligent... intelligent in the physical world, not just intelligent in a virtual world" [Steels-94a]. To achieve this, the work is focused on three major tasks:

- The *observation (analysis) of natural behaviour* displayed by living organisms within real environments (discussed in section 1.2).

- The *selection of models* based on, for example, internal and/or external mechanisms (e.g., simple sensorimotor processes or, more complex ones that include learning processes) that contribute to the production of behaviour in terms of actions and/or other functionalities.

- The *realisation of the models through the development of BB Systems* (see section 1.4), including some other useful mechanisms such as *behaviour-aggregation* (finding a way to connect the behaviours as to control the performance activities of the creature) or, *behaviour selection* (the ability to change from one behaviour to another and according to the experience of world [Mataric-90a]).

1.1.3.2. Slogan: Understanding Behaviour by Building Artificial Creatures

The slogan means understanding intelligent behaviour by building physical artificial creatures [Pfeifer-96a; Pfeifer-96b; Pfeifer-97]. Synthesis is necessary [Mataric-94a]. Indeed, the work *in simulation* tends to be considered as an early stage of development (that can reduce final production costs); as a tool for exploring and testing the artificial mechanisms although, "very often results from simulation only partially carry over to artificial systems" [Steels-94a].

1.1.3.3. Claim: Biological Fidelity of the Approach

KB AI lacks of biological inspiration [Brooks-91b]. The artificial systems built upon its principles cannot display it. BB AI, on the other hand, has a strong biological orientation. It takes inspiration from Biology to regulate behaviour change [Brooks-91a; Brooks-91b; McFarland & Bosser-93; Steels-94a; Mataric-95b]; it uses biological models [Pfeifer-94], as we shall see in section 1.2.

Engineers and biologists work together for constructing the artificial systems [McFarland-85; Brooks-86; Brooks-91b; Brooks-91e; Mataric-91b; Maes-92b; Pfeifer & Verschure-95; Mataric-92a; Smithers-94; Pfeifer-94]. This permits analysing the mechanisms that produce the BB forms of intelligence [Pfeifer-94; Brooks-91f], investigating what makes natural behaviour being adaptive and intelligent [McFarland-91]), how observed behaviour emerges from internal mechanisms [Steels-94a; Brooks-91g], how optimal behaviour can be associated to adaptation and learning abilities [McFarland & Bosser-93] or, what are the links between natural behaviour and learning processes [Maes & Brooks-90c; Brooks-93].

In summary, Nature offers "the keys" to reproduce behaviour (to some extent) and, the investigations advance so that the mechanisms being incorporated to the artificial creatures become closer to the biological models. In fact, one of the major claims is that it is necessary replicating more biologically inspired models of intelligence [Langston-90; Beer *et al.*-90; Beer-90; Moran *et al.*-95] instead of using, for example, simple, brain-like models [Roitblat-82; Gregory-66; Grossberg-80; Grossberg-86].

1.2. Autonomous Agents

Here we extend the major BB insights and argue for a multi-disciplinary area of research termed Autonomous Agents, based on the belief that a fundamental part of this contributes to the foundation (consolidation) of the behaviour-oriented approach to AI. In addition, we gather some aspects of successful

Computational Techniques (e.g. Fuzzy Systems) while making emphasis on the role played by Computer Science within the current investigations.

Autonomous Agents (AAs) is a group of research communities that investigate a more complex paradigm than that of BB AI. It represents the result of many years of investigations within a wide variety of disciplines (e.g., Computer Science, Cognitive Science, Neuroscience, Biology, Ethology, Developmental Psychology, Mechanical Engineering, Artificial Life) that look for the establishment of general theories about the study and construction of creatures (agents) able to act autonomously. Indeed, the literature reflects that this group appeals to a great diversity of ideas, formalisms and architectures that can be unified by the following common concerns:

- searching for *general principles to analyse, design and build the biological models* (see, for example, [Pfeifer-96a; Albus-96; Wilson-96; Horswill-96]),

- trying to find *new development methodologies and techniques* (e.g. bottom-up methods [Maes-92a; Pfeifer & Verschure-92; Maes-94a; Pfeifer-96a; Pfeifer-97] or, dynamical systems techniques [Smithers-92; Steels-94b]),

- defining and using *new characterisations of agents* (e.g. principled agent-environment interactions in robot systems and programs [Smithers-94; Agre-95; Verschure & Pfeifer-92; Beer-95]) and,

- investigating the use of *recent formal-frameworks* like, for example, hybrid computational techniques [Saffiotti-97; Zadeh-96; Azvine *et al.*-96; Goode & Chow-94; Goonatilake & Khebbal-95],

in order to guide

- the *study of living organisms*: their *morphology* (animals' bodies [McFarland-91; McFarland & Bosser-93]), their *underlying mechanisms* (e.g. observed learning mechanisms [Roitblat-82; Roitblat-94;

McFarland-95]) and, their *abilities* (innate and evolved behaviours [Gould-82; McFarland-85; McFarland-87; McFarland-92; Nepomnyashchikh & Gremyatchikh-96]),

- the *experimental work with artificial creatures*, focusing on the physical setups of robots [Clancey-95; Connell-90; Pfeifer-97; Ferrell-93; Scheider & Mataric-96; Webb-94; Webb & Hallam-96] or, the underlying control architectures [Beer-96; Kortenkamp & Chown-92; Belanger & Willis-96],

- the *investigation of behaviour-related capabilities* such as: *autonomy* and *self-sufficiency* [Wilson-85; Wilson-96], *cognitive control* [Sholl-87], *emotions* and *motivations* [Toates & Jense-91; Spier & McFarland-96], *adaptivity* and *learning* [Peng & Williams-92; Thornton-96; Scutt-94; Juille & Pollack-96], *coorperation* [McFarland-94; Mataric-94b; Arkin & Ali-94; Weiss-92; Parker-92; Kube & Zhang-92; Watt-96]), *evolution* [Floreano & Mondada-94; Cliff & Miller-96; Deugo & Oppacher-92], etc.

- the *design and synthesis of new artificial intelligent agents* [Beer *et al.*-90; Maes-90a; Maes-91; Maes-94b; Hallam-95; Mataric-95b; Pfeifer & Verschure-92; Pfeifer-96b; Steels-94b].

The success of the BB approach to AI requires some *cognitive*, *biological* and *technological* tendencies investigated within AAs. These tendencies are necessary to re-define concepts, ideas and methods to model and build the behaviour-oriented forms of intelligence. They study a wide variety of creatures (e.g. animals, brain-like models, cognitive structures or, hybrid forms) and help to smash the old dilemma of AI into a number of aspects of intriguing interest for the BB AI objectives. They investigate a very complex paradigm [Pfeifer-96b] and work on the establishment of quite complete theories. They use scientific basis that are necessary to consolidate the validity of the behaviour viewpoint although, as we shall see in section 1.3, there are many research issues that need being resolved before this can be demonstrated.

The following sub-sections describe all these common concerns and ideas in more detail.

1.2.1. What is an Autonomous Agent?

"Autonomous Agent" is a concept that has been proposed by researchers wishing to explore what acting intelligently means; a self-sufficient, adaptive and organised intelligent creature; a creature able to sequence its activities without need of human intervention; a concept that is associated (and sometimes identified) with many types of systems such as (see also [Maes-90a; McFarland-91; McFarland & Bosser-93; Pfeifer-96a; Pfeifer-97]):

- *Software agents*: computers and networks inhabiting in a cyberspace [Lyons & Arbib-89; Brooks-91f; Lynch-93; Maes-95; Nwana & Wooldridge-96c; Noble & Cliff-96; Pfeifer-97; Tyrrell-93; Tyrrell-94; Watt-96; Michel-96; Titmuss *et al.*-96; Belanger & Willis-96].

- *Robotic agents*: physical, synthetic robots [Brooks-91g; Cliff *et al.*-92; Clancey-95; McFarland-91; McFarland & Bosser-93; Blumberg *et al.*-96; Perkins & Hayes-96; Mataric-91b; Pfeifer & Verschure-95; Mataric-92a; Smithers-94; Pfeifer-94; Arkin & Ali-94] or, market (useful) robots [McFarland & Bosser-93].

- *Cognitive structures*: man-like machines based on assimilation processes, accommodation tasks and other related human intellectual skills [Khube & Zhang-92; Levenick-91; Albus-96; Williamson-96; Rutkowska-94; Blythe *et al.*-96].

- *Animats*: animal-like systems, cyber animals or, biobots that establish important landmarks in Robotics and Computer Science (see, for example, [Wilson-85; Wilson-91; Brooks-91e; Meyer & Willot-94; Werner & Dyer-92; Bersini-94; Blumberg-94; Blumberg *et al.*-96; Cliff & Miller-96; Rowe-98]).

- *Hybrids*: physical and software agents (e.g. [Dickinson & Dyer-96; Hallam *et al.*-97]) or, cognitive-software agents [Webb-94; Web & Hallam-96; Dygney & Gupta-94; Titmuss-96].

1.2.2. Understanding Behaviour-Oriented Views of Intelligence

There are many fields that have influenced the thinking in BB AI while investigating AAs. We describe the most relevant tendencies (also motivating the thesis presented in this document) in the following sections.

1.2.2.1. Understanding Behaviours

The concept of intelligent, natural "behaviour" is being studied from psychological, cognitive and ethological tendencies.

Psychology and the Study of Behaviours

Psychologists explore what *behaviour* means. They address numerous experiments while observing individuals, measuring their behaviours and, establishing theories that guide some attempts to develop artificial organisms. Their Reinforcement Learning Theories, for example, have been extensively used in BB AI [Mataric-91a; Prescott-93; Bersini-94; Digney-96; Humphrys-96; Mahadevan & Connell-91; Munos & Patinel-94; Ring-92]). Other interesting psychological experiments refer to the motivation and accomplishment of behaviour [Werner-94; Spier & McFarland-96; Toates & Jense-91; Deugo & Oppacher-92; Donnart & Meyer-94; Donnart & Meyer-96; Dellaert & Beer-96; Nepomnyashchikh & Gremyatchikh-96; Blythe *et al.*-96]. Some criticisms to this work refer to the limitations of the controlled environmental conditions on which they (psychologists) work (see, for example, [Mataric-94a]).

Cognition and Behaviours

The problem of building *full cognitive structures* has been extensively studied in KB AI and still represents a major issue in BB AI [Piaget-62; Toates-94]. In KB AI, cognition was associated with perceptual, learning, reasoning, thinking processes [Watt-96] and related Information Processing Theories [Grossberg-81].

The BB approach to AI, on the other hand, associates *cognitive* to *behaviour*. It focuses on lower levels of cognition [Clancey-95; Deugo & Oppacher-92; Prescott-96b]; it concentrates on "frames of cognition" for describing, for example, "time-varying interactions of the agent with its environment" [Toates-94].

Animal Behaviours

Modern Ethologists study the *behaviour of animals*. They focus on the causation, development and survival evolution of the animals' abilities and behaviours [McFarland-85; Bonner-88; Toates & Jensen-91]. They extensively contribute to the current discussions on innate abilities [Gould-82; McFarland-87], intelligent interactions [McFarland-85; Webb-96; Beer-90], learning and adaptive behaviours [McFarland-87, McFarland-91; Thornton-96; Roitblat-94; Steels-94a; Prescott-96b], ethological perspectives to the analysis of intelligent behaviour [McFarland & Bosser-93; Blumberg *et al.*-96] and, many other related issues.

1.2.2.2. Robots as Animals and Animals as Robots

The study of *robots as animals and animals as robots* represents another important contribution to the BB viewpoint [Brooks-91f; Pfeifer-96b; Smart & Hallam-94; Tani-96b; Todd & Wilson-92; Wiener-48; McFarland & Bosser-93; Todd *et al.*-94; Darley-94; Webb & Hallam-96; Werger & Mataric-96; Belanger & Willis-96; Prescott & Redgrave-96a]. This establishes some key insights about the behaviours that are displayed by both animals and robots (e.g. *the creature's interaction with its environment* [Gould-82; Arbib & Liaw-95], *adaptive behaviour* [McFarland-91; Steels-89; Beer-90; Steels-91], *autonomy, self sufficiency and self-control* [McFarland-91; McFarland-95], etc.). Other relevant investigations focus on "the learning problem in the context of AAs" [McFarland-85].

Here modern ethologists describe the learning problem in terms of *behaviour change* [Roitblat-82; Roitblat-94; McFarland-95]. Basically, these researchers suggest that natural learning mechanisms involve more complicated structures than simple stimulus-response connections and/or inhibitory/excitatory schemes

[McFarland-85; McFarland-87; McFarland-92; McFarland-95; Smithers-95; McFarland & Bosser-93; Green-94; Roitblat-94; Maes & Brooks-90c; Mataric-90b; Brooks-91b; Brooks-93]).

1.2.2.3. Brain-Like Models

Here autonomous roboticists and neuroscientists are working together for a better understanding of how biological and artificial neural systems work. Their common work points to general observations (formalisms and applications) about how to relate *structure and function of neural systems*, from lessons on Neuroscience to Technology and, vice versa [Gregory-66; Grossberg-80; Sholl-87; Tolman-84; Braitenberg-84; Scutt-94; McFarland-85; Edelman-87; Beer-90; Beer *et al.*-90; Brooks-91e; Maes-92b; Pfeifer & Verschure-95; Moran *et al.*-95; Smithers-94; Pfeifer-94].

The most significant *biological design principles* brought to BB AI can be found in the biological models used to design agent's brains. The engineers reproduce brain-like models using topology networks of Artificial Neural Network (ANN) Controllers and related Machine Learning Techniques (see section 1.2.2.5). They use neural network-like controllers to mimic, for example, the natural connections that link some sensory and motor neurones found in the biological brain. Indeed, most mechanisms used to construct behaviour-producing modules are based on explicit ANNs that link sensory devices of mobile robots to their actuators (motors) (see, for example, [Schmajuk & Blair-92; Tani-96a; Ziemke-96b; Munos & Patinel-94; Miller & Cliff-94]). Further, the work is updated with the latest *models, theories* and *systems* so that roboticists and neuroscientists help each other.

[Rowe-98] argues that the development of cyber-animals (also called biobots or, animats) is of interest for biologists who are trying to test causality links among certain neural activities and behavioural patterns in animals. Basically, as he argues, they (biologists) have spent decades trying to theorise how behaviours of animals are associated to neural activities of their brains. They observe and analyse patterns of behaviour while recording samples of their neural activities (neural signalling). However, it has been (and still is)

extremely difficult for them to demonstrate that both behavioural and neural activities are "causally" linked. They can isolate some neural networks and simulate the features of some external stimuli that fire the neural activities to control some activities. However, their work cannot be developed using real organisms. The building and use of *biobots* (cyber animals or, animats) helps them to carry out their experiments.

Using these artificial creatures (animats), they can reproduce certain neural signalling they record in the animal and, test whether or not these display the behaviours that have been advocated (anticipated) as causally associated. They use robot systems more often than computer programs because they think that the artificial creature has to face the real world. Furthermore, they work with roboticists and replicate the animal's mechanisms that are relevant for both neural and behavioural activities [Schmajuk & Blair-92; Prescott & Redgrave-96a; Prescott-96b; Prescott-97; Ziemke-96a; Ziemke-96b]). They work together for constructing the artificial versions of natural receptors and they need specific ANNs to truly control the behaviours of the creature when placed in the real world.

In summary, the use of artificial devices and ANN controllers help biologists to demonstrate what they try to theorise and this implies that "it is not necessary for biologists to look too deeply into the results of their experiments" [Rowe-98] because some behaviours can be displayed from extraordinary simple control mechanisms. Indeed, some biologists tend to complicate the description of behaviours and the way these are linked to neural activity because, for example, they assume that most behaviours need intentions (e.g., goal-seeking) that might be located their nervous systems. The problem is that they cannot find intention-like neural activities, perhaps, because this does not exist. Thus the simple mechanisms being investigated help them to understand that "life might not be as complicated as they think" [Rowe-98].

1.2.2.4. Genetic Side

Evolution is a "designing agent" [McFarland & Bosser-93]. It is characterized by the natural selection and reproduction processes that identify best elements in populations [Bonner-88]. It makes living organisms

learning in their lifetime while evolving their innate behaviours over generations. Further, evolution represents one of the most important mechanisms that effect on the formation of new structures in the brain (see e.g. [Edelman-87]).

From a more technical perspective, there exist an evolutionary approach to develop the artificial intelligent systems using Artificial Life techniques. Here the evolutionary tools are studied from a mathematical viewpoint. The main idea is evolving architectures instead of designing the systems by hand or, using learning mechanisms [Cliff *et al.*-92; Cliff *et al.*-93a; Cliff *et al.*-93b; Harvery *et al.*-94; Floreano & Mondada-96], although the differences between evolution and learning approaches do not seem to be clearly identified (they usually come together).

The tools used to evolve the architectures that control the activities of the agents fall into *Evolution Strategies* (see e.g. [Schwefel-81]) and, *Genetic Algorithms* [Goldberg-89]. Simple ANNs (finite state machines), for example, can be evolved using Genetic Algorithms [Wilson-91]. Dynamical ANNs can also be evolved using the same tools (see e.g. [Yamauchi & Beer-94; Cliff *et al.*-93b; Miller & Cliff-94; Hallam *et al.*-97; Colombeti & Dorigo-92; Floreano & Mondada-94; Floreano & Mondada-96]).

[Koza-91] reports some results about the simulation of navigation behaviours of a robot using genetic programming implementations. [Harvey *et al.*-94] evolve control architectures and use dynamical networks [Cliff *et al.*-93a]. [Gallagher & Beer-92] analyses whether it is possible generating appropriate control signals that allow an agent to exhibit adaptive behaviour and conceive this as natural consequence of the dynamics of recurrent networks.

1.2.2.5. Technological Perspectives

Here we describe some models, computer programs, programming languages and other techniques that have also contributed to the thinking in BB AI.

The Use of Models

The shift in AI represents passing a frontier in the definition and use of models. From the KB AI perspective, *the model is the system* whereas BB AI provides (is based on) a more physicalist view that might be stated as *only the system is the system* [Brooks-90b; Brooks-90c; Brooks-91a; Brooks-91e; Resnick-92; Brooks-94; Wilson-91; Mataric-91b; Mataric-94a; Pfeifer-96b].

Scientists use models to *describe possible representations about the real facts* (phenomena) that they observe and analyse [McFarland-87; Roitblat-94; Green-94]. Engineers, on the other hand, are involved with *the best and more efficient way to construct systems based on these models.*

BB Control engineers, for example, decompose the problem of controlling the behaviours of a creature into a number of sub-processes that facilitates the design, implementation and evaluation of the required systems [Hopfield-82; Cliff-91; Lynch-93; Arbib *et al.*-94; Cruse *et al.*-96]. Further, they combine all these sub-processes in such a way that optimal, global control is provided for the creature to perform efficient, coherent and robust behaviours within real-world situations [Mataric-91b; Maes-92b; Mataric-92b; Pfeifer-94; Steels-94a].

Robot Control Programs

The most valuable BB approaches to design robot control programs fall into (see also [Brooks-91f; Steels-94a]):

Algorithmic Approaches

Some behaviour-oriented control programs use traditional computer programming techniques (i.e. hand coded). An example can be seen in [Brooks-86]. Brook's architecture uses an algorithm that is compatible with Turing descriptions (state-space representations). Other algorithmical approaches are based on Fuzzy Logic strategies. An illustration can be in http://borneo.gmd.de/EIA/moria.html. Here the robot is provided

with ultrasonic sensors to supply some environmental (sensory) information that is converted into linguistic variables to control its navigations.

Artificial Neural Network Approaches

The ANNs can be used to develop control programs in close relation to neural, biological structures. These systems represent neural-like control structures based on neurone-like units that are weighted so that, when the sum of the weighted inputs to a unit exceeds a pre-established threshold value, an activation process propagates through the net and towards its output units [McCullock & Pitts-43]. The learning algorithms used to train these networks help to automate the design process of the control mechanisms being investigated [Maes & Brooks-90c; Schmajuk & Blair-92; Brooks-93; Millan-94; Tani-96a; Ziemke-96b; Spector & Stoffel-96].

Circuit Approaches

Here the programming of combinatorial circuits is used to design hardware implementations of behaviour-producing modules using VLSI techniques. The components of these circuits are connected one to other forming completely fixed networks, in a similar fashion to static (non dynamical) ANNs.

Dynamic Approaches

These help to formulate dynamic processes using, for example, differential equations (see [Smithers-93a]). Here the behaviour programs use a *cycle control loop* that maintains internal quantities and performs appropriate behaviour changes. The major advantage of the approach is that it gives rise to smoother behaviours since it is not limited to discrete, fixed conditions (see section 1.4.2.3).

Programming Languages

Some programming languages have been specially defined to develop BB robot control programs. [Brooks-90a], for example, presents a *behaviour language* and uses this to design Subsumption Architectures. [Lyons

& Arbib-89] presents another programming language for ANN-based control programs. The language described in [Steels & Vertommen-93b] serves to implement dynamical behaviour programs.

Environment-Based Programming Techniques

Environment-based programming techniques have been explored in BB Computer Science. [Basye *et al.* -95] presents an example of these techniques. These authors address agent-environment interactions in such a way that the agent influences input/output pairs of the environment it is presented by while exploring new strategies. The inputs of the environment represent the actions that the agent executes. The outputs correspond to some perceptual information that is available for the agent. [Barto *et al.*-95] presents a dynamic programming technique that solves the learning problem (when the system under control is completely unknown) while compiling some planning results into reactive strategies (real-time control). [Hammond *et al.*-95] also presents a very innovative approach to match agents to their environments. This illustrates an agent that *improves its performance* by stabilising its environment. [Agre-95] describes some simple techniques to develop program agents that decide *what to do next* from some useful structures that the agents find in their environment. He also describes *action-oriented perception* using Genetic Algorithm techniques.

Computational Intelligence: Fuzzy Systems, Artificial Neural Networks and Soft Computing

Computational Intelligence represents a significant shift in Computer Science. It replaces the classical Information Processing Theories (used by computer scientists for longer that 40 years) with *Agent Technologies* that incorporate *ANNs*, *Fuzzy Logic* and *Soft Computing* techniques.

Artificial Neural Networks

ANNs are considered as excellent control architectures because these systems allow storage without knowledge of location [Grossberg-78; Grossberg-80; Takagi *et al.*-92; Kosko-92; Bezdek-92]; because ANNs propagate activation rather than only storing symbols; because the methods used to design ANNs

differ from Hard Computing methods (i.e. centralised storage) (see, for example, [Millan-94; Tani-96a; Ziemke-96b; Mariano & Morasso-94].

There exist two main types of ANNs: *Feedforward ANNs* (FANNs) and *Recurrent ANNs* (RANNs). The *perceptron* was the first (more simple) trainable FANN [McCullock & Pitts-43]. Its learning algorithm compares output values with some desired outputs (the training data). The Multilayer FANNs contain several layers of units that are connected with learnable connections. Most learning algorithms that are associated to these nets consist of presenting input samples and determining how some error values can be reduced adjusting the weights of the connections. The RANNs (continuous-time response ANNs) are *non-linear dynamical systems*. They present very rich *temporal* and *spatial behaviours* (fixed points, limit cycles or, chaotic behaviours). Further, these nets can learn using unsupervised techniques as to self-organise themselves (see, for example, [Hopfield-82; Kosko-92; Mariano & Morasso-94; Bose & Liang-96] for more complete descriptions about these dynamical nets and their applications).

The learning techniques for the ANNs fall into two main groups: *Unsupervised Learning* and *Reinforcement Learning*. In the first group, the system is presented with samples but it does not know *how well* is doing. The second group makes the ANNs receiving some feedback (also called *reward* or *punishment*) that tells the systems whether their responses are right or not. Then, according to this feedback, the nets use a search strategy to try finding the outputs that correspond to maximum rewards. This reinforcement learning technique seems to be preferred upon the non-supervised methods, mainly, because the learner does not receive complete supervisions [Barto-90; Mataric-94; Munos & Patinel-94; Foner & Maes-94].

[Maes-89b] presents and describes a BB System that is constructed using ANNs (a hierarchy-based aggregation of ANNs). This and other examples of ANN-based BB Systems are described in section 1.4.2.2.

Fuzzy Systems

Fuzzy Systems (FSs) incorporate *fuzzy sets* [Zadeh-65] and *fuzzy logic methods* (see section appendix A) to cope with the uncertainties of real world situations [Bellman & Zadeh-70; Zadeh-72a; Zadeh72b; Zadeh-72c; Zadeh-72d; Zadeh-73a; Zadeh-73b]. They have been advocated because of their abilities to handle uncertain information using fuzzy sets, fuzzy weightings and other operations related to the aggregation and comparison of fuzzy sets of objects (see, for example, [Zadeh-72d; Zadeh-74; Zadeh-75; Zadeh-83; Zadeh & Kacprzyk-92]).

FSs and, more particularly, Fuzzy Logic Controllers [Mamdani-76; Pappis & Mamdani-77; Smets-88; Zadeh-83; Zadeh-89; Foulloy-93; Foulloy-93; Lee-90; Zadeh-83; Zadeh-89; Zadeh-92] have had successful applications in AI [Zadeh-83; Kandel-87; Zadeh-89; Dubois & Prade-83; Zadeh-72d; McNeill & Thro-94; Wang & Loe-93; Keller *et al*-92; Schmajuk & Blair-92; Brooks-93; Gorrini & Bersini-94].

However, as far as we are aware at the time of writing this document, there are not many examples of FSs within BB AI and AAs research. The unique references that we know at the time of writing this dissertation (i.e. [Bohner-95; Hercock & Barnes-96; Pipe & Winfield-96a; Pipe *et al.*-96b; Saffiotti-97]) focus on how FSs can deal with the complexity of the real-world situations that cannot be handled using either deterministic or stochastic frameworks.

This thesis emphasises *other potential benefits in the use of FSs* such as the flexibility of the fuzzy aggregation methods, fuzzy granularity and other aspects. Further, our research work has been motivated by the fact that FSs could be used in the tractability of complexity in behavioural competences and related interaction processes (i.e. in the development process of rather complex behaviour-oriented systems). Because, as we discuss in section 1.3, the complexity issues of the behaviour-oriented approach refer not only to the underlying uncertainty of the real world but also, to the complexity of the behaviour patters that must characterize the current artificial intelligent systems. Indeed, the current BB Systems have to reproduce

more and more complex behaviours displayed by natural systems and, *FL tools can help to reduce this complexity* [Zadeh-83; Zadeh & Kacprzyk-92].

Intelligent Soft Computing

Soft Computing introduces a new revolution of *Intelligent Software Systems* and *Agent Technologies* to develop *reactive software systems* using *Hybrid Intelligent Systems* [Zadeh-94; Zadeh-96; Smith & Mamdami-96; Titmuss et al.-96; Wilde-96; Nwana et al.-96a; Nwana & Ndumo-96b; Azvine et al.-96; Goonatilake & Khebbal-95]. Some of these hybrid systems are:

- *Fuzzy Neural Systems* [Narazaki & Ralescu-92; Keller *et al*-92; Goode & Chow-94; Gorrini & Bersini-94; Goonatilake & Khebbal-95] and,
- *Fuzzy Genetic Algorithms* [Wilson-85; Kosko-92, Mariano & Morasso-94; Wang & Loe-93; Donnart & Meyer-94].

All these examples suggest that the fuzzy approach is now bearing fruit.

1.3. Implications and Research Issues

Perhaps, the main problem is that the work carried out so far with autonomous creatures is not extensive enough to demonstrate all the claims that have been made with the behaviour viewpoint. In other words, things have only been demonstrated to work well in terms of designing principles and practice of simple behaviour-based models whereas, the understanding of more complex ones are still seen at a quite remote (even idealised) distance. But we think that the modelling of more realistic autonomous agents (discussed, not solved, within this dissertation) is not even well understood because the multi-disciplinary context of AAs raises too many research issues. Indeed, some of the ideas (conjectures) to build truly autonomous creatures are actually contradictory and/or exclusive.

The difficulties found in defining general design techniques [Maes-94a; Pfeifer-94; Pfeifer & Verschure-95; Pfeifer-96b], for example, refer to the fact that the design principles that can be of interest for the cognitive scientists differ from those defended by the engineers [Pfeifer-96a; Brooks-91b]. Other serious research issues refer to the biological fidelity of the systems being developed. It has been argued that the Organic Neural Networks cannot be compared with the ANNs being used (see [Brooks-94; Pfeifer-96a; Pfeifer-97; Rowe-98]); that biological methods of assessment are of no use in appraising ANNs. Formal models are only and, the work with ANNs is fundamentally motivated by a very distant goal i.e. the understanding of the real, complex nervous system. Perhaps we have to wait until biologists know more about this biological system ...

Furthermore, we believe that, nowadays, there is no general, single methodology to fully support the biological fidelity of the behaviour-based models. First, because we are still not sure about the nature of the behaviours and the way these can be related to the internal workings of the living organisms. Second, because the techniques currently used remain too simple to replicate what biologists have experimentally demonstrated about how neural systems and behaviours relate to each other.

In summary, "AAs" can be understood as a substantial conceptual problem arising from those different perspectives that have contributed to its emergence. However, many authors now feel (e.g., [Brooks-91b]) that discussion on this is untimely unless the other, technical issues (discussed in section 1.4.3) concerning the viability of the approach can be resolved. It is necessary finding, at least, the basis of more general frameworks to build large-scale behaviour-based models and, in order to achieve this, it is worth trying to resolve the technical problems found with the current approaches to develop BB Systems.

This thesis has been strongly motivated by some of these technical research issues. The research work that is presented in this dissertation focuses on the problem of modelling large-scale behaviour-oriented networks. It follows some suggestions in [Brooks-94] that point to a possible solution by looking for other (new) levels of abstraction rather than the action selection mechanisms most widely applied in BB AI [Maes-89b; Maes-90a; Beer et al-90; Kortenkamp& Chown-92; Mataric-92b; Parker-92]. The FBMs (presented in chapters 2 to 4)

are based on innovative levels of abstraction at which it is possible arriving through a number of refinement stages.

1.4. The Development of Behaviour-Based Systems

The main idea is following *a bottom-up approach*: "... start by decomposing the problem into pieces, solving the sub problems for each piece, and then composing the solutions" [Brooks-86].

1.4.1. General Steps

The most general steps for building behaviour-oriented artificial models can be summarised as follows:

1. identifying behaviours,
2. constructing the system to perform these competences,
3. leaving the creature to operate within its real environment,
4. recording data from the way it performs the competences and,
5. comparing these to the original behaviours (the ones that have been observed),
6. stepping back (if necessary) to modify the model until observing more efficient behaviours.

Other key principles about the development of BB Systems (BBS) are outlined below (see also [Maes-90a; Maes-92b; Mataric-92b; Mataric-94b; Mataric-95b; Maes-94a; Steels-93a; Steels & Vertommen-93b; Steels-94a; Steels-95; Pfeifer-96b]).

1.4.1.1. Selecting and Describing Behaviours

The development of a BBS usually starts selecting the behaviours, observing the creature that is able to perform these and, analysing *what causes them* [Mataric-92b; McFarland & Bosser-94]. However, the mechanisms and/or processes that cause the behaviours are not always observable i.e. analysis does not always help.

Following [Steels-94a], "functionalities and behaviours belong to the descriptive vocabulary of the observer" whereas "the mechanisms play a central role in establishing the behaviours".

- The *functionalities* (i.e. avoiding obstacles, locomotion, etc.) represent what the agent needs to achieve (what robots might be designed for [Brooks-96e]; tasks or goals [Maes-90b; Millan-94]).

- The *behaviours* represent more theoretical units than the functionalities. Sometimes, a behaviour unit can be identified with the functionality (or, functionalities) to which it contributes. However, this is not the normal case. Usually, several behaviours contribute to the same functionality and depend on very complex internal mechanisms of the creature and external aspects of its environment. Further, the same behaviour can be characterised in a number of different ways. It is possible, for example, identifying agent-environment interaction processes [Smithers-95] or, functions dependent on the internal states and sensory information of the creature [Pfeifer-96].

- The *mechanisms* have physical existence and cause the behaviours. A mechanism can be either an internal component of the agent (e.g. sensors, internal states, adaptive mechanisms) or, an internal process (e.g. a change of an internal state or, a the transformation of this into control commands, etc.). Simple mechanisms can give rise to very complex behaviours [Braitenberg-84; Steels-94; Pfeifer & Verschure-95; Pfeifer-97].

Therefore, neither the behaviours nor the mechanisms might be easily described. It is necessary studying, describing and combining the related functionalities, components and/or processes that can be easily designed and implemented.

1.4.1.2. Designing and Implementing the Behaviour-Producing Modules

Once we have been able to describe the selected behaviours (using related components, mechanisms and/or processes), it is necessary designing and implementing the modules able to produce them. Some guidelines for the development of these *behaviour-producing modules* are:

- *Self-contained behaviour-producing modules.* Each module is responsible for producing behaviour. It does not need to depend on any other module although interaction might help to produce more complex behaviours (see below). The creature has to be able to decide what actions to take without using any reasoning engine or, centralised module [Mataric-92b].

- *Avoid user-driven methods when the environment interactions are needed.* These modules have to be developed so that they can be directly related to the environment of the agent.

- *Keep them simple.* A strong tendency is designing the behaviour-producing modules as simple as possible; selecting the simplest mechanism that produces the desired behaviour. This is directly associated to the dislike of complex objective world models (discussed, for example, in [Brooks-91e]).

- *Design reactive modules.* The behaviour-producing modules cannot be implemented as static structures. They must be able to react to the internal changes of the agent and, the external changes of the physical world. For example, a reactive obstacle avoidance module must be able to make the agent avoiding new obstacles placed within its real environment.

- *Use reactive control rules.* One of the simplest way of implementing the behaviour-producing modules is using simple "if condition do action" rules. In the presence of a condition, the rule is activated so that control is driven to the actuators of the robot. Other more complex mechanisms can be used instead. For example, we can associate strengths to the rules using, for example, ANNs. Section 1.4.2 describes some of these control architectures.

- *Specialisation can reduce complexity.* It is much more effective concentrating on the development of mechanisms that produce the desired behaviour (e.g. those that can couple sensing and acting) rather than designing general-purpose modules [Brooks-91b, Steels-94a]. And, in order to develop these mechanisms, it might be helpful exploiting the physics of the world and the morphology of the agent [Brooks-91b; Smithers-94; Smithers-95; Pfeifer-96a; Pfeifer-97]. For example, [Horswill-92] develops a navigation module and demonstrates how the complexity of this can be reduced by making assumptions about the physics of the environment. He optimises some sub-activities rather than encoding situation-specific information. Further, any designer can improve the performance of an agent by specialising the modules. [Horswill-95] makes distinctions between "specialisation to an environment" (related to the properties of the environment) and "specialisation to a task" (formal transformation that help to map mechanisms with behaviours).

- *Simple and more complex modules.* Some behaviour-producing modules might be implemented very easily (e.g. sensor-motor connections) whereas other, more complex ones might require more computational effort (see, for example, the map building behaviours described in [Mataric-90]).

1.4.1.3. Organising and Aggregating the Behaviour-Producing Modules

The next step consists of organising and aggregating the behaviour-producing modules into a single architecture so that the agent can optimally switch (or select) from one to another. The most general principles for building this architecture can be summarised as follows.

- *The architecture has to be open, scalable, distributed and interaction-based.* The overall architecture has to be able to cope with the changes of the environment. Another major concern is producing very scalable architectures so that new behaviour-producing modules can be aggregated to earlier ones. It is also necessary using highly distributed structures because the modules must be able to operate in parallel [Maes-92b] while allowing fast reactions [Smithers-95]. Finally, the architecture must incorporate the modules in such a way that these can interact with each other as with the physical world (when needed).

- *Selection mechanisms might be useful.* Some arbitration methods are commonly used for the agent to select the appropriate behaviour. This can solve multiple conflicting actuator commands. For example, [Maes-89b] implements a "winner-take-all" arbitration network whereas [Brooks-86] implements a hand-coded "priority network" (see also [Kortenkamp & Chown-92]).

- *Hierarchical organisations are not needed.* The organisation of the behaviour-producing modules can be based on a hierarchy of layers (as presented in [Brooks-86]) but, this is not needed; it does not seem to be a flexible organisation [Maes-92b; Maes-94a].

- *Evaluate environmental changes.* The architecture should be able to adapt the behaviour selection to the environmental changes. [Maes-90] presents a BBS (called Action Selection Mechanism) that evaluates some environmental changes doing parallel searches of actions.

1.4.1.4. Hardware and Software Platforms

There has been a large amount of work on building the hardware and software platforms for BBSs. Some of these are described in sections 1.1.2 (robot control systems), 1.2.2.5 (technological tendencies within AAs) and 1.2.3 (the role played by Computer Science using, for example, ANN techniques). Other examples are outlined with the classification of approaches presented below.

1.4.2. A Classification of Approaches

Several authors have classified the development of BBSs according to different, relevant criteria. [Maes-94a], for example, distinguishes between "developmental approaches" and, "incremental approaches". A more general classification is discussed in [Pfeifer-96]. This author categorises the work according to three different perspectives: *functional, learning and developmental* and, *evolutionary*. He suggests that, in practice, these perspectives "contribute in a complementary way". [Steels-94a] follows the line of how BB AI seems to contribute to Artificial Life and looks at the progress made in designing and implementing specific competences (navigation towards a target, avoidance, etc.).

In this dissertation, we provide another valuable classification based on the following *development decisions*:

e) *the description and definition of the behaviour units* (i.e. procedural, algorithmical, etc),

f) *the methods used to design and implement the BBSs* (i.e. aggregation methods, how to drive the functionality, etc.) and,

g) *the selection of formal frameworks, techniques and tools* (e.g. ANNs, Genetic Algorithms, etc.)

and discuss how the major achievements of the behaviour-oriented viewpoint roughly fall into five tendencies: *Subsumpion Architectures*, *BB Networks*, *Dynamical BBSs*, *Evolutionary Approaches* and, *Hybrid Paradigms*.

1.4.2.1. Subsumption Architectures

Brook's Subsumption Architecture (SA) [Brooks-86] and its hierarchical versions (see e.g. [Rosenblant & Payton-89; Ferrell-93]) represent the first *reactive architectures* designed through the use of layers of robot competences that operate asynchronously. Conflict resolution and communication is allowed between the

behaviour-producing modules of these architectures. Other important features of this approach can be summarised as follows (see also [Kaelbling-87; Maes-89a; Connell-90]).

In all SAs (mostly developed at MIT), each behaviour unit is described as a layer of control that couples sensing and acting capabilities of the agent; each behaviour module hides low level design details.

The behaviour units of a SA are described using algorithms and Turing compatible functions. Each unit is an Augmented Finite State Machine with a number of internal registers used to hold input (e.g. sensory signals) and output (e.g. the result from some computations) values as well as some states through which it cycles until a condition is reached. At any instant of time, each unit can either move to another state or, change the contents of its internal registers.

These behaviours are *self-controlled*: they can be active (at any instant in time) and change their states according to the input hold in the corresponding internal registers. Further, the SAs can use an *internal clock* to make the behaviours waiting during a certain time and resume operation after that.

The overall control mechanism is based on hierarchical and distributed *subsumption relations.* All the behaviour units operate together and drive control from sensors to actuators. One behaviour unit can inhibit some input values (or other action parameters) of another and so, stop the second behaviour unit to change its state and resolve computations according to this.

The development of these architectures (also called BB programming) consists of building control layers in an *incremental fashion* [Brooks-91f]. New, higher levels behaviour units can be aggregated while the already implemented ones keep functioning. Then *conflict resolution mechanisms* are used to arbitrate the different control layers that are needed for the system to perform correctly. These mechanisms usually work at the action level i.e. using a *priority scheme* to decide which particular behaviour unit has its output(s) routed to the actuators of the system. These simple action-arbitration schemes seem to be appropriate for navigation tasks. The states of each lower level control layers (e.g. leg up) can be used as a precondition to provide

coherent sequencing of movements and, the higher level layers can modulate the motion of the lower level ones.

The use of pure Finite State Automata architectures is large in the scope within the SA approach to develop BBSs. There have been a number of state-based characterisations of behaviour-oriented controllers using different variable sets. [Holland *et al.*-96], for example, defines a motion controller using bi-directional sensitivities provided by radial distances to points of sources that the agent has to find. [Mataric-90b] uses Subsumption Architectures to implement wall-follow and avoidance behaviours.

Other developments incorporate *learning capabilities* to the first, "brooksian" approaches; focus on trying to solve *how to learn behaviours* using similar subsumption mechanisms. [Mataric & Brooks-90c] describe this problem and propose an ANN-based solution. Here the behaviour units have associated strengths and make the system modifying the behaviour outputs according to the inputs that it receives (e.g. using sensor/actuator calibrations).

1.4.2.2. BB Networks

This group includes most connectionist approaches used in BB AI [Bond & Gasser-88; Cruse et al.-92; Schmajuk & Blair-92; Tyrrell-93; Werner-94; Schmajuk-94; Ziemke-96a; Ball-94; Aitken-94; Munos & Patinel-94; Yamauchi & Beer-94; Morasso & Sanguineti-94; Beer-96; Donnart & Meyer-96; Ferrel-96; Blumberg *et al.*-96].

The Action Selection Mechanisms [Maes-89a; Maes-89b; Maes-90b; Beer *et al.*-90; Tyrrell-92; Tyrrell-93; Tyrrell-94; Weiss-92; Kortenkamp & Chown-92; Blumberg *et al.*-96; Humphrys-96] represent a clear example of this type of architecture. Here, the definition of the behaviour units consist of mapping the required functionalities into local, simple operations of (simple) processor units that operate in parallel. The processor units connect to each other within a network structure. The output of one processor unit is an input

of another processor unit. The signals propagate through the network and relating sensing to acting. Several processing units (a sub network) can be responsible of a specific behaviour.

The BB Networks are characterised with *parallel and local operations* that facilitate the design and overall performance of the behaviour-oriented control schemes. [Narazaki & Ralescu-92; Aitken-94; Millan-94; Schmajuk & Blair-92; Tani-96a; Ziemke-96b; Munos & Patinel-94; Miller & Cliff-94; Moran *et al.*-95]. Additionally, *mathematical formalisms* can help on the representation and manipulation of the underlying structures of the BB Networks.

Control in simple BB Networks is due to the *selection of parameters* [Beer *et al.*-90]. *Multi-layered networks* can be used to implement more complex BB Networks and, more particularly, to combine several behaviour units (each provided within one of the layers). *Categorisation* is also available in the scope of the BB Network approach. This allows the agents to distinguish, in real time (while the agent performs some actions), between different aspects of their environments. An adaptive categorisation is investigated in [Scheicher & Lambrinos-96]. Their approach includes some actions of the agent into the categorisation of objects.

Traditionally, the classification processes were solved in the basis of information processing techniques that allowed the creature to map some input data (sensory information) onto internal representations and, sometimes, via supervised learning schemes. The main problem of this classical approach was the wide number of input patterns that could be mapped onto the same internal representation. In the BB Network approach, on the other hand, adaptive categorisation can be achieved through sensory-motor coordination [Maes & Brooks-90c; Brooks-93]. The Temporal Kohonen's map described in [Kohonen-82] implements such categorisation processes including conditioned associations between learned sensory-motor mappings and some behaviours units. An example of this map can be seen in [Ball-94].

Most BB Networks have been implemented using ANNs. Action Selection Mechanisms are developed using, for example, RANNs [Blumberg *et al.*-96]. These networks usually associate sensor signals to motor outputs. Further, ANNs are beneficial for the construction of BB Networks, mainly, because their *learning power* can automate the design process. *Cognitive Mapping* represents another valuable technique to develop BB Networks. This technique has been widely used to develop navigation systems [Gould-86].

A Cognitive Map represents a record in the central nervous system of macroscopic relations between behaviourally important points in their environment. It is motivated by the fact that some animals (e.g. insects) record these relations and use them to plan movements through their environment (to navigate according to the geometric position that is represented in the animal's metric maps). *Topological* and *Feature Maps* are special types of Cognitive Maps [Tolman-84; Sholl-87; Laszlo *et al.*-93; Smart & Hallam-94; Tsuji & Li-92]. The *Map Builders* (e.g. [Levenick-91; Drescher-91; Laszlo *et al.*-93]) can also be classified within the same group. These allow the agent to learn navigation competences through Self-Organising ANNs [Malaka *et al.*-96]; these network-like systems do not represent static models of the world because they can use dynamic processes such as "moving averages from duration to turn angles" [Mataric-95a]. *Topographic maps* are also developed using similar BB Networks (see, for example, [Morasso & Sanguineti-94; Ferrell-96]). These can orient the agent towards sources of stimuli (e.g. visual and auditory ones). Orientation behaviour can serve the agent to direct its sensory signals while making it to explore its environment. These maps seem to be inspired by the discovery of organised networks within the brain of mammalian vertebrates [Kohonen-82]. In practice, these networks are reproduced with structures of spatio-temporal representations. For example, the Topographically Organised Networks used in [Ferrell-96] self-organise and effect multi-modal information (of the space and temporal domains of the agent and its environment) according to experience. The dynamics of this network is affected by the behavioural performance of the agent.

BB Networks vary in form and complexity. Most of them are characterised with simple units (e.g. TLU's used in Perceptron FANNs). There are also more complex BB Networks where the units of the net are

associated to specific components of the internal mechanisms/structures of the agent (e.g. actions and goals). [Maes-89a] represents a clear illustration of these networks. [Maes-90a] provides a wide review of Action Selection Mechanisms (and other BB Networks). The Action Selection Mechanism presented in [Maes-90b] includes time-varying goals using explicit representations of the agent's goals.

[Maes & Brooks-90c] provides some examples about how to learn (select) behaviours (actions) while trying to solve some problems found with the original version of Maes' architecture (ASM). This last architecture introduces some ideas about how to scale-up the BBS (when the arbitration mechanism cannot cope with large action-selection problems). The system also introduces time-varying goals; it can be classified within "learning to select behaviours" (focuses on what the agent learns).

The *learning algorithms* are used to adapt the weighting factors of networks. In explicit applications of ANNs, several learning algorithms have been proposed and tested according to the *time of convergence* (or time required for the learning method to settle on a stable set of weighting factors that provide the required control outputs of the ANN controller). There are *supervised* and *unsupervised* learning methods. The unsupervised methods, however, seem to be more beneficial for the BB objectives (see e.g. [Mataric-91a; Bersini-94; Schmajuk-94; Munos & Patinel-94]). In [Mahadevan & Connell-91], the agent concentrates on learning the arbitration network among some primitive actions. This represents another example of "learning from experience" (as suggested by Maes); further, the agent seems to learn "new composite actions" in applying Reinforcement Learning through the use of a mechanism that can generalise over the state space. [Maes-92a] presents a *model builder* (learning to select behaviours). She presents an approach to design spreading activations where the behaviour is defined in terms of conditions, primitive (or composite) actions and expected results. The architecture is purely "goal-oriented" since the exploration strategy biases experimentation of the environment towards the goals of the agent.

1.4.2.3. Dynamical BBSs

This approach to develop BBSs uses dynamical control mechanisms as opposed to conventional information processing architectures [Smithers-95; Steels-95; Tani-96a; Tani-96b; Thornton-96; Verschure & Pfeifer-92; Gallager & Beer-92; Neven *et al.*-96; Wilde-96; Pfeifer-96a] or discrete computational systems [Steels-93a].

The main idea is incorporating dynamic aspects into the description of the observed behaviours i.e. avoiding the use of discrete characterisations of behaviours.

The description of a *dynamic behaviour unit* does not need specifying neither observer-based interpretations of the environment (i.e. abstracting external situations that effect on the agent's behaviours) nor static associations between these interpretations and appropriate control actions (e.g. fixed sensor-to-motor actions). A dynamic behaviour unit is like a property of a two-component dynamic structure i.e. the interaction space of the agent and its environment [Smithers-95; Steels-95]; like a sub-system that includes different structures and processes, extracts the information it needs from the physical world and, decides when to become active.

A *dynamic behaviour unit* is commonly defined as a set of processes that links many continuously varying variables [Steels-94a] like sensor readings or, motivational variables [McFarland-87; McFarland & Bosser-93]. Each of these processes can increase / decrease a quantity (as a consequence of the evolution of some other quantities) while playing different roles in one or more behaviour units.

The design of the Dynamical BBSs consist of using *dynamical system equations* and, representing the behaviours that the agent can perform in terms of the *stable solutions* of these equations (variable and value pairs). Further, there are two key principles for designing these systems: *combination* and *cooperation*. The overall control mechanism of a Dynamical BBS can be a distributed one. The behaviour-producing modules, however, do not influence each other using a subsumption relation. The "combined effect of the behaviours is added at the level of actions" [Steels-94a]; the behaviour units can cooperate through their effects at the

level of actions. Therefore, additive control replaces the layering used, for example, in Subsumption Architectures. All the dynamic behavioural solutions are located at the same level and help on *scalability* issues that cannot be easily addressed using simple Subsumption Architectures.

The formal frameworks, techniques and tools used to develop Dynamical BBSs (e.g. difference equations of continuous-time dynamic systems) are very similar to those used in Classical Control and Dynamical System Theories. The recurrent versions of ANNs can also be used to implement Dynamical BBSs. [Smithers-95; Schmajuk-94], for example, use these networks. [Beer-95] provides an interesting example of a Dynamical BBS. This author presents a rather theoretical framework designing an agent and its environment as "two coupled dynamical systems" whose interaction is responsible for the agent's behaviours. The two dynamical systems of Beer's framework fit in an adaptive fashion that can be analysed from the satisfaction of constraints in the underlying trajectories.

1.4.2.4. Evolutionary Architectures

Evolutionary Architectures try to evolve the BBSs instead of having to design them by hand or, using learning mechanisms (see, for example, [Cliff *et al.*-92; Deugo & Oppacher-92; Gallagher & Beer-92; Cliff *et al.*-93a; Cliff *et al.*-93b; Floreano & Mondada-94; Floreano & Mondada-96; Saunders & Pollack-96; Juille & Pollack-96; Werner & Dyer-92; Colombeti & Dorigo-92; Werner-94; Harvery *et al.*-94; Reynolds-94; Beer-96; Cliff & Miller-96]).

An Evolutionary Architecture represents an evolving structure that looks for some internal organisations. Here each behaviour units represents a solution in a space of solutions of the architecture. [Koza-91] shows how to evolve programs for obstacle avoidance and wall-following behaviour systems. The primitive building blocks of the program are: sensory inputs and actions

The methods used to design these structures start from a population of behaviour systems, each representing a solution in a certain space of solutions. Individuals with higher fitness reproduce more often and, therefore, their distribution changes in the global populations.

Reproduction means that copies are made, possibly after mutation (which introduces a random change), or recombination (which combines parts of two algorithms). Recombination may potentially result in a better algorithm. The algorithm can be further reinforced by the selection step so that the overall process evolves towards better regions of the search space. However, as I describe latter in section 1.4.3, this work has only been proved of benefit in the simulation of rather simple behaviours.

The behaviour-oriented systems are evolved using Artificial Life Techniques like:

- *Evolution Strategies* (see e.g. [Schwefel-81; Brooks-91f]) or,
- *Genetic Algorithms* [Goldberg-89] usually operating on *Classifier Systems* [Holland-85; Wilson-85; Booker-88; Iba *et al.*-92; Dorigo & Bersini-94].

These techniques allow some parameter optimisations using bit strings that represent the populations of the behaviour units being evolved [Floreano & Mondada-94; Floreano & Mondada-96; Reynolds-94]. [Koza-92] reports on a more innovative, higher level of abstraction for representing Genetic Algorithms. RANNs have also been used in combination with Genetic Algorithms to speed up the evolution of the BBSs (see, for example, [Beer-96]).

1.4.2.5. Hybrid Paradigms

There are BBSs that cannot be classified within any of the approaches that I have described before. These represent the group of Hybrid BBSs.

The hybrid behaviour units can be described, for example, as dynamic and distributed systems.

Some hybrid development methods are discussed in [Hendler-88; Cliff et al.-93b; Koza-91; Laszlo *et al.*-93; Goode & Chow-94].

Fuzzy, Neural and Genetic Systems have been used, in combination, to develop Hybrid BBSs (see, for example, [Goonatilake & Khebbal-95; Zadeh-96; Smith & Mamdami-96; Titmuss *et al.*-96; Wilde-96; Nwana *et al.*-96a; Azvine *et al.*-96; Hercock & Barnes-96]). Various authors (e.g., [Pfeifer-96a]) have suggested the potential benefits of using these hybrid approaches. The FBM approach presented in this dissertation can also be classified as a Hybrid BBS. In fact, as I describe latter in chapters 2 to 4, FBMs have been widely motivated by the features of BB Networks and Dynamical BBSs.

1.4.3. The Technical Issues

Generally speaking, all the current approaches to develop BBSs work for a short number of rather simple behaviour patterns (i.e. with short-scale behaviour-based models); they all can make an artificial creature displaying some simple behaviour and/or actions although there are, of course, important differences from one group to another. The current investigations, however, advance towards the construction of large-scale behaviour-based models incorporating more and more complex behaviour patterns and, here is the problem. Each group faces some kind of technical problem that makes these complex models difficult to foresee. In fact, the validity of the approaches does not seem to be demonstrated with the upper levels of the bottom-up development processes.

SAs based on complex behaviours seem to be difficult to foresee. These architectures have a limited scalability because the behaviour units have to be built on top of others and, the required subsumption relations can only work well with basic sensory-motor competences [Steels-94a].

The design and implementation of BB Networks using ANNs is not sufficient either for the development of a complex BBS. These BBSs might be able to perform different (even complex) behaviours which design is

limited by the use of ANNs. It is difficult to express the behaviours in neural-network terms [Steels-94a]. Further, the nets do not really learn from experience [Pfeifer-96a; Pfeifer-96b]; they behave as fixed structures (after the training phase) that limit the adaptation of the system to its changing environment [Pfeifer & Verschure-95; Ziemke-96a; Ziemke-96b]. Building non-computational agents supervised learning methods also face a number of problems. [Steels-94a] argues that the "supervised methods require a teacher which is more intelligent that the agent". Further, the time of convergence of these methods grows with the number of behaviours that are implemented within the architectures. Reinforcement learning also faces serious difficulties with physical AAs (e.g. "it is unrealistic to assume that the gent gets a clear scalar reinforcement signal after each action or series of actions" [Steels-94a]). [Maes & Brooks-90c] suggests that the problem of learning new behaviours will not be solved in a near future; that it might be necessary finding new learning techniques that can provide an automatic acquisition of new behavioural competences.

The Dynamical BBSs also require further investigations, mainly, because these systems seem to lack of appropriate integration methods. Following [Steels-94a], there exist an urgent need of formal frameworks to develop these systems (e.g., mechanisms able to integrate the time-varying behaviour sub-systems). Other disadvantages of Dynamical BBSs discussed in [Steels-94a] are: a) the design is hard to admit by those who are used to algorithmic approaches, b) it is not possible to explicitly control the timing of actions. [Steels-95] suggests that there exists a fundamental problem in the implementation of structural integration of dynamic BB units.

[Brooks-91f] and [Steels-94] argue that the Evolutionary approach is still not a feasible solution. According to this survey, the main problem of this approach is that it takes too much time to converge. [Brooks-91f] also explores the difficulties inherent in transferring the evolved programs in a simulated environment to run on an actual robot. Other problems discussed by this author are: a) the design of the evolutionary control systems depends on structures of search which have to be carefully designed (and need to much computational effort [Steels-94a]) and, b) natural evolution affects neural controllers and physical entities

through the same genetic mechanisms whereas, in the experiments which have been reported so far, this type of evolution (co-evolution) does not take place. The problem with the Hybrid approaches is that these require additional integration mechanisms that complicate the design process [Mataric-94a]. It is not possible guarantying that such hybrid methodologies can help on the construction of large-scale BBSs.

Finally, the technical problems of the BB viewpoint seem to go beyond the suitability of the current approaches to design and implement these systems. In 1994, Rodney Brooks argued that most of these BBSs fail in their approach to reproduce natural systems due to the levels of abstraction that were used by their designers. As he discusses, most of these approaches focus on the abstraction of priority schemes to select among one (or maybe more) particular control action(s). The SA, for example, uses fixed selection schemes to define the sequences of the agent's competencies from the activation of some logical rules (preconditions). Action Selection Mechanisms are based on the abstraction of actions, goals and environmental conditions [Maes-89b]. Their arbitration scheme (in the top level of the hierarchy of behaviours) incorporates representational information (from the environmental conditions that the robot can experience) to facilitate the execution of lower level tasks (show, for example, in [Mataric-94]). "... We must be careful that we choose the right abstraction to simulate. Traditional AI chooses the symbol level as a higher abstraction. Many now feel that this was the wrong level of attack. The situated robotics community must also be prepared to carefully, and rethink, about its level of abstraction" [Brooks-94]; "... an appropriate level of abstraction has to be found" [Pfeifer-96a].

Chapter 2

Fuzzy Behavioural Maps

Abstract

In this chapter, we start presenting the contributions to knowledge of the thesis i.e. the work to date exploring and evaluating the role of the *Fuzzy Behavioural Maps*.

More particularly, the following sections present:

a) The concept of a Fuzzy Behavioural Map based on some influences from Kosko´s Fuzzy Cognitive Maps, other Cognitive Mapping Techniques and BB Systems

b) Some examples of Fuzzy Behavioural Maps of varying degrees of complexity ("simple" and "more complex" Fuzzy Behavioural Maps)

c) The most general intuitions we have about Fuzzy Behavioural Maps as designing tools of behaviour-based models (their possible forms and uses).

2.1. Influences and Motivations

The concept of FBM draws its inspiration from many sources. Foremost amongst these are Kosko's Fuzzy Cognitive Maps (FCMs) [Kosko-86a; Kosko-88], Adaptive FCMs [Kosko-94] and, other Cognitive Mapping Techniques (e.g. Cognitive Maps used to represent organism-environment interactions [Tolman-84; Sholl-87; Tsuji & Li-92; Levenick-91; Brannback & Malaske-95]).

Non-hierarchical behaviour decompositions [Brooks-86; Mataric-94a; Mataric-94b], BB Networks [Maes-89b; Tyrrell-92] and, Dynamical BBSs [Smithers-94; Smithers-95; Steels-95] also influence the definition and use of the FBMs.

The following sub-sections (2.1.1 and 2.1.2) describe the major aspects of these various systems that we have considered to define the concept of FBMs.

2.1.1. Fuzzy Cognitive Maps

FCMs model causal webs with simple directed graphs [Kosko-86a; Kosko-88] that "provide qualitative information about the (hidden, non apparent) inferences in complex social and psychological models" [Craiger & Coovert-94]. Their nodes and edges show how some concepts of a specific model effect others. They are also non-linear dynamic systems because they incorporate feedback connections [Kosko-88].

FCMs have been used as alternative techniques to classical mathematical models [Kosko-86], differential equations [Kosko-94] and statistical models [Craiger & Coovert-94] because they do not state equations nor suffer from some limitations commonly attributed to structural equation modelling techniques (i.e. problems in model identification). FCMs are suitable tools for modelling real situations (see, for example, [Kosko-94]). These environments could be modelled using either stochastic or deterministic techniques. But such

models are "hard to find, hard to solve and hard to run in real time" [Kosko-94]. FCMs can be used instead. "They facilitate the modelling of arbitrarily complex, dynamic and time-evolving phenomena and processes... FCMs are ideal for prototyping models, and for assessing global information of complex and dynamic social and behavioural processes" [Craiger & Coovert-94].

The following points summarise other, more specific aspects about the FCM approach that we have considered to define FBMs. The main differences between FCMs and FBMs are described in section 2.2.1.

a) Labels of edges in FCMs

The label of a FCM edge can be: a *positive sign* (to mean positive relation or change in the same direction), a *negative sign* (to mean negative relation or change in opposite directions) or, *null* (to mean no causal relationship).

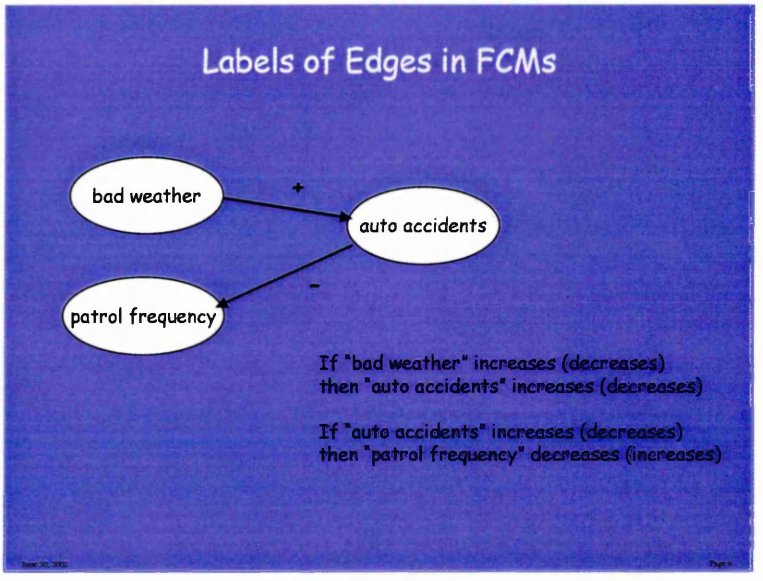


Figure 2.1: Labels of edges in FCMs.

These symbolic edges represent causal relationships between pairs of nodes. For example, Figure 2.1 describes two edges of a FCM found in [Kosko-93]. This map incorporates three nodes ("bad weather", "auto accidents" and "patrol frequency"), a positive edge (from "bad weather" to "auto accidents") and, a negative edge (from "auto accidents" to "patrol frequency") meaning that, if the "bad weather" increases (decreases) then "auto accidents" increases (decreases) and, if "auto accidents" increases (decreases) then "patrol frequency" decreases (increases).

Figure 2.2 represents a complete FCM found in [Kosko-86a].

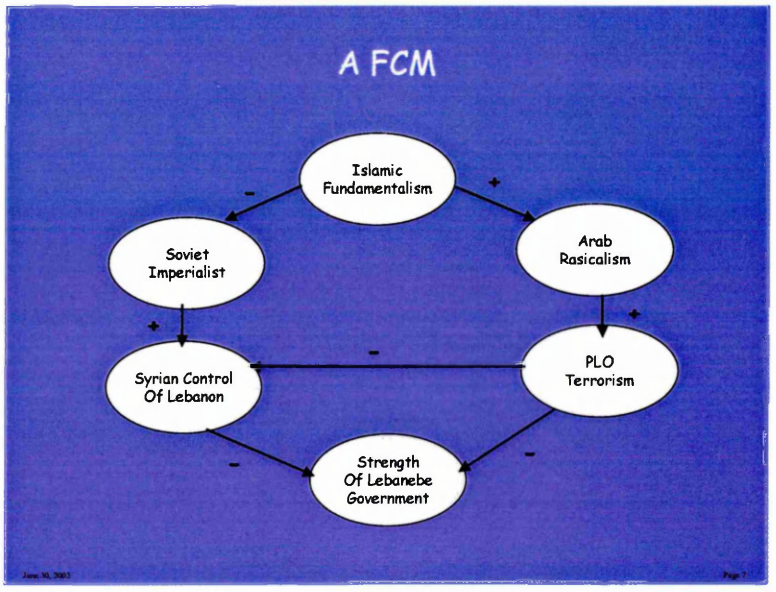


Figure 2.2: An example of a FCM found in [Kosko-86a].

b) Nodes and edges of a FCM "could be fuzzy"

In 1986, Kosko suggested that the nodes of a FCM could be fuzzy sets and that its edges could be implemented with fuzzy relations.

"Nodes stand for fuzzy sets or events that occur to some degree... directed edges stand for fuzzy rules ...". "The FCM system turns each picture into a matrix of fuzzy rule weights" [Kosko-94]. However, published examples of FCMs with fuzzy nodes, fuzzy edges or, fuzzy rule weights are difficult to find. An examination of the standard literature:

- modelling, analysing and combining expert knowledge [Taber-91],
- modelling social and psychological processes [Craiger & Coovert-94] and,
- modelling virtual worlds with Adaptive FCMs [Kosko-94; Dickerson & Kosko-94]

only reveals relatively simple architectures based on crisp strategies.

The *nodes and edges of a* FBM *can be fuzzy*. Indeed, one of our ideas around the design of behaviour-oriented models using FBMs is to allow the use of *fuzzy nodes* and *fuzzy edges* (see section 2.2.1).

c) FCMs behave as dynamic systems

[Kosko-88] draws and uses FCMs as dynamic systems. This author exploits the dynamic behaviour of FCMs to search for hidden patterns of the edges of these maps. He presents input vectors to the FCMs and let these structures evolve until they converge to a hidden pattern that can be either:

- a *fixed point* (a recurring pattern; a stable state with length = 1 from which the system provides a stable outcome) or,
- an *attractor cycle* (a sequence of patterns, a stable state with a length > 1 in which the system remains until something was changed or, it was provided with a new input vector) or,
- a *random state* (an unstable equilibrium from which the system cannot give any stable outcome).

The basic threshold FCMs have node strings expressed as (see Figure 2.3)

$$C(t_n) = (c_1(t_n), \ldots, c_N(t_n)) \qquad \text{(Equ. 2.1)}$$

that change according to the following law (see [Kosko-88] for a complete explanation on this equation).

$$c_i(t_n + 1) = S\sum_{k=1}^{N} e_{ki}(t_n) * c_k(t_n) \qquad \text{(Equ. 2.2)}$$

The output equilibrium of a FCM is an answer to a causal "what-if" question. FCMs store a set of rules "if $C(0)$ then equilibrium A" and recall patterns as they equilibrate. What if an initial node string occurs? For example, if the FCM given in Figure 2.3 is presented with an initial node string $C(0) = (0, 0, 0, 1, 0)$, its output equilibrium is the answer to "what if..." and could be a four-step limit cycle *C1 -> C2 -> C3 -> C4* resulting from the sequence *C2 = C1*E, C3 = C2*E, C4 = C3*E* and, *C1 = C4*E.*

d) Inference equations, connection matrices and augmented FCMs

FCMs are "graphical means of representing directional influences among concepts (variables)" which implications can be calculated simply via matrix algebra and used to evaluate time-evolving effects of its components [Craiger & Coovert-94]; like Fuzzy Associative Memory Systems [Kosko-87]. These structures can be transformed into connection matrices whose components are the numerical values of the causal relationships of its nodes.

Figure 2.3 shows an example of a FCM found in [Craiger & Coovert-94]. The inference equation of this FCM is expressed as:

$$c(i+1) = O(i) = \sum_{n=1}^{i}\sum_{m=1}^{k} c_m * E_{mn} \qquad \text{(Equ. 2.3)}$$

where E represents the connection matrix and C is the nodes string of the map.

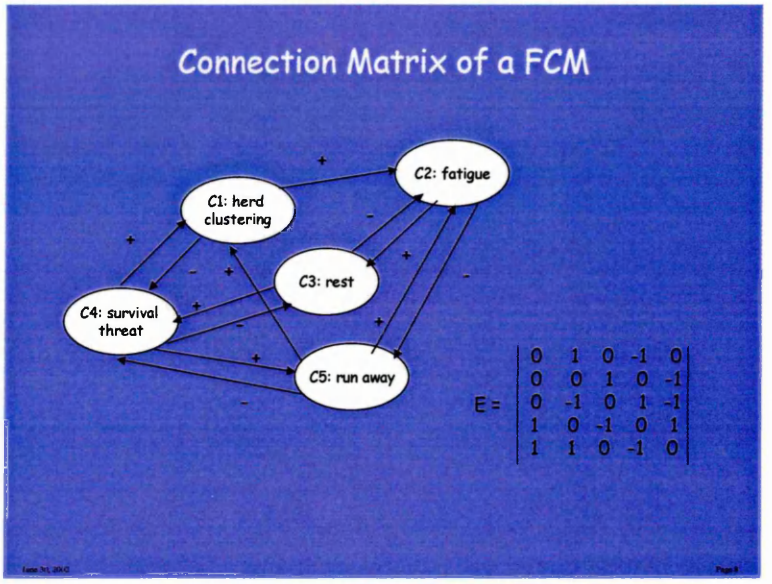


Figure 2.3: Example of a FCM; Edge Matrix and Nodes String [Kosko-94].

"A FCM weights and adds FCM matrices to combine any number of causal pictures" ... "we can also let a FCM node control its own FCM to give nested FCM in a hierarchy of virtual worlds" ... "This is the kind of things we can model with FCMs" [Kosko-94].

An Augmented FCM (a combination of FCMs) is converted into an edge matrix form once the links between FCM components have been weighted using relative values (*wi*). This is illustrated in Figure 2.4. In this case, the FCM structure has a global connection matrix *Fw* of the form:

$$F_w = \sum_{i=1}^{N} F_i * w_i ; N = 3 \qquad \text{(Equ. 2.4)}$$

where N represents the number of FCMs and *wi* the relative weighted connections between the N FCMs.

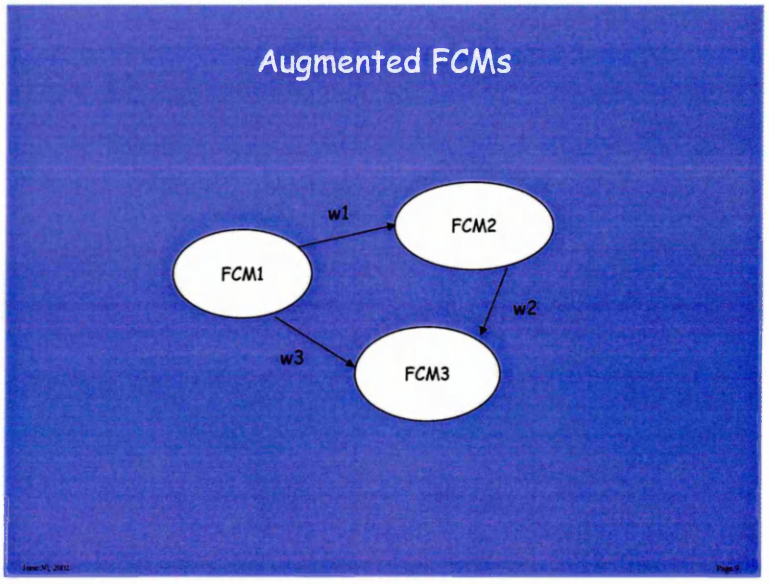


Figure 2.4: Example of an Augmented FCM [Taber-91].

f) Adaptive and learning features

Perhaps the most interesting application of FCMs is presented in [Kosko-94] where the adaptive version of the system is used to *model virtual worlds*. "The FCM itself acts as a non-linear dynamical system. Like a neural net it maps inputs to output equilibrium states" [Kosko-94].

An Adaptive FCM (AFCM) changes its causal connections in time. The causal web of this system learns (i.e. changes its edge strengths) from the data that is presented to it and, the combination of AFCMs (augmented AFCMs) is used to mean the design of large AFCM structures that can also learn and behave as "true adaptive fuzzy systems". These AFCMs can change via processes of excitation and combination. They can also learn (as any ANN) by applying neural learning laws to change their edges and limit cycles. These learning algorithms can be used to create and tune large FCMs. "Everytime you change a FCM, you learn in some way since learning is change" [Kandel-87]. "Neural learning laws change the causal rules and the limit cycles" [Kosko-94].

2.1.2. Behaviour-Based Systems

The concept of FBMs has also been influenced by non-hierarchical (flexible) behaviour decompositions [Brooks-86; Mataric-94a; Mataric-94b] and BB Networks [Maes-89b; Tyrrell-92], namely, to recognise the fact that artificial behaviour-producing systems should be built on highly distributed and richly connected networks.

In section 2.2.2, we describe the similarities and differences between FBMs and these BB Systems (non-hierarchical behaviour systems and, BB Networks).

2.2. Main Features

A Fuzzy Behavioural Map (FBM) is a behaviour-oriented network. Its nodes represent *behaviours* and, its edges represent their *causal interactions*. It represents a tool that helps to model behaviour-oriented forms of intelligence i.e. the natural behaviours that the researchers of AAs observe and analyse.

The nodes (behaviours) of a FBM can fire to some degree. These have associated levels of activation that can vary on time of interaction of an agent with its environment. The performance of a FBM results from driving its causal interactions (its cause-effect flow) among the *levels of activation of the behaviours* it represents.

These behaviour-oriented maps do not state explicit equations about the competences of the autonomous creatures. They do not represent alternative techniques to the current algorithmic, ANN-based or dynamic approaches to design behaviour control mechanisms (described in section 1.2.2.5). FBMs are suitable tools for modelling the behaviour-oriented structures before identifying and/or implementing any control mechanism.

As we shall see later in this dissertation, the fuzzy and dynamic capabilities of the FBMs allow the modelling of rather complex behaviour-based situations to later recognize the best way of designing and implementing

the required BB System. For example, the designer can use a BB Network approach such as an Action Selection Mechanism to develop a behaviour-based control architecture. But complex behaviours and their relationships cannot be easily modelled using these approaches (as discussed in section 1.4.3). FBMs can be used instead. *These maps facilitate the modelling of dynamic, time-evolving behaviour-oriented phenomena and related processes.* In other words, FBMs can be used as tools for prototyping behaviour-producing modules before implementing these. They introduce dynamical aspects and fuzzy frameworks that help on the modelling of autonomous agents' competences (and related internal mechanisms).

The following sections concentrate on these *modelling capabilities* of simple and more complex FBMs while presenting some examples of these structures. Then, in chapters 3 and 4, we focus on *the engineering problem of FBMs.* There we decompose the problem of controlling the behaviours of the creature into a number of development steps that facilitates the design, implementation and evaluation of the resulting BB System. Further, we combine a number of behaviour-oriented processes in such a way that optimal, global control can be provided to make the creature performing the required functionality more efficiently.

2.2.1. FBMs and FCMs

Simple FBMs draw pictures about the behaviours of AAs and how these can be causally related. They are *FCM-like networks* which symbolic edges show possible causal interactions among the degrees of firing of their nodes (behaviours).

More complex FBMs incorporate feedback cause-effect connections that make FBMs behaving as dynamical systems. But the scope of these dynamic features of FBMs is very different to that of FCMs.

The convergence (or stability feature) of FCMs has been used to evaluate psychological processes. For example, in [Craiger & Coovert-94], the FCM nomenclature is used to evaluate effects of a specific model by introducing a number of bit vectors with elements representing states of one of the concepts of the model.

This author applies an appropriate inference equation so that the overall output vector (global state of the model) changes until the map repeats a hidden pattern (equilibrium after few iterations) and the corresponding output vector can be interpreted. [Kosko-94] uses the same procedure (mapping input states to limit equilibrium) to give virtual worlds with new equilibrium behaviour (to reveal mode implications).

Complex FBMs help to design behaviour-based models instead of psychological processes or, virtual worlds. These maps can be used to evaluate global behaviour models by introducing a number of behaviour-oriented units with elements representing the levels of activation (or firing degrees) of these. We exploit the dynamic features of complex FBMs to model behaviour-oriented forms of intelligence so that the autonomous agent can perform stable behaviours (or, actions) from the evolution (time-dependent changes) of many possible behaviours and related mechanisms that causally affect each other.

Other, more specific similarities and differences between FBMs and FCMs are summarised below.

a) The edges of a FBM are labelled and weighted using qualitative and quantitative information

A FBM incorporates nodes and edges that help to model the behaviours of an AA. Here the edges are labelled to indicate either positive or negative cause-effect relations among the behaviour nodes.

The weighting factors of an edge of a FBM are numbers between –1 and 1. Simple FBMs have edges with weighting factors in {-1, 0, +1}. Everytime we connect two behaviours with one of these edges, we represent a *causal relationship* between their degrees of activation. Further, we can associate a *qualitative label* (a weighting word such as *little* or, *more or less*). Thus, FBMs provide quantitative and qualitative data about causal relationships of their behaviour nodes.

b) The nodes of a FBM can fire to some degree

The nodes (behaviours) of a FBM can fire to some degree (as FCM's nodes) and, in their most simple forms, they can be just *on* or *off* (as in Kosko's FCMs or, in deterministic ANN math structures).

Using simple FBMs, we set behaviour nodes and link them to others. Using large-scale FBMs, we model more complex and dynamic behaviour-oriented structures. Each behaviour node can fire on its own time scale and, each causal connection can have its own time scale too. Further, it is possible using non-linear mechanisms to design these maps so that both nodes and edges change accordingly (see chapter 4).

c) Nodes and edges of a FBM "can be fuzzy"

The structure of FBMs differs from Kosko's FCMs, mainly, in that the former can incorporate truly *fuzzy parameters*.

Nested FCMs seem to incorporate *fuzziness* by means of *mapping* processes. A nested FCM can be used to divide a concept (node) into number of sub-concepts (more nodes). For example, the "auto accidents" node represented in Figure 2.1 can be divided into three different sub-nodes: "large auto accidents", "small auto accidents" and "medium auto accidents". Following [Kosko-94], these sub-concepts form a Fuzzy System (or a set of fuzzy rules) that map input to outputs by linking fuzzy sets and, in practice, each sub-concept can map to different tactics so that new feedback loops are incorporated to the initial FCM.

By contrast, the degrees of activation of the behavioural nodes and the causal relations of FBMs can be designed using:

- *fuzzy numerical information* (degrees of membership functions of fuzzy sets) and,
- *fuzzy words* (linguistic variables and fuzzy edges)

to mean "dual, intrinsic characterisation of fuzzy sets and fuzzy systems" [Zadeh-94]; to introduce fuzzy functions as any Fuzzy System should do [Zadeh-72d; Zadeh-73a; Zadeh-73b; Zadeh-72d; Zadeh-74; Zadeh-75; Zadeh-83] (see also appendix A). In other words, FBMs do apply *truly fuzzy design parameters.*

The levels of activation of the FBM nodes can be designed using membership functions and descriptive values of fuzzy sets. Here the use of membership functions means that the behaviour nodes of the FBM get any value between –1 and 1 (i.e. fuzzy granularity as described, for example, in [Zadeh-65; Zadeh-72a; Zadeh72b; Zadeh-72c; Zadeh-72d]). The linguistic variables that characterize the fuzzy sets help to design behaviour-oriented maps adding descriptive information about behaviours and their relationships. For example, we can design a FBM where some behaviour nodes are *completely activated* (i.e. it is possible observing these behaviours) whereas other competences (behaviours) are *somehow activated* (i.e. it is possible observing these behaviours very soon) or, *not activated at all* (i.e. these behaviours cannot be observed at all).

The edges of an FBM can be fuzzy sets too. It is also possible using the membership functions and descriptive values of fuzzy sets to design the edges of a FBM. Here the use of membership functions means that the cause-effect relationships between the behaviour nodes can increase (decrease) to some extent (i.e. the edges can also get numbers between –1 and 1). The linguistic variables, on the other hand, help to design the FBMs so that, for example, some behaviour nodes affect the degree of activation of others *very little*, *quite much*, *usually* or, *very often.*

Further, we can implement FBMs using fuzzy control parameters. In fact it is possible mapping a FBM onto a Fuzzy Logic Controller (or any other Fuzzy System). We can convert the fuzzy relations that the FBM represents onto *fuzzy control rules* and manipulate these using some *fuzzy inference mechanisms.* The fuzzy control rules help to control the behaviours of the agent. The fuzzy inference mechanisms serve to determine the level of activation of each behaviour node.

In section 2.3, we present an example of a FBM that incorporates fuzzy sets (membership functions and linguistic variables) to design both nodes and edges. In chapter 3, we describe how to design a FBM and map this onto a Fuzzy Logic Controller, including complete definitions of fuzzy sets, fuzzy relations, fuzzy control rules and, fuzzy inference mechanisms. Doing so, we model, design and control the activation of an avoidance behaviour of a simulated robot. We convert all the fuzzy nodes/edges of a FBM onto fuzzy control rules and use several fuzzy inference mechanisms (and other fuzzy control design parameters) to implement the behaviour map (FBM). Further, in chapter 3, we emphasise some potential benefits in the use of the fuzzy approach to develop BB Systems, mainly, the flexibility of the fuzzy aggregation methods, fuzzy granularity and other related aspects.

d) Simple formal frameworks to manipulate FBMs

Simple and dynamic FBMs can be manipulated using some of the formal frameworks (tools) that Kosko uses with FCMs. For example, we can calculate the behaviour implications of a FBM using the *matrix algebra* described in [Craiger & Coovert-94] and [Kosko-87]. We can use the equation Equ. 2.1 to express behaviour strings (or, behaviour vectors introduced in section 2.3.1) of a FBM and, Equ 2.2 to represent how these behaviours change over time of interaction of an agent with its environment (see example in section 2.3).

The convergence of this law means that we have found a *stable behaviour solution*. This helps to evaluate behaviour-oriented processes. We present the FBM network with behaviour vectors representing the level of activation of some competences of an autonomous agent. The inference equation (Equ. 2.2) helps us to evaluate how the global behaviour solution (the behaviours that the agent can display) changes until the map repeats some hidden pattern. This is very useful for modelling the behaviour-oriented systems for AAs, as described in chapter 1.

Each FBM can be transformed into a connection matrix, as Figure 2.3 illustrates, with components meaning numerical values of causal relationships among the behaviour nodes of the map. This helps us to graphically

represent the influences among the behaviours nodes and, to easily calculate (using the equation Equ 2.3) time-dependent evolutions (changes) of some crisp behaviour activation values (0, 1 or, -1). However, as we shall see later in this section, these matrix components should correspond to *degrees of behaviour activation* by means of using membership functions of fuzzy sets.

Finally, we can manipulate FBMs as Augmented FCMs (Figure 2.4) to study and model some complex behaviour-oriented situations but, again, the matrix components (that, in this case, represent the influences among complete behaviour maps) should not be limited to some crisp numerical values. *Degrees of behaviour maps activation* should be used instead. The equation Equ 2.4 (expressing an inference equation for a collection of FBMs that interact with each other), for example, should be converted into truly fuzzy inference mechanisms. We use flexible fuzzy relations for manipulating large-scale FBMs rather than crisp weighted sums.

In section 2.3, we provide some examples of all these formal representations of FBMs. The use of fuzzy relations and fuzzy inference mechanisms with FBMs are widely described in chapter 4.

2.2.2. BB Aspects

The main behaviour-oriented features of FBMs can be summarised as follows.

a) The realisation of refined behaviour-oriented systems

Using FBMs, we search for *the realisation of refined behaviour-oriented systems* where it is possible (in latter stages of design), for example, identifying node strings with behaviour-oriented commands that will control specific components of a robot/agent (see chapter 4).

Here the concept *refinement* is used to identify lower design parameters (i.e. control actions) rather than linguistic variables or, fuzzy sets (as it corresponds to a fuzzy implementation technique like the FLC

described in chapter 3). This refinement helps on the definition of the behaviours selected from *first levels of abstraction*. For example, an avoid collision behaviour (a FBM node) can be refined down to movements and specific control actions over the motors of a mobile robot. This is, the avoidance behaviour of a mobile robot can be defined with control actions such as turn left or turn right that might control the left and right wheels of the agent.

This refinement process (that we associate to the design of FBMs) relates to the *fuzzy mapping techniques* firstly introduced by Lofti Zadeh in 1972. The fuzzy mapping applies to those complex scenarios that can be defined using fuzzy terms, as opposed to deterministically described. "For the control of physical systems with uncertainty, it may be desirable to have a robust control, robust in the sense that the controller is simple and gives guaranteed performance within the uncertain range. A first step in this direction is the introduction of fuzzy functions ..." [Zadeh-72d].

b) FBMs and Non-Hierarchical BB Systems

Following the investigations summarized in chapter 1, the non-hierarchical behaviour decompositions such as Brooks' Subsumption Architectures) help to design behaviour units as layers of control that couple sensing and acting capabilities of an artificial creature (i.e. low level design details). Further, these systems allow self-controlled competences that can be active and change their states according to inputs that the creature receives from its environment. At any instant of time, each behaviour unit can either change its internal state or, change the contents of some internal registries that hold input and output values for the agent to perform the required functionalities.

c) FBMs and BB Networks

The BB Networks facilitate the design of behaviour-oriented control schemes by introducing parallel and local operations. The Action Selection Mechanisms [Maes-89a; Maes-89b; Maes-90b; Beer *et al.*-90;

Tyrrell-92; Tyrrell-93; Tyrrell-94; Weiss-92; Kortenkamp & Chown-92; Blumberg *et al.*-96; Humphrys-96], for example, help to define behaviour units by mapping the competences of the creature into local, simple operations of some units that are connected to each other using a network structure. Further, most of these systems are implemented using Recurrent ANNs because, as discussed in chapter 1, these control structures have learning capabilities that help to automate the design processes.

FBMs also represent network structures that help to design behaviour units. However, these maps are not intended to map the competences of the creature into simple processing units. Each behaviour unit can represent a complex competence that has to be refined before any processing unit can be identified and implemented.

The learning capabilities of FBMs are discussed as possible extensions of the work presented in this dissertation (see chapter 6). These ideas are widely inspired by the learning features of Kosko's FCMs and Learning BB Networks.

Action Selection Mechanisms help to build BB Systems by identifying goals, sensor inputs and behaviour networks (e.g. Maes' Action Selection Mechanisms) for the agents to be able to select actions in diverse situations [Maes-89b]. These maps implement communication links from the goals and sensors of the agent to a behaviour network which activation spreads around once this has been activated from some external links. However, the literature of AAs seems to reflect that BB Networks lack feedback connections from the behaviour network to its external causes; these systems do not seem to implement truly agent-environment interactions; it is difficult to describe natural behaviour using their neural-network terms [Steels-94a].

By contrast, FBMs can be activated from external causes that fire/change the level of activation of one/all the levels of activation of its behaviour nodes. This dynamical process can effect the external inputs and, in this way, FBMs model *circular causality* that non-hierarchical BB Networks do not seem to implement. We describe this in more detail in chapter 4.

d) FBMs and Dynamical BBSs

The FBMs are closely related to these Dynamical BB Systems too. In fact, both types of systems emphasise the fact that the activities of an agent (behaviours, actions, goals, etc.) need interacting with each other while being modified by (part of) other possible agent-environment interaction processes.

The Dynamical BB Systems approach incorporates dynamic aspects into the description of the behaviour units [Smithers-94; Smithers-95; Steels-95; Tani-96a; Tani-96b; Thornton-96; Verschure & Pfeifer-92; Gallager & Beer-92; Neven *et al.*-96; Wilde-96; Pfeifer-96a].

Each behaviour unit of a Dynamical BBS is a dynamic element that forms part of an interaction space between the agents and their environments [Smithers-95; Steels-95]. Further, these dynamical units can cooperate with each other at the level of actions.

The behaviour nodes of a FBM are like dynamical units. Additionally, we can combine multiple FBMs so that these maps cooperate at the level 0f activating their behaviour units or, internal components (see chapter 4).

e) More general BB features of FBMs

A FBM represents something more than a hybrid (FCM, non-hierarchical, distributed and dynamical) system. An FBM is a behaviour-oriented system that can be used as a tool to address:

- the analysis and modelling of behaviour-oriented forms of intelligence (e.g. animal behaviours),
- the way these seem to be organised and displayed (performed) by complex organisms (refer to examples in section 2.3) and,
- the abstraction and description (through refinement processes described in chapter 4) of large-scale BB Systems.

The last feature is an extension of the work presented in this chapter. It represents the basis of the FBM Framework i.e. the formal relationships between the FBMs (designing tool) and the development of BB Systems (tasks and processes). In chapter 4, we present the basis of this framework to design and implement large-scale FBMs and, a case study on how to derive a BB System using this. As we shall see, this work serves to demonstrate that FBMs help in the tractability of some complex behavioural competences and related interaction processes. Because, as we discussed in chapter 1, the complexity issues of the behaviour-oriented approach refers not only to the uncertainties found in real world situations. There are also many problems to handle complex behaviour-oriented scenarios.

2.3. Examples

In the following sections, we introduce some examples of FBMs with varying degrees of complexity.

2.3.1. Simple FBMs

As we described above, simple FBMs is a FCM-like directed graph in which the nodes represent behaviours the designer is hoping to implement within an AA and, the edges are labelled to indicate either a positive or negative cause-effect relations.

In simple FBMs, a positive edge means a causal increase (change in the same direction) whereas a negative edge represents a causal decrease (change in opposite directions). That is, *positive* (*negative*) means that as the *cause* increases, the *effect* on the connected nodes increases (decreases). These cause-effect connections provide a view of the interaction between competencies (or, physical components) of the agent. But the idea is that the *strengths* of these interconnections should be expressed in Fuzzy Logic terms using, for example, fuzzy sets (membership functions and linguistic terms) or, fuzzy inference mechanisms.

Figure 2.5 shows a very simple example of a FBM. In this map, two physical components (a *sensor* and a *motor*) of an agent are interconnected using a positive edge so that, when the firing degree of one of them (the *sensor*) increases (decreases), the firing degree of the other component (the *motor*) increases (decreases) too.

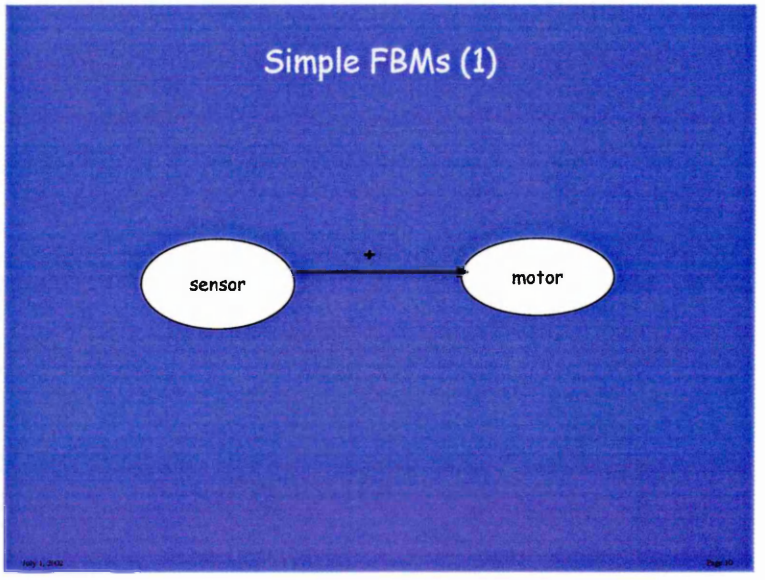


Figure 2.5: Simple FBM (1).

Another simple illustration is given in Figure 2.6. In this case, when the level of activation of a *right_sensor* (*left_sensor*) increases, the level of activation of the *left_motor* (*right_motor*) is also increased whilst that of the *right_motor* (*left_motor*) is decreased.

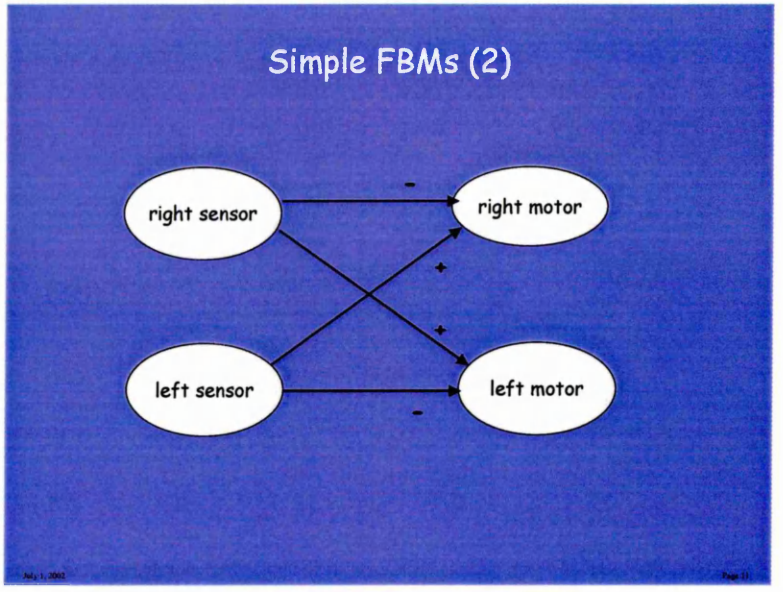


Figure 2.6: Simple FBM (2). As object is sensed on left or right, the resulting levels of activation will make the robot to turn right or left, respectively. The example is inspired by one of the Vehicles of [Braitenberg-84].

The *nodes* and *edges* of these simple FBMs can be *fuzzy sets*. Each behaviour node can be designed using membership functions of fuzzy sets so that the cause-effect connections (edges) become fuzzy relations.

For example, the following equation

$$\mu_{sensor}(A,k) * \mu_{connection}(B,k) \rightarrow \mu_{motor}(C,k) \qquad \text{(Equ. 2.5)}$$

can be applied to the FBM represented in Figure 2.5 to mean that the activity of the *motor* at time index k (membership function of a fuzzy set C that belongs to the domain of a linguistic variable called *speed*) is given by the fuzzy conjunction (*) between

- the level of activity of the *sensor* (membership function of the fuzzy set A that belongs to the domain of a the linguistic variable called *distance*) at the same time index k and,

- the strength of the *connection* (membership function of the fuzzy set B that belongs to the domain of a linguistic variable called *influence*).

Similarly, we can design the FBM represented in Figure 2.6 in terms of the following fuzzy relations:

$$(\mu_{right\_sensor}(A,k) * \mu_{connection}(B,k)) x (\mu_{left\_sensor}(C,k) * \mu_{connection}(D,k) \rightarrow \mu_{right\_motor}(E,k) \quad \text{(Equ. 2.6)}$$

$$(\mu_{right\_sensor}(F,k) * \mu_{connection}(G,k)) x (\mu_{left\_sensor}(H,k) * \mu_{connection}(I,k) \rightarrow \mu_{left\_motor}(E,k) \quad \text{(Equ. 2.7)}$$

where the levels of activities of the nodes *left_motor* and *right_motor* (at time index K) are given by the fuzzy implications of some fuzzy conjunctions between the fuzzy sets of connection links and, the levels of activities of the *right_sensor* and *left_sensor* nodes (see the definitions of fuzzy operators in appendix A). We come back to the definition and use of all these fuzzy sets and fuzzy operators in chapter 3.

2.3.2. More Complex FBMs

As we have describe it so far, a simple FBM seems to only provide us with a static view of the potential behaviours of an AA, where "potential" is used in the sense that not all behaviours need be activated at any one instant in time. However, a dynamic element can be introduced by observing the following two aspects. First, the use of Zadeh's Fuzzy Logic [Dubois & Prade-86; Foulloy-93; Lee-90; Hercock & Barner-96; McNeill & Thro-94; Wang & Loe-93; González *et al*-97] suggests that, at any t instant in time, all behaviours can be active to some degree and their causal connections can have different (fuzzy) strengths. In fact, the activation values of the behaviour units and the strengths of their connections can be formalised as time-dependent membership functions like $\mu(A,t)$ of some fuzzy set A which can be shaped in a number of ways (see examples of membership functions and fuzzy sets in appendix A).

Second, a more complex FBM incorporating recurrent feedback loops could be *switched on* in a similar fashion to Hopfield Nets, Bi-directional Associative Memory Systems or, any other RANNs [Bose & Liang-96]). Indeed, the behaviour activation patterns of the FBM can be propagated around this map until some stable state has been reached or, some other terminating condition (such as timing out or, maximum level of activation of behaviour/s) has been satisfied.

To illustrate more complex FBMs, an example is given in Figure 2.7.

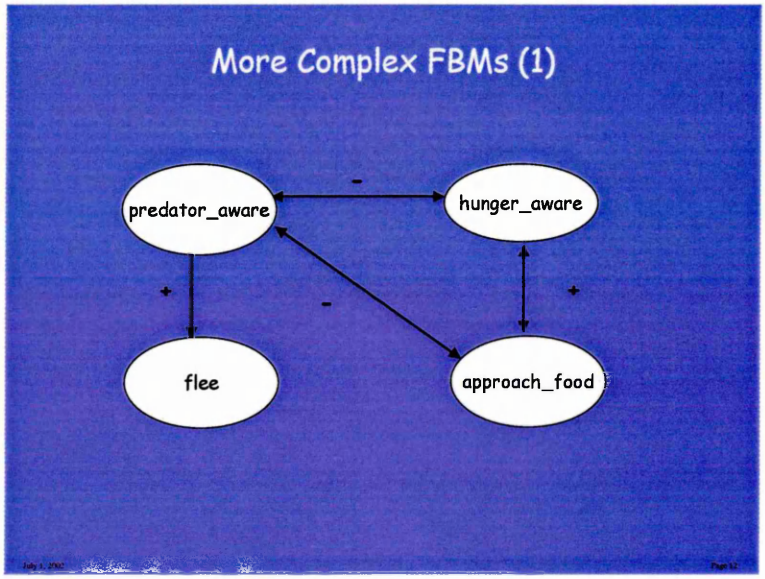


Figure 2.7: More Complex FBMs. Predator & Food.

This FBM represents the following relationships among behaviours:

- as the degree to which *awareness-of-hunger* increases, the tendency to *approach-food* increases,
- as *approach-food* is highly fired (the agent approaches food), *hunger* increases (through a sense of anticipation),
- as *hunger* increases (and food is approached), *awareness-of-predator* decreases and,

- as *awareness-of-predators* increases, the urge to *flee* rises, the *awareness-of-hunger* drops and the urge to *approach-food* falls.

Interpreting this diagram in Hopfield Net terms (and using these values with all weights equal), it is not difficult to see that if at some instant, the node *hunger_aware* is activated, then *approach_food* becomes activated. If the *predator_awareness* node is then switched on, the *flee* behaviour is the only behaviour which is active.

The fuzzy interpretation is more subtle since we can add both quantitative and qualitative data to define the map. The nodes and edges of the map can be defined in terms of membership functions of fuzzy sets so that fuzzy granularity is added to the interpretation provided above.

For example, if we represent the nodes with fuzzy sets and, *hunger_aware* is "highly" activated, the *approach_food* also activates in a "high" fashion since both nodes are connected using a positive edge. Using this fuzzy granularity (in nodes and edges) means that the influences between the nodes is fuzzy too. Therefore, the degree of activation of each node (e.g. *approach_food*) becomes affected by the degree of the communication among them.

An example of these fuzzy connections can be seen in Figure 2.8. Here the level of activity of a *left_motor* is "highly" influenced by the level of activity of a *left_sensor* whereas the same activity of the *left_motor* is "low" affected by the level of activity of a *right_sensor*. Both positive influences help an autonomous creature to display turn movements considering it is equipped with these simple mechanisms.

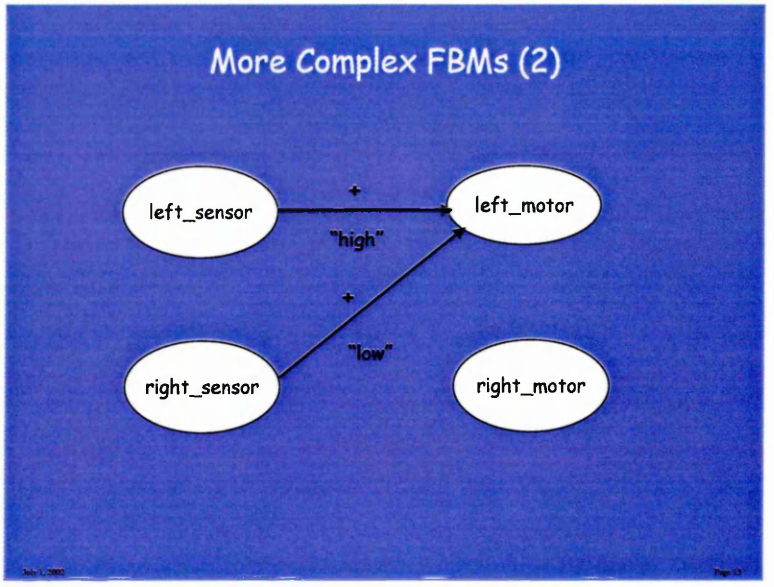


Figure 2.8: Turn Movements with FBMs.

With the three examples, we have been flexible in the interpretation of the term *behaviour*. In Figure 2.6 (simple FBM) and Figure 2.8 (turn movements), it has referred to the level of activation of physical resources of the AA (sensors and motors of a mobile robot). In Figure 2.7 (predator & food), a more abstract view has been adopted. We return to this point (*abstraction*) in chapter 4.

The examples provided in figures Figure 2.8 and Figure 2.9 illustrate different ways of interpreting the term *interaction* in the context of FBMs. The FBM represented in Figure 2.8 represents some physical, negative communication links between the physical resources of a mobile robot whereas the FBM in Figure 2.9 uses *co-operation* and *competition* to communicate the behaviours it represents. In this last FBM (Figure 2.9), the behaviours *b1*, *b2* and *b3* (that could be designed down into a number of nodes such as those represented in Figure 2.8) are connected in such a way that *b2* and *b3* compete with each other. There is a *high* negative influence between these nodes so that both (*b2* and *b3*) compete for being *highly activated*. By contrast, the behaviours *b1* and *b2* co-operate with each other. There is a *high* positive influence from one to another. This means both behaviours (*b1* and *b2*) will be able to *highly activate* each other at some instant in time.

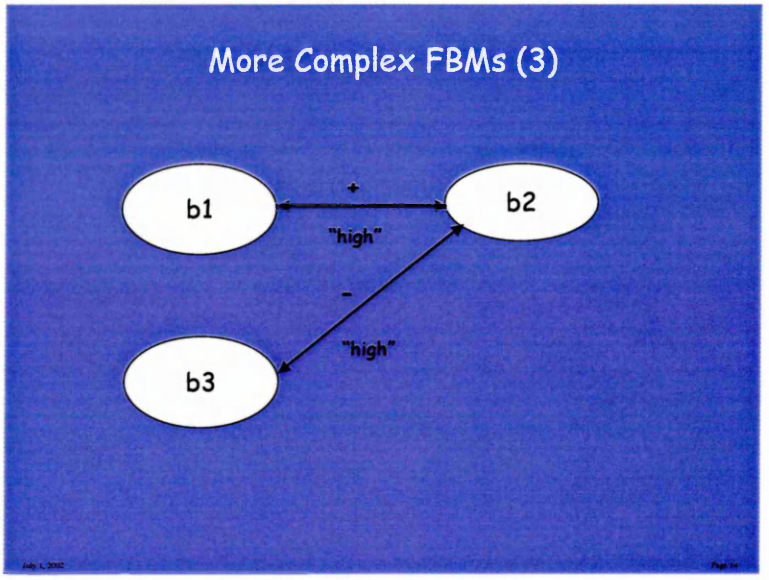


Figure 2.9: FBMs with Co-operation (edge > 0) and Competition (edge < 0) of Behaviours.

We can also achieve co-operation by means of combining FBMs. For example, Figure 2.10 represents an augmented FBM where the low-level FBMs either *highly activate* or, *lowly activate* one another. Further, we can represent *hierarchies of behaviour-producing modules*. Following [Tyrrell-92], for two nodes *b1* and *b2*, *b1* is the "boss" of the *b2* if the *b1* has direct causal influence on *b2*. Here the term "direct" is meant to imply that *b1* is directly above *b2* in the hierarchy and, the concept "causal influence" is meant to state that *b2* is to some extent dependent on the state of *b2*.

Figure 2.11 shows an example of a hierarchy of behaviour-oriented nodes inspired by [Tyrrell-92]. In this FBM, the top level nodes represent major "instincts", all the downward arrows (fuzzy edges of the FBM) represent "causal factors" (positive edges) and, the two-way arrows between two nodes at the same level represent "mutual suppression" (negative effects between the two nodes).

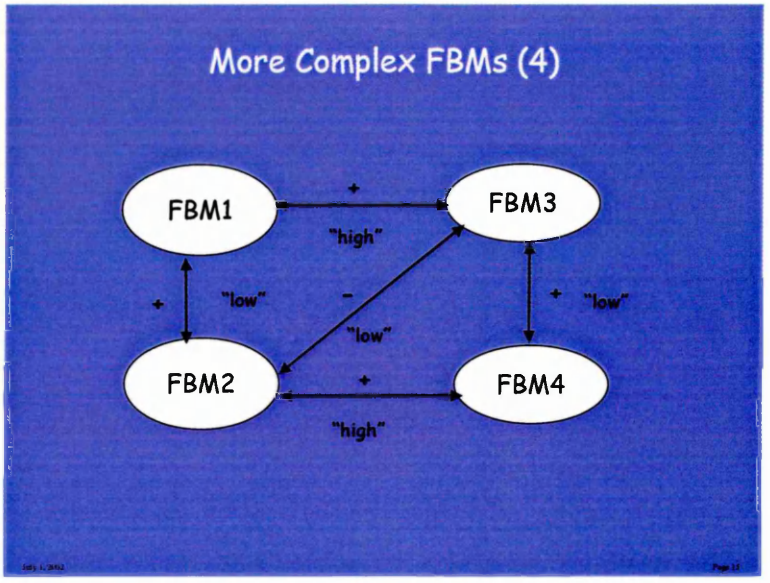


Figure 2.10: Co-operation among FBMs.

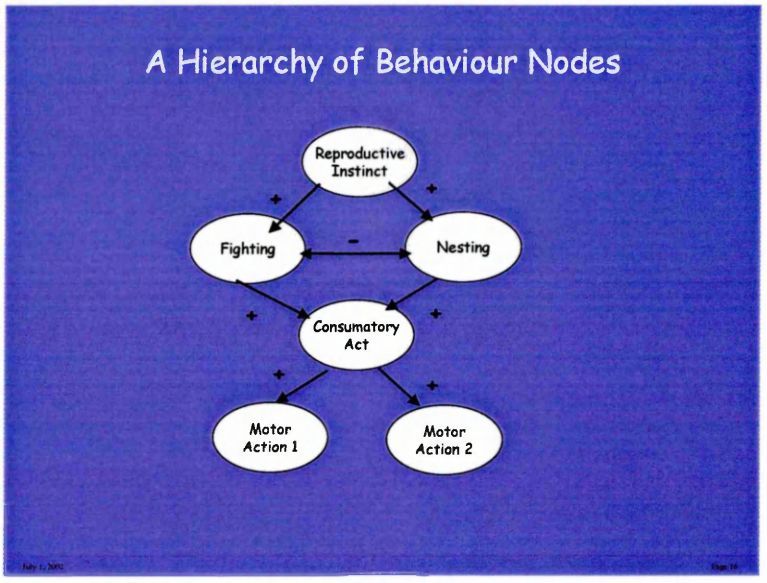


Figure 2.11: A Hierarchy of Behaviours inspired by [Tyrrell-92].

2.4. General Intuitions

In summary, the most general intuitions we have about FBMs are as follows:

a) A FBM is an hybrid tool that draws its inspiration from many sources

A FBM is a highly distributed network with a graph topology similar to that of ANNs and Cognitive Mapping Techniques. Further, a FBM is an hybrid BB System that draws inspiration from non-hierarchical BB Systems, BB Networks and Dynamical BB Systems.

b) FBMs represent flexible tools to design behaviour-based models

FBMs are flexible tools that help to design behaviour-oriented models. These maps allow multiple behavioural activity (all behavioural nodes can have the potential of being activated and can be fired "too some degree" at any instant in time), they allow flexible interpretations of behaviours and interactions and, their structure and performance can be addresses in different ways.

c) A FBM can operate dynamically

A FBM can be designed with a recurrent topology network (also used in Dynamic and Evolutionary approaches to the design of BBSs). The feedback connections of a FBM make it able to operate dynamically and, to converge towards asymptotic net-behaviours. The stability of this convergence depends on the symmetry of the connection links among its behavioural nodes. For example, a simple symmetrical FBM can converge to a stable state of equilibrium. It provides a single outcome of behavioural activities that can be taken for execution in combination, if necessary, with other relative computations.

d) The graph structure of a FBM is built upon the use of "fuzzy" parameters

The behavioural nodes and external effectors of the FBM must be characterised with fuzzy sets. Their levels of activity can admit different sort of membership functions (e.g. triangular *μb1*) and linguistic values (e.g. "high", "low", "moderate"). The edges of these maps can be designed using fuzzy sets. These can also be described using both quantitative (membership functions) and qualitative (linguistic values) parameters. Finally, the whole map can be converted into a number of fuzzy rules driven by a fuzzy inference mechanism to determine the levels of activity of all the nodes at any particular instant in time. This last feature can be accomplished when the FBM is map onto a Fuzzy Logic Controller (see sections 3.4 and 3.5).

e) FBMs are interaction-based models of activity that can run in asynchronous or parallel mode

The activity of a FBM is the result of number of interaction processes (e.g. environment-behaviour, behaviour-behaviour, behaviour-resource, etc) that can run in asynchronous or parallel mode depending on the (overall) model of dynamics that is selected by the designer. This model of activity strongly differs from the centralised model of activity of classical SMPA systems or, the goal-directed activity of Action Selection Mechanisms and other approaches to the design of BBSs.

f) FBMs can help on the design of complex BB Systems

They can be used as tools for capturing and modelling behavioural dynamics (behaviour producing nodes and their internal/external interactions) at different levels of abstraction (and through different stages of refinement) and, in combination with fuzzy formal frameworks (using fuzzy aggregations and other fuzzy logic methods). This final feature of the FBMs is outlined in more detail in chapter 4.

Chapter 3

The Development of a FBM for a Mobile Robot

Abstract

In this chapter we describe our initial research work around the design and implementation of behaviour-producing modules using simple FBMs and FLCs.

More particularly, the following sections provide a practical example on how to design the avoidance behaviour of a mobile robot called Khepera using a simple FBM as well as the implementation of this behaviour-oriented graph using a FLC architecture. Further, we summarise some interesting test results around the use of different inference mechanisms to activate the FBM. All this represents an extension of the work presented at the International Workshop on Advanced Robotics and Intelligent Machines [González *et al*-97].

3.1. Mobile Robots and Control Systems

The physical architecture of a *mobile robot* is usually equipped with three types of physical components:

- *sensors* (e.g. infra red or IR, ultrasound, vision, etc.) which allow the artifact to measure some external conditions of its environment,
- *motors* (e.g. wheels) to move around this environment and,
- a scheme of *sensorimotor connections* responsible for controlling the performance (e.g. motion) of the whole (artificial) structure.

Figure 3.1 shows the mobile robot Khepera [Michel-96] as an example of the physical set-up described above. This is arranged with two IR proximity sensors (6 in front and 2 on back) and two motors (each driving a wheel).

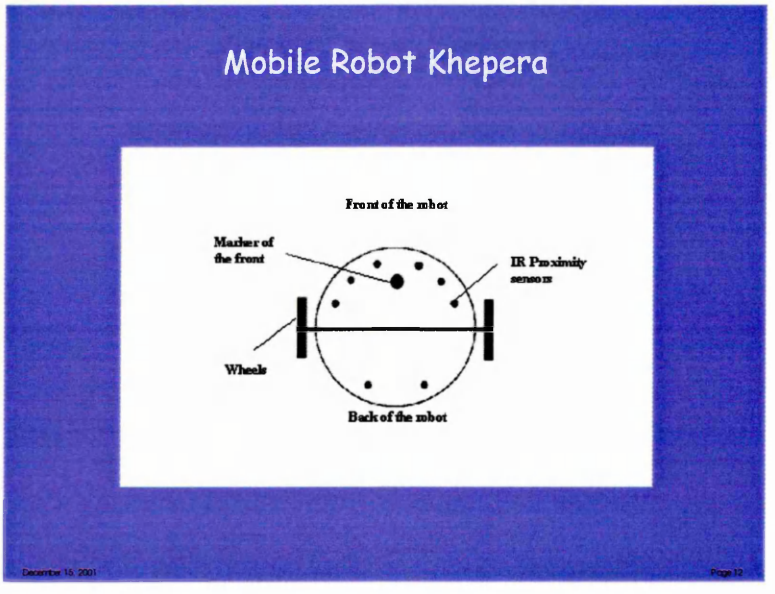


Figure 3.1: A Top View of the Mobile Robot Khepera.

In our research studies, we have used a simulated version of this robot. The main components of this *Khepera Simulator* are outlined in section 3.5.

As described in section 1.1.2, research in Mobile Robotics regarding the *controller component* of a mobile robot can be classified into two main approaches to the sensorimotor connections involved: the "sense-think-act" Robotics and the Autonomous Robotics approach. The first (classical) controllers corresponded to a *functional decomposition* of the sensorimotor connections in terms of perception, planning and execution units. These architectures attempted to accurately map, at any particular time, the location and direction of the robot in its environment [Keller *et al*-92] by reference to an internal, explicit representation (model) of the environment inhabited by the robot. In contrast, the task-achieving approach of Brooks (Autonomous Robotics) leads to the design of mobile robots according to a *layered behavioural decomposition.*

This recent research synthesising and analysing behavioural blocks into control architectures is also being influenced by disciplines such as Ethology and Neurology. Indeed, the required functionality of the mobile robots appears to be very similar to conduct what is observed in the animal kingdom [Goode *et al*-94; Mataric-95b; Roitblat-94; Scutt-94; Schmajuk & Blair-92; Schmajuk-94]. However, although the literature reflects successful applications of the ANN systems in the design of simple behavioural competencies, it has shown some limitations related to those of providing high level behaviours through processes of interaction between the robot and (possible) unpredictable situations of the environment.

To address these difficulties encountered in the use of ANNs, some authors have considered the possibility of applying Hybrid Fuzzy/Neural Systems (see, for example, [Keller *et al*-92; Takagi *et al*-92; Narazaki *et al*-92]) or heterogeneous ANN Systems that include computations driven by *fuzzy logic operations* (fuzzy logic inference), in addition to some special network propagation algorithms.

In our initial investigations *defining and using FBMs* (beginning with basic behavioural competencies), we have considered *Fuzzy Logic Control* (or FLC, firstly conceived by [Zadeh-72a]) as an effective technique for implementing these behaviour-oriented maps for the reasons that:

- The derivation of a *control policy* within a FLC (i.e. a set of fuzzy control rules) appears to be simpler to set up and quicker to tune than a traditional control architecture or an ANN.

- The *fuzzy weighting factors* applied in the inference mechanism of a FLC (through the contribution of the whole set of fuzzy control rules) provide *dynamical processes* that strongly differ from the single matching typically used in non-fuzzy rule base or, the static operations performed by FANNs once these have been trained.

- The FLC structures have a potential capability to *react to disturbed input data* using both fuzzy partitions and smooth transitions given by the membership functions of the fuzzy sets it manipulates.

The following sections describes these, our evaluations of Fuzzy Logic techniques while analysing the avoidance behaviour of the Khepera robot. Then we go on to report the design of the Collision Avoidance FBM, the implementation of the FLC architecture and, our experimental results.

3.2. Perception, Motion and Collision Avoidance

One of the benchmark areas of BB research is *collision avoidance*. Indeed, several authors have considered obstacle avoidance as a basic behaviour to be accomplished by a mobile robot. For instance, in [Brooks-86], avoidance of objects corresponds to one of the lowest levels of competencies desired for an autonomous agent. [Mataric-94a] has also proposed a set of basic behaviours consisting of avoidance, safe-wandering, aggregation, dispersion and homing (as observed in many species of the animal kingdom). [Cliff *et al*-93a]

maintains that a fundamental behavioural competence is the ability to move and navigate in changing surroundings.

We have analysed the following *linguistic strategy* (close in principle to one used for vehicles by [Braitenberg-84]) that attempts to control a mobile robot as to avoid collisions with obstacles while moving around its environment.

REPEAT,

IF no obstacle is perceived, THEN move forwards

ELSE IF obstacle is perceived on the right, THEN turn left

ELSE IF obstacle is perceived on the left, THEN turn right

Although a controller built on this navigation routine would make a robot turn left or right according to the presence of an obstacle on its right or left side, respectively, it takes into account neither *how strongly the object is perceived* nor the speed with which the robot is moving. Additionally, it can be demonstrated that this controller does not assist the *robot's decision making process* when it simultaneously perceives obstacles on both its left and right sides. Indeed, in simulation, it is seen that the mobile robot gets stuck in the corners of a room and stops moving within slim corridors.

We have therefore, modified this basic strategy to allow the robot to, firstly, *gradually change its path* as the obstacle's presence is felt more strongly and, secondly, to take into account the speed of the robot relative to this *perception of objects.*

3.3. The Collision Avoidance FBM

Figure 3.2 shows the Collision Avoidance FBM that we obtained as a result from the previous analysis. This map incorporates three nodes: *perceiving_obstacles*, *varying_speed* and *moving_around* that causally influence one another in such a way that:

- as the degree to which the robot *perceives an obstacle* increases (decreases), the *speed variation* of its motors increases (decreases) and,
- as the degree to which the robot *moves around* within its environment increases (decreases), the *speed variation* of its motors increases (decreases) too.

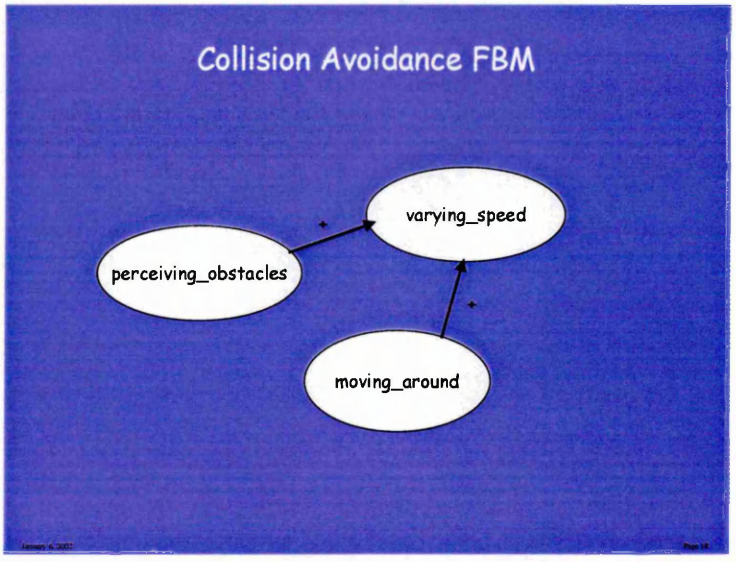


Figure 3.2: The Collision Avoidance FBM.

One of the reasons for which we have introduced *fuzzy sets* and *fuzzy relations* to design this FBM concerns their form of representing information. In general, fuzzy sets and fuzzy relations (see appendix A) operate at a higher (set) theoretic level than the purely numeric deterministic frameworks [Dubois & Prade-83]. The fuzzy framework is based on degrees (membership values) to which an element or condition may fit into a fuzzy set or a fuzzy relation. Further, this *fuzzy granularity* reflects in the ability of Fuzzy Logic Control techniques to handle the uncertain aspects of mechanisms such as:

- *perception* (internal reconstruction of visual signals related to speed motion) or,
- *robot's movement* (variation of motor speed dependent on sensor signals)

that can be achieved using, for example, fuzzy sets for representing

- various levels of perception, such as *obstacle lightly perceived* or, *obstacle moderately perceived* and,
- variation in motor speed in terms of fuzzy sets like *very fast*, *slow*, etc.

In practice, we have built the nodes of the Collision Avoidance FBM using *fuzzy sensor signals* and *fuzzy motor speed values* of Khepera robot. More particularly, we have mapped the *perceiving_obstacles* node of this simple FBM onto a set of fuzzy control rules using appropriate fuzzy relations among fuzzified sensor signals. That is, we have *refined* the *perceiving_obstacles* behaviour unit of the FBM so that it can be defined in terms of the sensor signals that the robot can receive. Then we have processed these behaviour control rules as a *Sub-Rule Base of an FLC architecture* using appropriate fuzzy relations of fuzzified sensor signals called *R1*. Similarly, we have selected some fuzzy sets of crisp speed motor values to design the *moving_around* node of the Collision Avoidance FBM and, fuzzy sets associated to speed control actions to design the *varying_speed* node. Finally, we have mapped the overall *collision avoidance behaviour strategy* that correspond to the FBM (fuzzy implications of *varying_speed* fuzzy sets based on all the possible fuzzy relations among the *perceiving_obstacles* and *moving_around* fuzzy sets) onto another set of fuzzy control

rules and processed these as another *Sub-Rule Base* (of the same FLC architecture) called *R2*. In other words, we have designed the Collision Avoidance FBM by means of *prototyping fuzzy control rules of a FLC architecture*. This avoidance strategy is widely described in section 3.5.

Figure 3.3 shows two examples of the levels of perception that we have considered for the implementation of the Collision Avoidance FBM: *obstacle highly perceived* and *obstacle moderately perceived*. Both fuzzy relations are related to the speed of the robot in the following way:

- *medium distance* (*d*) to an obstacle associated with *slow speed* (Figure 3.3(a)) builds an *obstacle highly perceived* within the internal scheme of the robot,

- the same proximity value (*d*) associated to a certain perturbation (Δd) due to a *fast speed* movement (Figure 3.3(b)) produces a level of perception equal to *obstacle moderately perceived*.

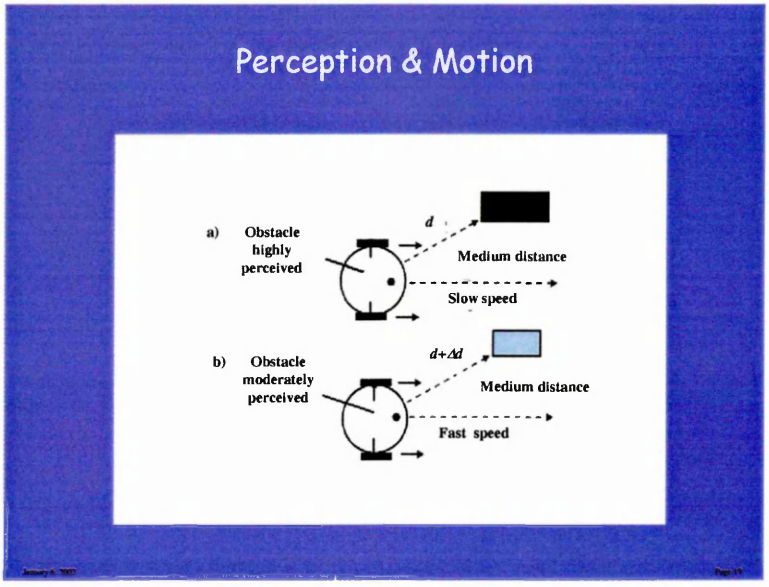


Figure 3.3: Perception and Motion. (a) Slow movements. (b) Fast movements. (a) And (b) for medium distance.

Using the simple example shown in Figure 3.3, we have also attempted to model some mechanisms by which we (humans) can perceive dimensions of objects [McEachern-93]. Certainly, when we are *moving slowly* (Figure 3.3(a)), the boundaries of any object placed at a medium distance from our current position appear clear to our *visual perception*. However, when we *run* (Figure 3.3(b)), what we perceive *is a distortion of what is out there*, an image smaller than the real figure. Thus, due to the well-known Physical Principles of Relativity, the immediate effect of a *fast movement* is that an obstacle appears to be at a further distance than in the case of *slow movements* or, more generally, that motor speed affects the level of perception of an object.

In this chapter, we have introduced the notion of perception in order to understand and exploit some ideas of *abstract FBMs* and the implementation of these maps using the FLC architecture that we describe in section 3.4. However, it should be noted that the use of this concept raises some interesting issues, which have yet to be fully explored, in relationship to the general ethos of BB Robotics, since it clearly introduces some elements of world modelling into the control scheme.

3.4. The FLC Architecture

Fuzzy Logic has provided Control Systems Theories with *a broader methodology to cope with dynamic and complex processes* [Zadeh-83; Zadeh-89; Foulloy-93; Foulloy-93; Lee-90; Zadeh-83; Zadeh-89; Zadeh-92].

One of the main advantages of using the Fuzzy Logic Control approach (see appendix B) is that no explicit mathematical model of a system under control is necessary for implementing a *Fuzzy Logic Controller* (FLC). Fuzzy Logic Control helps to define non-linear control systems through a *development (implementation) process* that might be difficult to achieve by standard methods of Classical Control techniques.

In effect, the implementation of a FLC architecture emphasises the fact that *fuzzy control rules should be seen as an implementation technique of non-linear transition functions* that can determine *effective (successful) control actions* even in the case of complex external conditions.

The implementation of a classical controller usually starts with the specification of a time-dependent variable that has to reach a desired value. This time-dependent variable is called *output variable* and its value is controlled with a *control variable* that can be adjusted. Then, the classical controller is developed so that the output variable is determined from the measurement of *input values* and appropriate time-dependent changes on the control variable. But classical control is not only restricted to the measurement of output variables and the change of these in time steps.

Sometimes, classical control deals with *dynamical processes* that can determine the change of the output variables instead of computing the control values. Further, the solution of a control problem can be generalised in the form of a control function that maps an input space to an adequate control space using, for example, *differential equations* so that the control solution is the control values set that makes these equations converge within an acceptable time towards cero.

The main problem of these classical control methods is that, sometimes, it is impossible finding *an accurate mathematical structure* to control the process at hand (e.g. a set of accurate differential equations). Indeed, the control problem might require a deep physical understanding of the sub-processes involved (e.g. the movements of objects which influence has to be controlled) as well as a deep mathematical knowledge to be able to solve the formalisms used to find the control solutions (e.g. approximation of differential equations).

FLCs seem to help on solving these problems found with classical control techniques [Zadeh-72a; Zadeh-72d; Zadeh-94]. There is not need of a deep mathematical knowledge, nor a deep physical understanding of the system and situations under control.

FLCs can be characterised with either *table-based control* or, *non-linear transition functions*. The *semantic foundation* of these architectures represents one of the most innovative aspects. Further, the derivation of the fuzzy control policies (fuzzy control rules described in section 3.4.3.3) appear to be more easily solved and tuned than the crisp ones.

The structure and implementation parameters that we have identified for building a *table-based FLC* are as follows (see also [Mamdani-76; Dubois & Prade-83; Lee I-II-90; Foulloy-93; Wang & Loe-93]).

3.4.1. Structure and Operations

Figure 3.4 shows the basic configuration of the table-based FLC architecture. The overall scheme contains a generic *controlled process* and, five functional blocks (controller components): a *Fuzzification Interface*, a *Data Base*, a *Rule Base*, a *Decision Making Logic Unit* and, a *Defuzzification Interface*.

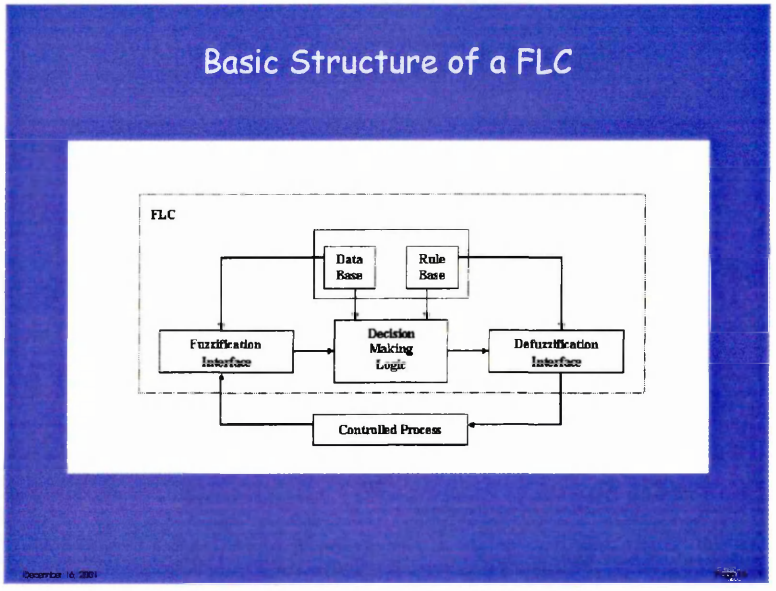


Figure 3.4: Basic Structure of a FLC.

The controller components compute the action of the controlled process. The Fuzzification Interface (input phase) transforms the numerical level of the process being controlled by the FLC into a conceptual level of fuzzy sets. Then all the *mappings* (computations) involving the Rule Base and the Decision Making Logic (inference unit) are completed at the level of the fuzzy sets being active. Final actions inferred by the inference unit are converted by the Defuzzification Interface (output phase) to the numerical level of the controlled process.

3.4.2. Components and Implementation Parameters

Table 3.1 shows the implementation parameters (I.Ps) that we have identified as the main functional blocks of this FLC (see also [Lee I-II-90] and, appendix B).

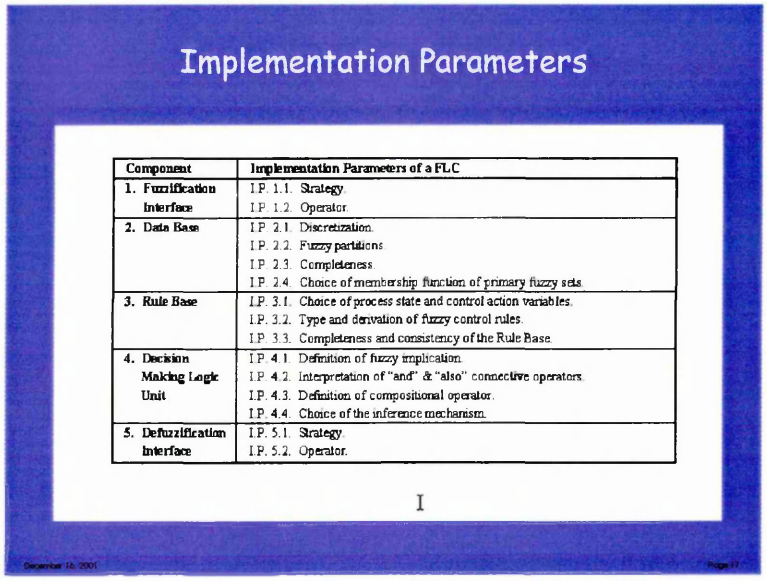


Table 3.1: Table of Components and Implementation Parameters of a FLC.

3.4.2.1. Component1: Fuzzification Interface

Fuzzification involves a subjective evaluation that transforms a measurement of crisp data into a fuzzy set of values. It is defined as a mapping from an observed input space to the fuzzy sets required in a Fuzzy System.

The following parameters need to be addressed in the implementation of a Fuzzification Interface.

I.P. 1.1. Strategy

The analysis of the controlled process constitutes the basis for the *input interface* (mapping) developed by this functional unit. From the observed and, usually uncertain and complex inputs, a basic fuzzification strategy involves:

- specifying the universes of discourse that characterise the controlled process,
- determining input variables and their crisp (numerical) values,
- articulating each input variable into its corresponding universe of discourse and,
- converting the numerical data of each input variable into a suitable set of fuzzy sets which can define their related linguistic variables.

I.P. 1.2. Operator

The selection of the *fuzzification operator* may depend on the type of fuzzy sets (as outlined in appendix A) that have been chosen for the FLC. Most of fuzzy control applications map the input space into fuzzy numbers, or numerical fuzzy sets. However, other examples of FLC techniques present the manipulation of *functional fuzzy sets*. In section 3.5, we provide an example of this last type.

3.4.2.2. Component2: Data Base

The Data Base of a FLC represents some knowledge of the controlled process in the form of *fuzzy data that is used in the fuzzy control policy*. In the next sections, we address the main parameters required in the definition of the fuzzy control policy within the Rule Base (Component 3) as well as the manipulation of this policy and the fuzzy data within the Decision Making Logic Unit (Component 4).

The following implementation parameters I.P. 2.1 to I.P. 2.4 summarise the role of the Data Base.

I.P. 2.1. Discretization

This depends on the type of *universe of discourses* obtained from I.P. 1.1. It solely affects to continuous input data. In the case that we describe, the discretization process consists of arranging a certain scale mapping for labelling different segments as generic elements of a discrete space. Then fuzzy sets of the FLC are defined by assigning membership values to each generic element of the new *discrete space* (see appendix A).

I.P. 2.2. Fuzzy Partitions

There are two important aspects regarding the *fuzzy partitions* of a FLC.

- First, the selection of the primary fuzzy sets, needed for the definition of the terms of each linguistic variable (within the Data Base), and the fuzzy relations defining the control policy (within the Rule Base).

- Second, the design of the fuzzy sets (form, support and so on) chosen for each universe of discourse (e.g. fuzzy sets of *distance* supported on a universe of discourse $U=D$).

In practice, *a look up table of partitions* can be used in order to specify the distribution of the supports of the fuzzy sets that share values in an interval of crisp data.

Unfortunately, the procedure for finding an optimal partition of these distributions cannot be deterministic in the sense that several trials are usually necessary.

This chapter shows a limit function that we have developed in order to help in the implementation of fuzzy partitions. The formalisation of this function is explained in section 3.5.2.

I.P. 2.3. Completeness

After having specified the fuzzy partitions within the Data Base of the FLC, the completeness of this unit must be guaranteed for each universe of discourse [Lee I-90]. This means, the union of the supports of each set of terms must cover the corresponding interval (support) of crisp values. Hence, the fuzzy control algorithm applied in the Decision Making Logic (Component 4) can be said to infer a proper control policy for every step of the controlled process.

I.P. 2.4. Choice of Membership Function of Primary Fuzzy Sets

The fuzzy sets of the Data Base can be defined with either

- numerical membership function, or
- functional shapes such as triangular, trapezoidal, bell, etc.

A *numerical definition* is commonly represented as a set of numbers whose dimension depends on the degree of discretization (as mentioned in I.P. 2.1). A *functional definition* deals with the design of the fuzzy sets that allow the manipulation of some fuzzy arithmetic operations (based on the fuzzy logic operators summarised in appendix A) that are explained in the rest of this section.

3.4.2.3. Component3: Rule Base

This component consists of a number of *fuzzy control rules* often expressed by the following *linguistic description* (see also the fuzzy control rules outlined in appendix A):

IF (a set of conditions is satisfied) THEN (a set of consequences can be inferred) (Def. I.1)

This can also be reduced to the conditional statement

IF "antecedent/s" THEN "consequent/s" (Def. I.2)

where "antecedent/s" and "consequent/s" are associated with expressions (e.g. "x" is T("x")) which link a linguistic variable ("x") to one of its corresponding fuzzy sets (T("x")). Then *each antecedent and/or consequent of a fuzzy control rule is defined as a fuzzy relation* (see appendix A) and, additionally, some symbolic sentence connectives such as "and" can be applied in case this fuzzy conditional proposition relates several antecedents to (several) consequents. For instance, the following fuzzy control rule

IF "x is *A*" and "y is *B*" THEN "z is *C*" (Def. I.3)

contains two antecedents ("x is *A*" & "y is *B*") related by a sentence connective "and". In practice, each "and" sentence connective must be converted into a *fuzzy conjunction operator*. Other sentence connective such as "or" are treated as *fuzzy disjunction operators*. Finally, the consequence/s (e.g. "z is *C*") are represented by the *fuzzy implication operators*. All these operators are outlined in appendix A.

I.P. 3.1. Choice of Process State and Control Action Variables

The selection of the (*linguistic*) *control action variables* depends on the implementation objectives (and, more particularly, on the desired policy for the controlled process).

These variables have a substantial effect on the general performance of the FLC. The number of input (process state) and output (process control) variables relies on the complexity of the fuzzy control system being implemented. Simple FLCs only require the use of a Single-Input, Single-Output (SISO) Rule Base. More complex fuzzy control structures are referred to as Multiple-Input, Multiple-Output (MIMO) systems whose Rule Base is usually expressed as

$$R = \{R_i\}_{i=1}^{n} \qquad \text{(Def. I.4)}$$

where R_i represents a single fuzzy control rule defined by

$$\text{IF } (\text{“}x_1 \text{ is } A_1\text{“ and ... and “}x_p \text{ is } A_p\text{”}) \text{ THEN } (\text{“}z_1 \text{ is } C_1\text{“ and ... and “}z_q \text{ is } C_q\text{”}) \qquad \text{(Def. I.5)}$$

where p is the number of inputs, q is equal to the number of outputs and n corresponds to the total number of fuzzy control rules in the MIMO system.

I.P. 3.2. Type and Derivation of the Fuzzy Control Rules

Two main *classes of fuzzy control rules* can be applied within a FLC architecture. These are:

- state evaluation rules and,
- object evaluation rules.

Depending on the source of derivation of the Rule Base, the designer can decide whether to apply fuzzy conditional propositions between linguistic input variables (i.e. through state evaluation) or, to predict a fuzzy control policy with the use of an object evaluation that is suitable for the system. The first choice consists of applying methods such as observation or heuristic verbalisation of the controlled process. With the second type, the control of the process (which may, of course, not be easily observed) is analysed and expressed in the form of predictive conditional statements.

The literature of Fuzzy Logic Control provides the following four methods for the derivation of fuzzy control rules [Lee I-90]:

- *Expert experience and control engineering knowledge.* Conditional statements such as Def. I.5 express certain domain knowledge and experience in the language of fuzzy IF-THEN rules. The formulation of these involves either introspective verbalisation or interrogation of experts or operators.
- *Operator control actions.* This method corresponds to industrial man-machine control system applications requiring the manipulation of input-output connections which are not known with sufficient precision to use Classical Control Theories.
- *A fuzzy model of a process.* A linguistic description of the dynamic characteristics of the controlled process is used in order to produce a fuzzy model related to FLC implementation. Then the set of fuzzy control rules is generated from this model and through the attainment of optimal performance of the dynamic system.
- *Learning.* This methodology has been used to emulate human decision making behaviour focused on the ability to create fuzzy conditional statements based on experience. An example of this work is the Sugeno's fuzzy car which has a learning capability to allow this to park "by itself".

We have implemented our FLC architecture (described in section 3.5) by considering the heuristic verbalisation, used in the first method.

However, we are interested in the identification of the learning capabilities mentioned in the fourth method and it is our intention to investigate this in subsequent work.

I.P. 3.3. Completeness and Consistency of the Rule Base

These attributes of the Rule Base refer to the number (completeness) and coherence (consistency) of the fuzzy control rules given in Def. I.5. The maximum number that could be acquired for a certain controlled process is a function of both the type of the system (according to I.P. 3.1) and, the number of fuzzy partitions (from I.P. 2.2). For instance, in the case of a SISO system designed with an input linguistic variable "x" which has card(T("x")) = 5 and, an output "y" with card(T("y")) = 6, the maximum number which can be found is equal to 5*6 = 30 fuzzy control rules. Unfortunately, there is no general operation for estimating an optimal number of fuzzy control rules.

3.4.2.4. Component 4: Decision Making Logic Unit

This functional unit performs the *fuzzy inference of control action/s* by evaluating the Data and Rule Bases using the fuzzified input variables derived from the Fuzzification Interface.

To describe the parameters of this unit, we consider a MIMO system with a Rule Base R (as in Def. I.14) and a set of fuzzified input variables or fuzzy sets $A = (A_1, \ldots, A_p)$.

The *inference* of a set of q action (output) variables or fuzzy sets denoted by $C = (C_1, \ldots, C_q)$ has been defined as the result of the compositional equation:

$$C = A \circ R \qquad \text{(Def. I.6)}$$

where the symbol $\circ$ represents a sup-star compositional operator (Def. A.11.1) such as the sub-product operation or the sup-min operation.

I.P. 4.1. Definition of Fuzzy Implication

There are a number of fuzzy relations that can be used *to relate antecedents to consequents* within each fuzzy control rule manipulated in a Decision Making Logic Unit. Some selecting criteria are:

- smoothness,
- unrestricted inference or,
- symmetry of inference (see [Lee-90]).

The designer's final judgement reflects not only the underlying implication of the fuzzy control actions but also, the effect of connecting different fuzzy sets within the inference process. Thus, it is important to notice that the selected fuzzy implication operator should be as flexible as possible.

The following formula extends the expression of a MIMO's fuzzy control rule (given in Def. I.5) into the symbolic definition of a fuzzy implication

$$(A_1 \times ... \times A_p) \rightarrow (C_1 \times ... \times C_q) \qquad \text{(Def. I.7)}$$

The practical implementation of this entails providing a fuzzy operator for the ($\times$) *fuzzy relation* and the ($\rightarrow$) *symbolic connective*. In our studies, we have used Mamdani's & Larsen's methods (see below) to control the movements of a simulated mobile robot. But it should be noted that there are other techniques available, as listed in appendix A.

I.P. 4.2. Interpretation of "and" & "also" Connective Operators

In applying Def. I.6 and Def. I.7, it is understood that the inference mechanism (and so the performance of the FLC) relies on the manipulation of a set of fuzzy control rules that are related by the twin concepts of the fuzzy implication and the sup-star compositional rule of inference.

The overall behaviour of the FLC can be characterised as the combination of the fuzzy relations that represent the fuzzy control rules. This is achieved through the use of the sentence connectives "and" (in partial connections of a fuzzy relation) and "also" (in the combination of the whole rule set).

In other words, the "and" operator can be used to connect antecedent conditions and/or consequent conditions to each fuzzy control rule (see Def. I.5). Hence, any fuzzy relation defined as a *triangular norm* (fuzzy conjunction or * operator defined in appendix A) can be applied in its interpretation over the product space of the universes of the underlying related linguistic variables.

When the Rule Base of the FLC is designed and its fuzzy control rules are combined, the ordering of these is immaterial. The "also" sentence connective (having commutative and associative properties) can be defined as the union operator and put to use between the different fuzzy relations corresponding to each fuzzy control rule. For instance, following the example of the MIMO system and, applying this sentence connective to its Rule Base (in Def. I.4), the following symbolic expression

$$R = \text{also}(R_1, R_2, ..., R_n) \qquad \text{(Def. I.8.1)}$$

or, its extended formula

$$R = R_1 \text{ also } R_2 \text{ also } ... \text{ also } R_n, \qquad \text{(Def. I.8.2)}$$

completes the inference mechanism provided in Def. I.6. The final union operation between the fuzzy control rules is shown in our particular inference mechanism (Equ. 3.12 in section 3.5.4.1).

I.P. 4.3. Definition of Compositional Operator

This parameter consists of selecting the type of $\circ$ operator for the composition (Def. I.6) which, in practice may be expressed as a *sup-star composition*, where "star" denotes a *triangular norm operator* chosen for the specifications of the FLC (see appendix A).

Again, the literature of Fuzzy Logic Control offers different possibilities (compositional operators), namely

- *sup-min operation*,
- *sup-product operation* and,
- *sup-bounded* operation.

These differ from each other according to the *combination operator* used for the fuzzy control rules that is symbolically defined by the "also" (union) connective. The sup-min and sup-product compositional operators are the most frequently used in FLC applications (see [Dubois & Prade; Wang & Loe-93). The two methods provide similar accuracy but less computational effort.

We have used both *compositional operators* for the implementation of the Collision Avoidance FBM. We have applied the *sup-min operation* to *Mamdani's inference mechanism*. After this, we have added the *sup-product operation* to *Larsen's inference mechanism*. The results of applying both techniques can be seen in the control actions (Equ. 3.14 and Equ. 3.15) of the FLC architecture.

I.P. 4.4. Choice of the Inference Mechanism

The fuzzy operators selected for implementing the sentence connectives "and", "also" (with related compositional operator) and fuzzy implication ($\rightarrow$) configure a type of fuzzy inference mechanism in a FLC.

The *FLC's fuzzy inference method* results from the application of the parameters I.P. 4.1 to I.P. 4.3 expressing the membership values of the control actions inferred by the Decision Making Logic Unit.

Further, it is associated with *firing strengths* (or, *weighting factors*) that measure the contributions of the fuzzy control rules depending on the controlled-process input data.

In section 3.5, we describe our experiments with these inference mechanisms using two methods: *Mamdani's & Larsen's* and include formulas of the control actions as well as the formalisms of the firing strengths using *3D functional representations* (see Figure 3.15(a) & Figure 3.15(b)).

3.4.2.5. Component 5: Defuzzification Interface

Crisp (numerical or non-fuzzy) control actions are required in many practical applications of FLCs. Thus a Defuzzification Interface is necessary in order to calculate a crisp value for each fuzzy control action.

I.P. 5.1. Strategy

A *defuzzification strategy* is a mapping from the space of fuzzy control actions to a space of non-fuzzy control actions. This strategy has to be obtained using the inferred fuzzy control actions at the Decision Making Logic Unit. The specification of the non-fuzzy control actions comes from the I.P. 3.1.

The designer can select between, at least, max criterion, mean of maximum and centre of area strategies described below.

I.P. 5.2. Operator

There are three *defuzzification operators* that have been widely applied in the literature of Fuzzy Logic Control. These are:

- The *Max method*. This produces control actions reaching the maximum value.

- The *Mean Of Maximum method* (MOM) that generates a control action representing the mean value of all the local control actions whose membership functions reach the maximum. The following equation

expresses the (generic & crisp) action provided by this method in case of a discrete universe of control actions:

$$z_0 = \sum_{j=1}^{l} w_j \Big/ l \qquad \text{(Def. I.9)}$$

where wj is the value of the support of a fuzzy control action at which its membership function reaches the maximum value and l represents the (discrete) number of elements in such support.

- The *Centre Of Area method* (COA). This is the defuzzification operator most commonly used in FLC applications. It outputs the centre of gravity of the possibility distribution of each control action by calculating

$$z_0 = \frac{\sum_{j=1}^{n} \mu_c(w_j).w_j}{\sum_{j=1}^{n} \mu_c(w_j)} \qquad \text{(Def. I.10)}$$

where $\mu c(wj)$ is the membership value of the (control action) fuzzy set C with $z0 \in \sup(C)$ and n is equal to the number of its fuzzy partitions.

We have used COA in the FLC. Figure 3.8 (in section 3.5.5) shows a graphical interpretation of the data-driven developed in the Defuzification Interface while applying Def. I.10.

3.5. The FLC-Based Implementation of the Collision Avoidance FBM

This section describes how we have implemented the Collision Avoidance FBM (Figure 3.2) using the table-based FLC architecture described above (Figure 3.4), the Khepera Simulator program (see Figure 3.5) and, some Fuzzy Logic principles that we summarise in appendix A (Fuzzy Sets and Fuzzy Relations).

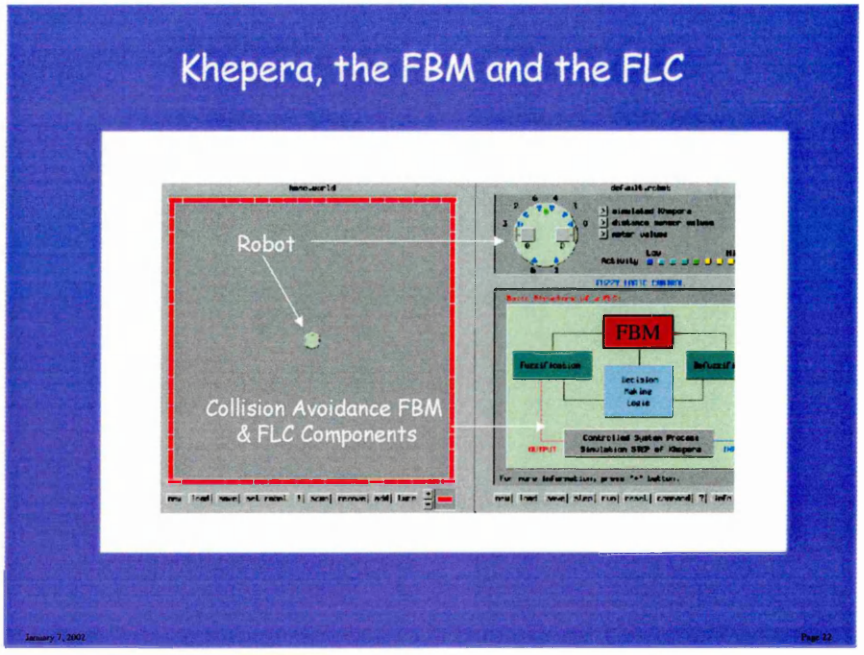


Figure 3.5: The Khepera Simulator, the Collision Avoidance FBM and the FLC.

The Khepera Simulator includes two main sources: *robot* and *world*. The robot (displayed on the right side of Figure 3.5) consists of two lateral motors that can go backward and forward at different speeds and, 8 infrared (IR) sensors that allow the simulated robot to detect by reflection (explorations of 15 points in a triangle) the proximity of objects placed in its surroundings (world shown on the right side of Figure 3.5).

Motor speed values range between -10mph and 10mph. A random noise $\Delta m = \pm 10\%$ is added to the amplitude of the motor speed while $\Delta r = \pm 5\%$ is added to the direction resulting from the difference of the speeds of the motors. The IR sensor values range between 0m and 1023m (plus an amplitude disturbance equal to ±10%). Then any suitable control model must read (update) these sensor and motor values from a specified position and direction of movement within a simulated world built of bricks.

The FBM-FLC architecture that we have developed and installed within the Khepera Simulator system controls the robot's movement (see Figure 3.5). At each simulation step, the input interface fuzzifies the 8 IR signals and motor speed values.

We have used fuzzified sensor values to implement of the *perceiving_obstacles* node of the Collision Avoidance FBM. This constructs the present perception of the environment at the *right*, *left* and *back* sides of Khepera. Similarly, we have used fuzzified motor values to implement the *moving_around* node of the FBM. These represent the current motion of the controlled robot in the level of fuzzy sets. Finally, we have used some fuzzy inference mechanisms to activate the FBM while increasing / decreasing the firing degree of its *varying_speed* node to control the robot's movements in its surroundings.

All these mechanisms control the observed perceptions and motions together to produce a fuzzy variation of speed for each motor of the robot. They fire the *level of activation of the collision avoidance behaviour*. The Defuzzification Interface of the FLC transforms the resulting fuzzy speed variations (fuzzy sets) into the corresponding crisp variations (one for each motor). In the next simulation step of the controlled process, the motor values are modified and the Khepera robot adjusts its speed and direction accordingly i.e., the control action (*varying_speed*) of the Collision Avoidance FBM is fired so that the robot moves accordingly.

The following sub-sections (3.5.1 to 3.5.5) describe all the FLC I.Ps that we have defined to implement our Collision Avoidance FBM. The next section (3.6) summarises our experimental results using the resulting FBM implementation (FBM-FLC architecture). This includes some explanations about how we have converted this architecture onto a C program that runs within the Khepera Simulator System.

3.5.1. Fuzzification Interface

This functional unit of the Collision Avoidance FLC fuzzifies crisp values of two universes of discourse: $D = \{0,1023\}$ and $S = \{-10,10\}$ which characterise Khepera robot's sensor signals and motor speed, respectively.

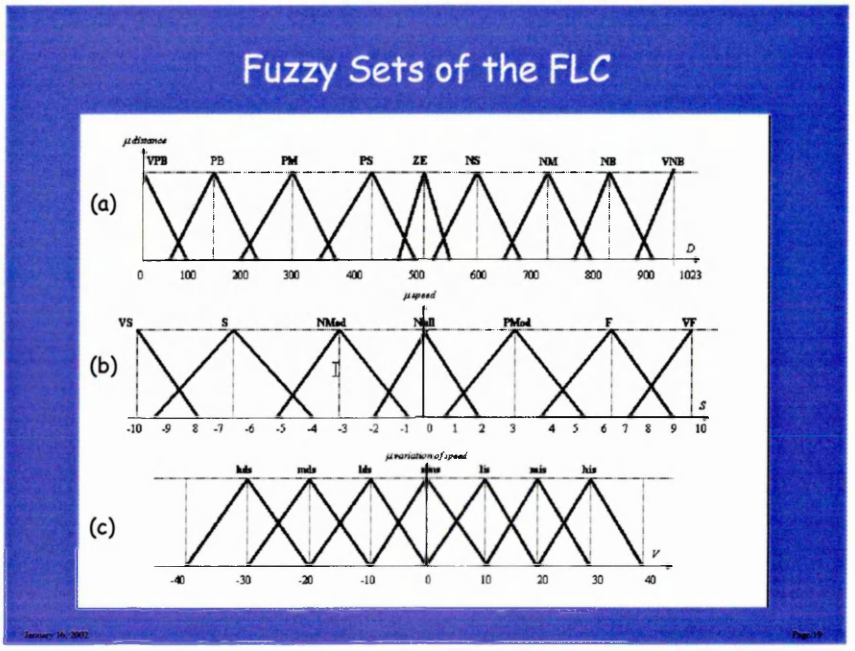


Figure 3.6: Membership Functions of Primary Fuzzy Sets of the FBM-FLC. (a) Membership functions of *fuzzy distance*: VPB, very positive big; PB, positive big; PM, positive medium; PS, positive small; ZE, zero; NS, negative small; NM, negative medium; NB, negative big; VNB, very negative big. (b) Membership functions of *fuzzy speed*: VS, very slow; S, slow; NMod, negative moderate; Null, null; PMod, positive moderate; F, fast; VF, very fast (c) Membership functions of *fuzzy variation of speed*: hds, highly decrease speed; mds, moderately decrease speed; lds, lightly decrease speed; nms, no modify speed; lis, lightly increases speed; mis, moderately increase speed; his, highly increase speed.

The mapping (I.P. 1.1) from input variables (distance and speed) to the corresponding universe of discourse is done according to the *singleton method operator* [Dubois & Prade-83].

The 8 IR sensor-input data are converted to a term of the linguistic variable *distance* defined in *D* and associated to the *perceiving_obstacle* node of the FBM. However, left and right motions associated to the *moving_around* node of the FBM are articulated into a term of *speed* that is defined in *S*. All terms (fuzzy sets) of *distance* and *speed* are shown in Figure 3.6(a) and Figure 3.6(b), respectively.

3.5.2. Data Base

We have implemented this functional module within the FLC architecture in order to meet two main objectives:

- the *static storage of fuzzy partitions* and,
- the *establishment of dynamic connections with the Collision Avoidance FBM* mapped onto the Rule Base of the FLC architecture.

The discretization (or I.P. 2.1 needed for the design of the fuzzy partitions) has been achieved by assigning membership values to each element in the discrete universes of discourse *S* and *D*. The set of fuzzy partitions (I.P. 2.2) stored in the Data Base includes those corresponding to the input variables of the controlled process (*distance* and *speed*) and other primary fuzzy sets (*perception* and *variation of speed*) needed for the implementation of the fuzzy control rules. These are explained in a later point of this section.

We have constructed all the fuzzy terms associated to the *perceiving_obstacle* node (*perception* in the FLC) using crisp values from *D*. Further, we have defined the membership functions of these fuzzy sets as fuzzy disjunction relations between fuzzified distances to obstacles (see Table 3.2).

The primary fuzzy sets of the *varying_speed* node (*variation of speed* in FLC) have been designed over the universe of discourse $V = \{-40,40\}$, where boundary values (-40 and 40) represent minimum and maximum values, respectively, to be output by the Defuzzification Interface.

Even though real extreme modifications of motor speeds could be restricted to an interval such as {-5,5}, the discrete space of variations (V) appears to refine the control by enlarging the number of membership values for each fuzzy variation of speed shown in Figure 3.6(c). A later validating function implemented within the defuzzification module of the FLC is responsible for reducing modified speed motor values to those that belong to S. Thus, the FLC is not permitted to output to the Khepera simulated robot control actions bigger than 10mph or, smaller than -10mph.

At each simulation step, the dynamic connection we have developed between the Data Base and the Rule Base (Collision Avoidance FBM) allows the Khepera System to build *levels of perception of obstacles* (on each side of the simulated robot) using the current fuzzified sensor values at the input interface. Hence, we have manipulated the fuzzy sets of the *perceiving_obstacle* node as a part of the fuzzy control rule set.

Regarding the distribution of the fuzzy sets, our FLC produces a look up table of partitions (segments of crisp values) by searching maximum membership values. A limit function is responsible for achieving this objective. For each ordered pair of fuzzy sets (e.g. A = "PB" & B = "PM"), this function calculates

$$\lim(A,B) = v_0 \text{ where } \mu_A(v_0) = \mu_B(v_0) \qquad \text{(Equ. 3.1)}$$

over the intersection of supports (see this in all fuzzy sets represented in Figure 3.6)

$$v_0 \in \sup(A) \cap \sup(B)\,;\, \forall A, B \text{ such that } A \cap B \neq \{\varnothing\} \qquad \text{(Equ. 3.2)}$$

so that the completeness (I.P. 2.3) of the whole Data Base is guaranteed since resulting fuzzy partitions of *distance*, *speed* and *variation of speed* cover D, S and V, respectively.

Finally, we have used triangular (functional) membership functions (I.P. 2.4) to represent all the fuzzy sets (see Figure 3.6). This implementation parameter deals with the *fuzzy arithmetic operations* that we explain in more detail in the next sections.

3.5.3. Rule Base

The Rule Base that we present in this section is based on *state evaluation* rather than predictive fuzzy control rules (see comments in I.P. 3.2). In fact, this functional unit implements the Collision Avoidance FBM for inferring actions from an evaluation of the controlled movements of the Khepera robot rather than from an evaluation of the control actions represented in such map.

According to I.P. 3.1, we selected both *distance* and *speed* as linguistic variables that describe the controlled process (*perceiving_obstacles* and *moving_around* nodes) and, the *variation of speed* as a control action linguistic variable (*varying_speed* node).

The terms of *perception* deal with an (additional) internal variable that we have added to develop the Rule Base. After this selection of variables and in order to cope with I.P. 3.2 & I.P. 3.3, we divided the FBM into two sub-units and mapped these onto two Rule Bases within the FLC. These two Sub-Rule Bases are as follows.

3.5.3.1. Sub-Rule Base R1: Levels of Perception

The Sub-Rule Base R1 is an Eight-Input, Three-Output system that allows the construction of the terms of *perception* by converting 8 fuzzy sets of *distance* into 3 fuzzy sets (one for each side of perception: left, right and back). More particularly, the mapping developed by this sub-unit consist of grouping the following sensor signals of the simulated robot:

- IRS0, IRS1, IRS2 → pleft ∈ T("perception")
- IRS3, IRS4, IRS5 → pright ∈ T("perception")
- IRS6, IRS7 → pback ∈ T("perception")

The other sub-component (*R2*) computes the verbalisation of the control policy by a Three-Input, Two-Output system. Thus, the FBM-FLC is a control architecture of the form

$$(v_{left}, v_{right}) = K(p_{left}, p_{right}, p_{back}, s_{left}, s_{right}) \quad \text{(Equ. 3.3)}$$

in which both control actions

$$v_{left}, v_{right} \in \text{T("variation of speed")} \quad \text{(Equ. 3.4)}$$

rely on the degree to which an obstacle is perceived (or level of perception) and, on the fuzzified speed motor values. These terms are denoted by

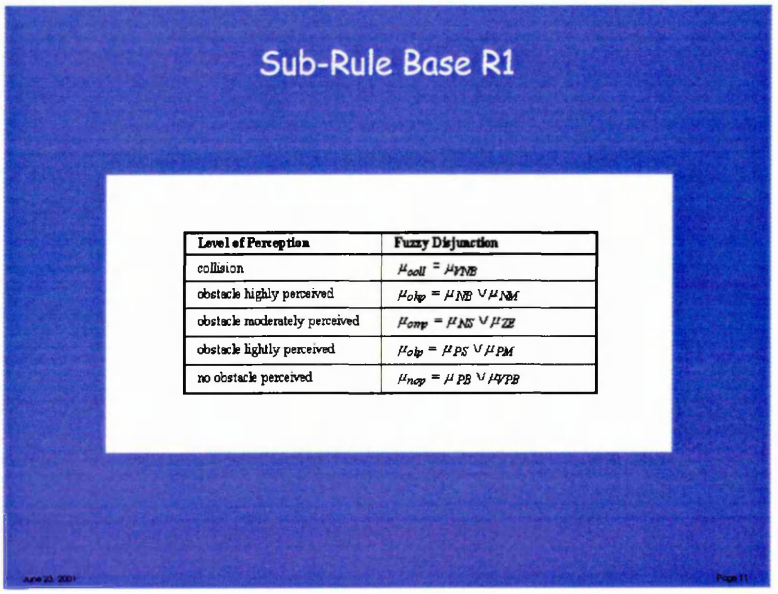
$$p_{left}, p_{right}, p_{back} \in \text{T("perception")} \quad \text{(Equ. 3.5)}$$

$$s_{left}, s_{right} \in \text{T("speed")} \quad \text{(Equ. 3.6)}$$

The computations (mappings) developed within *R1* involve the step-by-step (dynamic) connection of the Rule Base with the Data Base for this sub-unit (*R1*) to generalise groupings of fuzzified sensor signals into terms of *perception*. Regarding the formalism used for the construction of *perception*, we have applied the fuzzy disjunction between a number of terms of *distance*. Thus, levels of perception become point wise defined by the triangular co-norm operator selected for this fuzzy operation and, by the number and type of fuzzy sets of *distance* being related.

We have implemented a prototype of this Sub-Rule Base *R1* (see Table 3.2) using the *union triangular co-norm* as *fuzzy disjunction between two fuzzy distances*. However, it could be experimentally demonstrated that the sort of trajectories followed by the Khepera robot would vary modifying this aspect related to the construction of a level of perception. Additionally, we believe that the same consequence could be expected when either increasing or decreasing the cardinality of the set of terms of the variable *perception*.

The fuzzy sets that we have selected for the levels of perception correspond to T("perception") = {"collision", "obstacle highly perceived", "obstacle moderately perceived", "obstacle lightly perceived", "no obstacle perceived"}.



Level of Perception	Fuzzy Disjunction
collision	$\mu_{coll} = \mu_{VNB}$
obstacle highly perceived	$\mu_{ohp} = \mu_{NB} \vee \mu_{NM}$
obstacle moderately perceived	$\mu_{omp} = \mu_{NS} \vee \mu_{ZE}$
obstacle lightly perceived	$\mu_{olp} = \mu_{PS} \vee \mu_{PM}$
no obstacle perceived	$\mu_{nop} = \mu_{PB} \vee \mu_{VPB}$

Table 3.2: Prototype of R1. Levels of Perception

Notice in the left part of Table 3.2 that each of these fuzzy sets comes from an evaluation of 1 or 2 fuzzy sets. Notice also that the membership functions of these fuzzy sets are defined by the *union triangular co-norm operator* (∨) as *fuzzy disjunction*. For instance, the term "obstacle lightly perceived" ("olp") corresponds to the fuzzy disjunction of the fuzzy distances "NB" and "NM". However, another possible prototype of *R1* could be designed for adjusting this fuzzy set to the *algebraic sum triangular co-norm* "NB+NM+NS" as the *fuzzy disjunction of distances to obstacles*. The immediate effect coming from this and, without varying the main control policy (definition of *fuzzy variation of speed*), is that Khepera would have a modified *understanding* of terms such as "obstacle highly perceived" ("ohp"). Thus, the robot would perform different collision avoidance behaviours by following other sorts of paths.

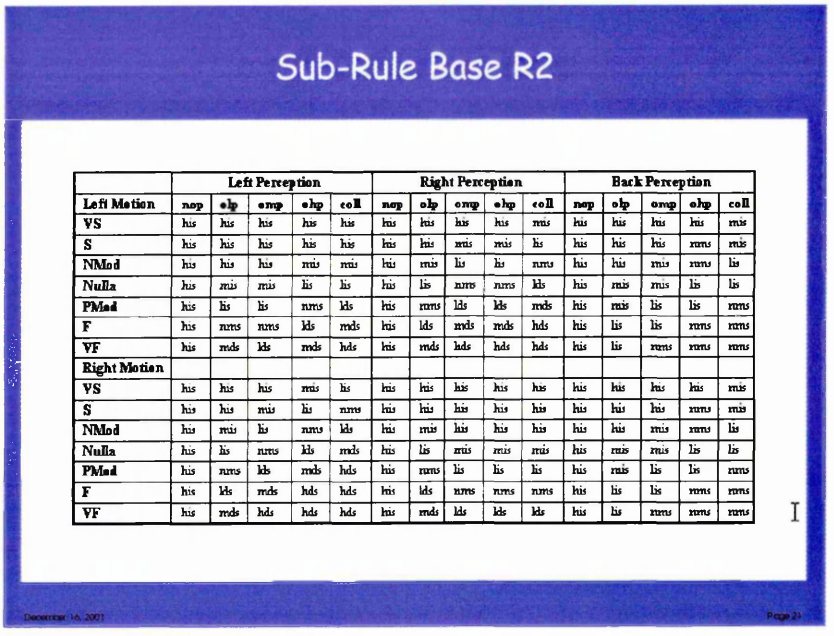
3.5.3.2. Sub-Rule Base R2: Collision Avoidance Control Policy

The sub-unit of the control policy (*R2*) has been built on a set of fuzzy control rules that relate perception to motion in the form of fuzzy conditional prepositions expressed as

$$\text{if } \text{“}p \text{ is } Ap\text{”} \text{ and } \text{“}s \text{ is } As\text{”} \text{ then } \text{“}v \text{ is } Av\text{”} \qquad \text{(Equ. 3.7)}$$

where *Ap* represents a fuzzy set of *perception* ($p \in$ T("perception")), *As* is a fuzzy set of *speed* ($s \in$ T("speed")) and *Av* is a fuzzy set of the control action *variation of speed* ($v \in$ T("variation of speed")).

Table 3.3 represents a prototype of *R2*. The three inputs to this control policy are presented in three different sub-tables. One for each side of perception.



	Left Perception					Right Perception					Back Perception				
Left Motion	nop	olp	ovp	ohp	coll	nop	olp	ovp	ohp	coll	nop	olp	ovp	ohp	coll
VS	his	his	his	his	his	his	his	his	his	mis	his	his	his	his	mis
S	his	his	his	his	his	his	his	mis	mis	lis	his	his	his	nms	mis
NMod	his	his	his	mis	mis	his	mis	lis	lis	nms	his	his	mis	nms	lis
Nulla	his	mis	mis	lis	lis	his	lis	nms	nms	kls	his	mis	mis	lis	lis
PMod	his	lis	lis	nms	kls	his	nms	lds	kls	mds	his	mis	lis	lis	nms
F	his	nms	nms	kls	mds	his	kls	mds	mds	hds	his	lis	lis	nms	nms
VF	his	mds	kls	mds	hds	his	mds	hds	hds	hds	his	lis	nms	nms	nms
Right Motion															
VS	his	his	his	mis	lis	his	his	his	his	his	his	his	his	his	mis
S	his	his	mis	lis	nms	his	his	his	his	his	his	his	his	nms	mis
NMod	his	mis	lis	nms	kls	his	mis	his	his	his	his	his	mis	nms	lis
Nulla	his	lis	nms	kls	mds	his	lis	mis	mis	mis	his	mis	mis	lis	lis
PMod	his	nms	kls	mds	hds	his	nms	lis	lis	lis	his	mis	lis	lis	nms
F	his	kls	mds	hds	hds	his	kls	nms	nms	nms	his	lis	lis	nms	nms
VF	his	mds	hds	hds	hds	his	mds	kls	kls	kls	his	lis	nms	nms	nms

Table 3.3: Prototype of R2. Collision Avoidance Control Policy

The possible combinations "*p* is *Ap* and *s* is *As*" make this prototype configurable by 210 fuzzy control rules. Note that the three Sub-Rule Bases are divided into 5 terms (fuzzy sets) of *perception* × 2 motors × 7 terms of speed = 70 fuzzy control rules for each sub-component of *R2*. All these fuzzy control rules represent *the final refinement of the Collision Avoidance FBM* since we have converted its 3 nodes into 210 fuzzy control rules that the FLC executes for Khepera to perform collision avoidance behaviour patterns.

The resulting *collision avoidance control policy* was achieved after several trials involving the simulated robot and the FBM-FLC architecture. The methodology we followed to develop this prototype of *R2* consisted of selecting approximated speeds (rather than all possible combinations of *Ap* & *As*) at which Khepera was desired to turn in order to avoid the perceived obstacles. Then the control action of each fuzzy control rule was achieved by selecting (depending on the speed *As* and level of perception *Ap*) the variation of speed required to achieve those approximated speeds.

For example, in the case of "collision on the left", we have taken the approximated speeds equal to "Pmod" (positive moderate) and "Nmod" (negative moderate) for left and right motors, respectively, to *control a turn right movement*. Thus, the control actions corresponding to *no obstacle perceived* on the left side of the Khepera robot ("nop" column of the left perception sub-table) while moving at the lowest speed ("VS" row for the left and right motions) appear in the table equal to "his". The same can be applied to any other level of perception on the left side of the robot while this is moving at left motor speed equal to "VS". Let us consider now the "coll" column at the left perception sub-table. This shows that in the case of running Khepera at a "VS" left motor speed, it is possible to give a *Av* = "his" to the same left motor. By contrast, if we consider that the Khepera's left motor stops (*As* = "null"), then this cannot be modified more than "lightly" (*Av* = "lis") because we would like the Khepera's left robot to acquire a "Pmod" speed. Therefore, when Khepera's left motor speed is equal to "Pmod", the selected control action *Av* comes equal to "nms", this is, "no modify speed".

Although our pilot tests (described in section 3.6.4) will show that this prototype of *R2* (that implements the Collision Avoidance FBM) allows Khepera to avoid stable obstacles placed in its way, *there are many other possible sets of fuzzy control rules that could provide a suitable control policy.* For instance, the whole table could be designed by pointing to other pairs of motor speed rather than "Pmod" (positive) and "Nmod" (negative) for solving collision on the left. The same could be achieved by considering other different fuzzy sets for an starting point on the right side of perception. Furthermore, it does not matter whether the selected speeds differ from one column to another. We just need to keep on training positive left and negative right motor speed for Khepera to turn right and, negative left and positive right for a turn left movement while the level of perception of an obstacle increases at left or right side, respectively.

The completeness and the consistency (I.P. 3.3) of our Rule Base are guaranteed since the number of fuzzy relations included in both *R1* and *R2* seem to allow the FLC to control the required avoidance behaviour of the simulated robot.

The performance of both prototypes (Table 3.2 and Table 3.3) can probably be improved by building very complex simulated environments. For example, it is worth placing many objects in intriguing situations or, using object (predictive) evaluation control statements to build up reactive fuzzy conditional statements within both Sub-Rule Bases.

In order to test the presented prototypes of *R1* and *R2*, our computer simulations carried out with the Khepera System and the FBM-FLC architecture have been focused on two main objectives:

- the *general fitness* of the Sub-Rule Bases *R1* & *R2* and,
- the *suitability of the fuzzy implication operators* needed (Table A.2) for inferring appropriate control actions from the whole fuzzy control rules set.

A later section 3.6 provides more detailed descriptions of these pilot tests.

3.5.4. Decision Making Logic Unit

This unit activates the Collision Avoidance FBM inferring the fuzzy variations of speed for both Khepera's motors. In effect, the *decision making logic* we have implemented with the FLC processes two membership values: μAv_{left} & μAv_{right}. The required computation uses a compositional rule of inference (see Def. I.6) between the current fuzzified speed values (Equ. 3.6), the side perceptions from *R1* (Equ. 3.5) and, the control policy encoded within *R2* (Table 3.3). Then the result of this inference process is supplied to the next unit (Defuzzification Interface) and Khepera moves accordingly with the new velocity. The compositional operators and related I.Ps that we have used to implement this unit are described below.

3.5.4.1. The Fuzzy Inference Mechanism

In general, the Decision Making Logic Unit of a MIMO (Multiple-Input-Multiple-Output) FLC performs the fuzzy inference of control actions by the compositional equation expressed in Def. I.6 where A corresponds to the set of fuzzified input operations, C is the set of fuzzy control actions and the Rule Base R (the set of fuzzy control rules that correspond to the Collision Avoidance FBM) is interpreted as the union of n fuzzy control rules.

By applying these definitions, the main design parameters that we have used for the FBM-FLC architecture correspond to the *fuzzy implication* ($\rightarrow$ symbol in I.P. 4.1) and the *sup-star composition operators* ($\circ$ or I.P. 4.3). The fuzzy implication operates on the $\times$ fuzzy relation between antecedents expressed in the fuzzy control rules of *R2*

$$Ap \times As \rightarrow Av \qquad \text{(Equ. 3.8)}$$

The sup-star composition computes the relation between the whole control policy (*R2*) and the set of current perceptions and fuzzified motor speed.

Current levels of perception (Equ. 3.5) become defined as the fuzzy disjunction of fuzzified distances developed within *R1*. The fuzzified motor speed (Equ. 3.6) is obtained from the connection of this inference unit with the Fuzzification Interface. However, the contribution of the whole set of fuzzy control rules comes from the connection between the Decision Making Unit and the Rule Base. Consequently, the whole inference mechanism is expressed by the following equations.

Firstly, the (2D) fuzzy relation among fuzzy sets of *perception* and *speed* (see appendix A) in sup-star composition with the whole control policy (*R2*) is expressed (in terms of membership functions of fuzzy sets) by

$$\mu Av_{\alpha} = (\mu Ap_{\beta} \times \mu As_{\alpha}) \circ R2 \qquad \text{(Equ. 3.9)}$$

where $Av \in$ T("variation of speed"), $\alpha \in$ {left, right}, $Ap \in$ T("perception"), $\beta \in$ {left, right, back} and, finally, $As \in$ T("speed").

After this formalisation and, following Def. I.8.2, our Sub-Rule Base *R2* is defined in the form

$$R2 = \bigcup_{\alpha}\bigcup_{\beta}\bigcup_{i=1}^{7} SR_i \qquad \text{(Equ. 3.10)}$$

in which i=1,...,7 corresponds to the cardinality of the set of terms of *variation of speed* and, the fuzzy relation

$$SR_i = Ap_{\beta} * As_{\alpha} \qquad \text{(Equ. 3.11)}$$

represents (from Equ. 3.8) the fuzzy conjunction (*) between the antecedents of the fuzzy control rules in the Sub-Rule Base *R2*).

In the end, we have developed an inference mechanism by a later implementation of the equation

$$\mu Av_{\alpha} = (\mu Ap_{\beta} \times \mu As_{\alpha}) \circ (\bigcup_{\alpha} \bigcup_{\beta} \bigcup_{i=1}^{7} (Ap_{\beta} * As_{\alpha}) \qquad \text{(Equ. 3.12)}$$

that denotes how the membership functions of the control actions $\mu Av\alpha \in [0,1]$ are derived from the contribution of the whole control policy and, by comparison of this to the membership values of the current fuzzy data μAp & $\mu As \in [0,1]$.

The completion of this fuzzy inference mechanism, in contrast to other methods of inference [Zadeh *et al*-92; Wang *et al*-93], is achieved by *firing the whole set of fuzzy control rules*. Each consequent obtained after the fuzzy implication ($\rightarrow$) of each rule is weighted by how close its antecedents matches the current fuzzy data and, finally, the results of the firing control policy (*R2*) are combined by sets of fuzzy control rules with equal consequent *Av*.

From the operations carried out in this inference mechanism, we have considered another important parameter. This is, the fuzzy conjunction * operator (or also fuzzy connective "and" as commented in I.P. 4.2) which relates the antecedents of a fuzzy control rule. The sup-star composition (I.P. 4.3) matches the current fuzzy data to the antecedents of all the rules. However, the * operator mainly determines the weighting factors for each fuzzy control rule.

Resulting contributions may be combined by the disjunction performed through the sentence connective "also" and, consequently, the interpretation of this last parameter coincides with the union (α, β, and i=1,..,7) which represents the disjunction of the number of combinations of antecedents for inferring each possible action $Av\alpha$ of the different fuzzy relations SR_i expressed in the Equ. 3.11.

Considering this, the membership functions of the control actions (α = left & right) inferred by the Decision Making Logic Unit of the FLC architecture becomes determined by extending Equ. 3.12 into a new formula

$$\mu Av_{\alpha} = \bigcup_{j=1}^{3}\bigcup_{k=1}^{70}\bigcup_{i=1}^{7}(\mu Ap * \mu As)_k \rightarrow \mu Av_i \qquad \text{(Equ. 3.13)}$$

where j is the number of sub-components of *R2* according to sides of perception (card(β)), k corresponds to the number of combinations of antecedents and, i is the number of fuzzy sets of speed. Additionally, this last expression also denotes the earlier weighting of fuzzy control rules that, from our point of view, establishes the most important aspect any FLC architecture (to be considered from I.P. 4.4).

3.5.4.2. Weighting Factors and Fuzzy Inference

Each weighting factor (firing strength) defines, along with the contribution of the *ith* rule, the inference mechanism given in Equ. 3.13.

Figure 3.7 represents a flow diagram of the fuzzy input data used in this mechanism. Here the functional units: Data Base, Rule Base, Fuzzification and Decision Making Logic Unit have been connected according to the data-transfer operations involved in the inference process. At each control stage, the Fuzzification Interface provides the fuzzy sets of speed (*As*) due to the connection with the Data Base (see the functional representation of a few fuzzy set within this last functional unit in Figure 3.6). The Sub-Rule Base *R1* deals the fuzzy levels of perception (*Ap*).

The contribution of *R2* comes from its corresponding Sub-Rule Base. Finally, the Decision Making Logic operates by applying all these fuzzy values in Equ. 3.13, according to a *fuzzy inference technique*.

In FLC applications, there exist different types of approximate (fuzzy) inference techniques [Zadeh *et al*-92; Lee I-II-90] that have been associated with inference mechanisms. Each type of fuzzy inference technique appears intrinsically defined by the fuzzy implication used in the application. Then each fuzzy implication is determined by the selection of the operators: * (fuzzy conjunction or triangular norm operator) and $\overline{+}$ (fuzzy disjunction or triangular co-norm operator).

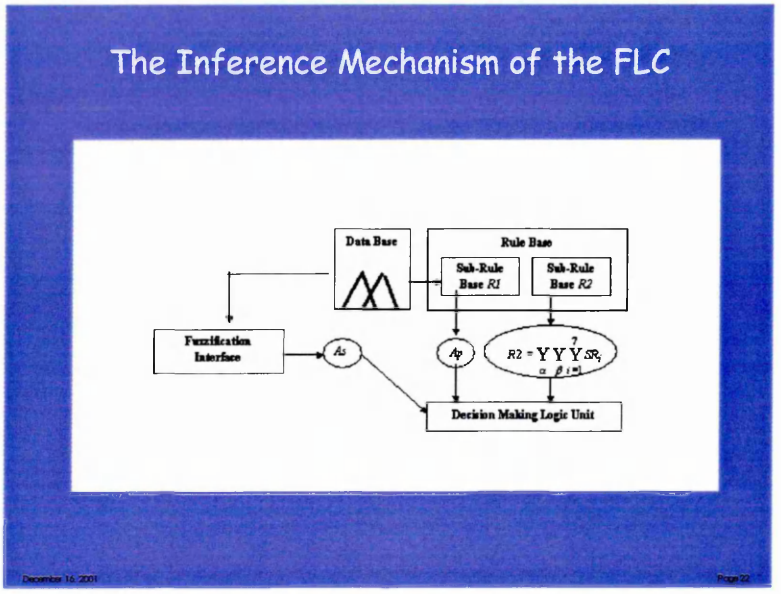


Figure 3.7: Flow Diagram of Input Data to the Inference Mechanism of the FBM-FLC

For instance, in the case of *Mamdani's minioperation rule*, the inference has been associated with the intersection as fuzzy conjunction (* = $\wedge$) and the union fuzzy disjunction ($\mp$ = $\vee$). In what follows, i.e. in the FBM-FLC architecture, the firing strengths applied for the Mamdani's minioperation rule of inference measure the contribution of the *ith* fuzzy control rule within *R2* through the expression

$$\alpha_i = (\mu Ap \wedge \mu As)_{k=1} \vee \ldots \vee (\mu Ap \wedge \mu As)_{k=70} \qquad \text{(Equ. 3.14)}$$

Other types of firing strength correspond to *Larsen's product operation rule* (second type of fuzzy inference). For the computation of these, the fuzzy implication becomes point wise defined by the algebraic product as triangular norm (* = .) for the fuzzy conjunction and the union triangular co-norm ($\mp$ = $\vee$) for the fuzzy disjunction. Hence, this technique applied to the definition of the firing strengths for the FBM-FLC architecture is defined by

$$\alpha_i = (\mu Ap.\mu As)_{k=1} \vee \ldots \vee (\mu Ap.\mu As)_{k=70} \qquad \text{(Equ. 3.15)}$$

Our FBM-FLC program (outlined in section 3.6.2) allows the manipulation of the first type (Mamdani's) and the second type (Larsen's) of fuzzy inference. Each space of firing strengths differs from each other since their values α_i depend on the operations of the membership values (Equ. 3.14 & Equ. 3.15). However, the dimensionality, cardinality and universe of discourse coincide for both sets of firing strengths. The cardinality and dimensionality can be seen in the 3D views of Figure 3.15.

The last implementation parameter (universes of discourse) of the Decision Making Logic unit corresponds to the product space of the universes D and S due to the supports of Ap and As, respectively. This makes the control architecture to compute, independently of the fuzzy inference technique, the firing strengths $\alpha_i \in [0,1]$ in a 2D-support of 1023x21 = 21483 pairs (d, s) of crisp data.

Having developed the previous computations for the contributions of our fuzzy control rules, the final expression applied in the Decision Making Logic Unit is expressed as

$$\mu Av_{\alpha} = \bigcup_{j=1}^{3} \bigcup_{i=1}^{7} \alpha_i \rightarrow \mu Av_i = \qquad \text{(Equ. 3.16)}$$

3.5.5. Defuzzification Interface

The functionality of this module is to output, at each step of the simulation, two crisp control actions: *vleft* and *vright* for modifying the left and right motor speed, respectively. The defuzzification strategy (I.P. 5.1) that we have implemented follows the association of the Av_{right} & Av_{left} pointed to by the Sub-Rule Base $R2$ with the corresponding inferred membership values μAv_{right} & $\mu Av_{left} \in [0,1]$. Before transferring the resulting crisp variation of speeds, the connections required between the Defuzzification Interface and the rest of the FBM-FLC architecture are as follows.

Firstly, the fuzzy set that corresponds to maximum firing strength (e.g. "his") is identified from the Sub-Rule Base unit $R2$. Then the Decision Making Logic Unit has to provide the membership value corresponding to

the fuzzy contribution of all the firing strengths. Finally, the defuzzification operator is responsible of providing the crisp control actions *vleft* & *vright* $\in V$ that modify the motor speed of the simulated robot Khepera. For this aim we have applied the Center Of Area method (given in Def. I.10) as the design parameter I.P. 5.2.

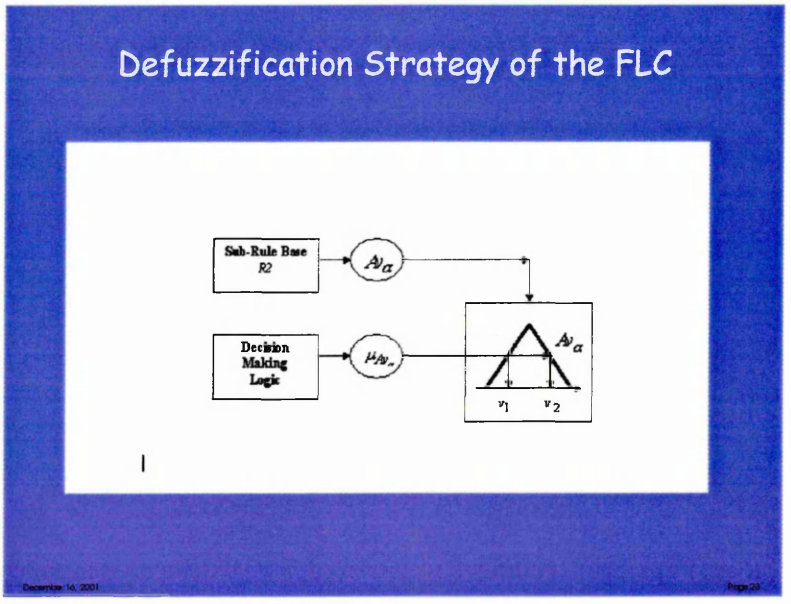


Figure 3.8: Defuzzification Strategy of the FBM-FLC.

Figure 3.8 shows a flow-diagram representation of the connections among the FLC's functional units and, the main data driven operations involved in this Defuzzification strategy. In this figure, the term $\mu Av\alpha$ denotes one of the membership functions (α= left or right) computed by the Decision Making Logic Unit. The fuzzy set represented in the left part of the figure corresponds to one of the $Av\alpha$ fuzzy control actions known by the Defuzzification Interface. This fuzzy set (with maximum firing strength) is pointed to by the Sub-Rule Base *R2*. The COA method operates by applying in Def. I.10 each crisp motor value (v_1 and v_2) identified in Figure 3.8 as $wj \in \sup(Av\alpha)$. Then this COA(v_1,v_2) operation produces the centre of gravity which corresponds to the final (left and right) crisp variation of speed supplied by the FBM-FLC in the current simulation step.

3.6. Pilot Tests and Results

This section compares pilot tests of computer simulations driven with the FBM-FLC architecture within Khepera Simulator system.

3.6.1. Khepera Simulator

Khepera Simulator allows writing and testing control algorithms using C or C++ languages. This software package runs on UNIX workstations and provides a X11 graphical interface (shown in Figure 3.5 and Figure 3.9) to visualize the robot, its surroundings and, the results of some controller components available in the system. Further, the system includes a number of libraries and control functions to facilitate the construction of new controllers driving the motors of the robot.

The graphical interface (Khepera's main window) is divided into two main parts: the *robot* and, the *world* (see Figure 3.5 and Figure 3.9). The robot (right side of the window) allows observing how Khepera's sensors and motors change according to the control architecture that the user initiates within the system. This part of the window can also be modified to display variables, graphics, explanations and other possible results coming from the controller components. Finally, in the world part, the user can observe the behaviours displayed by the robot.

Khepera source code is written in ANSI C. Its main C structures, libraries and functions are coded in the following files (available in a SRC directory):

- *sim.c* (main program),
- *sim.h* (sim header file),
- *robot.c* (robot drivers),
- *robot.h* (robot header file),

- *world.c* (world management; allows creating new worlds for testing the controllers),
- *world.h* (world header file),
- *graphics.c* (X11 graphical interface),
- *graphics.h* (graphic header file) and,
- *include.h* (to be included by *user.c*).

New controllers must be written into a USER directory, compiled into an OBJ directory and, linked to Khepera's source files. A successful operation using one of these new controllers consist of making the simulator to continuously call the corresponding control functions until the robot performs the desired functionality. Basically, this operation requires:

- building the new system (using a make file that links the simulator source files and the new controller)
- placing the robot in one of the available worlds (that Khepera loads from a WORLD directory),
- pressing a "run" button to drive the controller actions to the motors of the robot,
- observing the behaviour of the robot within the world,
- unpressing the "run" button when the desired functionality has been reached and,
- displaying and observing the path followed by the robot (to record the final results).

3.6.2. The FBM-FLC Program

We programmed our FBM-FLC architecture using the Khepera software package and, coding all the source files described below.

3.6.2.1. Fuzzy Logic Source Files

The Khepera Simulator version that we used for our computer simulations did not provide any implementation of fuzzy sets or, fuzzy logic operators. Thus, we needed to design, code and compile the following C source files (see Appendix C):

a) fuzzysets.h

The *fuzzysets.h* is the *fuzzysets* header file. It provides some basic C structures (degree of membership function values, 1-dimensional and 2-dimensional fuzzy sets, etc.) to implement the membership functions of fuzzy sets.

b) fuzzysets.c

The *fuzzysets.c* file implements a fuzzy sets framework for all the FBM-FLC units. More particularly, this file codes the following fuzzy sets definitions and operators:

- Singleton definitions.
- Triangular membership functions for 1-dimensional and 2-dimensional fuzzy sets.
- Fuzzy sets generation functions (to initiate the structure of one fuzzy set).
- Fuzzy sets cardinality functions.
- Fuzzy sets printing (to display the fuzzy sets as part of the graphical interface of our FBM-FLC).
- Writing fuzzy sets to files (to store data about the generated fuzzy sets).
- Reading fuzzy sets from file (to generate a fuzzy set structure from one data file).
- Fuzzy set theoretic operators (union, cartesian products, etc.).
- Triangular norms and co-norms.

The following source files implement all the functional units of our FBM-FLC architecture (widely described in section 3.5).

a) flc.h

This is the main header file of the FBM-FLC. In addition, it provides all the constants that are required to define the fuzzy partitions of this controller and includes some (common) dynamic structures used by all the functional units.

The constant values correspond to the minimum, maximum and centre values of the supports of the fuzzy sets represented in Figure 3.6 (section 3.5.1). Some of the dynamic structures defined within this file are:

- Partition of distances.
- Partition of Speeds.
- Perception of speeds.
- Speed variation.
- Rule Component.
- Rule Base.

b) flc.c

The *flc.c* file implements:

- fuzzy sets boolean functions from the examination of the sensors and motors of the robot (used by the Fuzzification interface) and,
- some common control functions (called by the Data Base and Rule Base units in some of our computer simulations).

c) database.c

This is the Data Base unit file. It creates all the fuzzy sets (and partitions) using the constants, structures and functions earlier described (*flc.h* and *fuzzysets.h*), provides some other functions called by the Fuzzification and Defuzzification interfaces (to manipulate and manage such fuzzy sets) and, guarantees the completeness of the database that we have designed. Some of these last functions allow:

- Finding distance partition members.
- Finding speed partition members.
- Fuzzifying sensor values.
- Fuzzifying motor values.

The file also implements some useful functions to write the partitions and fuzzified values to data files.

d) rulebase.c

The *rulebase.c* is the Rule Base unit file. This implements all the fuzzy control rules that we have designed within the R1 and R2 sub-rule bases (see equations in section 3.5.3). Some of its functions allow:

- Creating fuzzy sets of perception (as the fuzzy disjunction of fuzzified sensor values).
- Writing fuzzy sets of perception to data files.
- Creating fuzzy sets of speed variation ("his", "mis", etc.).
- Reading a fuzzy control rule antecedent.
- Reading consequences of a fuzzy control rule.
- Creating an empty rule component.
- Creating an empty rule base.

- Reading a rule base from a data file.
- Writing a rule base to a data file.

e) inference.c

The Decision Making Logic unit has been implemented in the *inference.c* file. This provides the fuzzy inference mechanisms earlier described (section 3.5.4), including some relevant functions to calculate and operate on the firing strengths (section 3.5.4.1) of the fuzzy control rules. Some of the specific functions included here are:

- Creating firing strengths.
- Calculating firing strengths disjunction.
- Writing firing strengths to a data file.
- Selecting final fuzzy set of speed variations (control action).
- Calculate Mamdani's fuzzy inference mechanism.
- Calculate Larsen's fuzzy inference mechanism.
- Inferring crisp control action (crisp speed variation).

f) interface.c

The Fuzzification and Defuzzification interfaces have been implemented in one single file (the *interface.c* file). Some of its functions allow:

- Creating fuzzy partitions of distances and speeds.
- Writing fuzzy partitions (of distances and speeds) to data files.
- Fuzzifying speed and motor values and, defuzzifying fuzzy variations of speed.

3.6.2.2. User Source Files

The following C libraries and functions correspond to the source files that we coded within the Khepera USER directory.

a) user.h (FBM-FLC user header file)

This file defines all the constant values and extern functions that are required to build one FBM-FLC within Khepera.

b) user1.c (FBM-FLC user file)

The *user1* file allows Khepera simulator calling the FBM-FLC for the robot to perform collision avoidance behaviours. Some of these functions are as follows (see code in Appendix C).

```
void UserInit (struct Robot *robot)
{
 gnuplot_file = popen("gnuplot","w");
 if (gnuplot_file == NULL)
 {
        fprintf(stderr,"Warning: gnuplot not available");
 }
 ShowUserInfo(1,1);
 CreatePrimaryFuzzyDistances();
 CreatePrimaryFuzzySpeeds();
 dist_partition  = CreatePartitionOfDistances(khepera_crisp_sensor_values);
 speed_partition = CreatePartitionOfSpeeds(khepera_crisp_motor_values);
 CompleteDistanceFuzzification(robot,dist_partition,fuzzy_distances);
 CompleteSpeedFuzzification(robot,speed_partition,fuzzy_speeds);
 CreateFuzzySetsOfPerception();
```

```
  fuzzy_states = CreateFuzzyPerceptionStates(fuzzy_distances);
  rulebase = CreateEmptyRuleBase();
  CreateFuzzySetsOfSpeedVariation();
  firing_strengths = CreateFiringStrengths( "Union2", "Triangular2",dist_partition, speed_partition);
  rule_contribs = DisjunctionOfFiringStrengths(firing_strengths,"alg union");
  fuzzy_variation = CreateNullSpeedVariation();
}
```

Khepera calls the *UserInit* function when the user starts a new simulation. We have completed this function to initialise all the fuzzy partitions of our fuzzy sets (described in section 2.5.2) as to create an empty rule base.

```
void NewRobot (struct Robot *robot)
{
  pas = 0;
  ShowUserInfo(2,1);
  FreeRuleBase(rulebase);
  FreeSpeedVariations(fuzzy_variation);
  DML_right                = 0.0;
  DML_left                 = 0.0;
  crisp_variation[RIGHT] = 0;
  crisp_variation[LEFT]  = 0;
}
```

The *NewRobot* function is called when the "new robot" button is pressed in Khepera's main window. Our implementation of this function serves to initialise all the fuzzy control variables that are manipulated by the FBM-FLC. Note that this function frees, for example, current fuzzy control rules and crisp speed values (output by the Defuzzification interface).

```
void RunRobotStart (struct Robot *robot)
{
 path_file       = fopen("STATS/path.dat","w"); /* modify to "a"*/
 leftspeed_file   = fopen("STATS/leftspeed.dat","w");
 rightspeed_file  = fopen("STATS/rightspeed.dat","w");
 leftsensors_file  = fopen("STATS/leftsensors.dat","w");
 rightsensors_file = fopen("STATS/rightsensors.dat","w");
 backsensors_file  = fopen("STATS/backsensors.dat","w");
 step_controller  = fopen("EXAMPLES/FBM_FLC/step.controller","w");
 InitStepControllerFile(step_controller);
 if (GetUserInfo()!=3 || GetUserInfoPage()!=1)
        ShowUserInfo(3,1);
 fuzzy_variation = CreateNullSpeedVariation();
}
```

The *RunRobotStart* function opens some data files (storing statistical information about the current trial of the controller) and initialises the FBM-FLC functional units.

```
boolean StepRobot (struct Robot *robot)
{
 pas++;
 DrawStep(pas);
 DrawPositionRobot(robot);
 DML_right = 0.0;
 DML_left = 0.0;
 fprintf(path_file,"%lg, %lg\n",robot->X,1000.0-robot->Y);
 fprintf(leftspeed_file,"%ld, %ld\n",pas,robot->Motor[LEFT].Value);
 fprintf(rightspeed_file,"%ld, %ld\n",pas,robot->Motor[RIGHT].Value);
```

```
/* Fuzzification */
CompleteDistanceFuzzification(robot,dist_partition,fuzzy_distances);
CompleteSpeedFuzzification(robot,speed_partition,fuzzy_speeds);
DrawFuzzySpeeds(robot,speed_partition,fuzzy_speeds);
DrawFuzzyDistances(robot,dist_partition,fuzzy_distances);

/* write fuzzy distances and fuzzy speeds to file */
fuzzy_states = CreateFuzzyPerceptionStates(fuzzy_distances);
DrawFuzzyPerceptions(robot,fuzzy_states);

/* select rules and discriminate perceptions */
 fuzzy_variation = FuzzyImplication(fuzzy_states,fuzzy_speeds,
                                    rulebase,firing_strengths,
                                    speed_partition,fuzzy_reasoning);

/* DML: Applying fuzzy reasoning.*/
/* Fuzzy implications and firing strengths
   for infering control action */
 DML_right=InferControlActionMembership(rule_contribs, fuzzy_variation->RightVariation,fuzzy_reasoning);
 DML_left=InferControlActionMembership(rule_contribs, fuzzy_variation->LeftVariation,fuzzy_reasoning);

/* Defuzzification */
  crisp_variation[RIGHT] = CAM(fuzzy_variation->RightVariation,DML_right);
  crisp_variation[LEFT] = CAM(fuzzy_variation->LeftVariation,DML_left);

/* Applying control action */
  robot->Motor[RIGHT].Value += crisp_variation[RIGHT];
  robot->Motor[LEFT].Value += crisp_variation[LEFT];
  CorrectMotorValues(robot);
```

```
  DrawFuzzySpeedVariation(robot,fuzzy_variation);
  fprintf(leftsensors_file,"%ld, %lg\n",pas,1000.0-10);
  fprintf(rightsensors_file,"%ld, %lg\n",pas,1000.0-10);
  fprintf(backsensors_file,"%ld, %lg\n",pas,1000.0-10);
  fprintf(step_controller,"%6d %6d\n",crisp_variation[RIGHT], crisp_variation[LEFT]);
  return(TRUE);   /* continue the run of the robot */
}
```

The *StepRobot* function is called by the simulator as long as the "run" button (of its graphical interface) is down. This is the core function of our controller user files. We have implemented this function to call all the FBM-FLC functional units to control the robot's behaviour. In addition, this function calls our FBM-FLC graphical components to display some relevant information on Khepera's main windows (see section 3.6.3).

c) user_inf.h (FBM-FLC graphical interface header file)

d) user2.c (FBM-FLC graphical interface)

We have coded the *user2* source file to display the FBM-FLC functional units and show all the fuzzy values that this architecture calculates / manipulates. We show all the functions of this source file in Appendix C.

3.6.2.3. Makefile

Our *makefile* compiles the FBM-FLC source files (fuzzy libraries and user files) using a C compiler (gcc -c), links these to Khepera software and, creates object files into the OBJ directory to finally produce an FBM-FLC executable file.

```
VPATH=OBJ/:SRC/:USER/
CFLAGS = -I/usr/include \ -I/usr/openwin/share/include \  -O3
LIBS   = -L/usr/lib -L/usr/openwin/lib -lX11 -lm
```

```
cc = gcc $(CFLAGS) -c

CC = gcc $(LIBS)

sim:    sim.o robot.o world.o graphics.o khep_serial.o fuzzy.o user1.o user2.o interface.o inference.o
        $(CC) OBJ/sim.o OBJ/robot.o OBJ/world.o OBJ/graphics.o OBJ/khep_serial.o OBJ/fuzzy.o
OBJ/user1.o OBJ/user2.o OBJ/interface.o OBJ/inference.o -o sim

sim.o:  sim.c sim.h
        $(cc) SRC/sim.c -o OBJ/sim.o

robot.o: robot.c robot.h
        $(cc) SRC/robot.c -o OBJ/robot.o

world.o: world.c world.h
        $(cc) SRC/world.c -o OBJ/world.o

graphics.o: graphics.c graphics.h
        $(cc) SRC/graphics.c -o OBJ/graphics.o

khep_serial.o: khep_serial.c khep_serial.h gen_types.h
        $(cc) SRC/khep_serial.c -o OBJ/khep_serial.o

fuzzy.o: CONTRIB/fuzzysets.c CONTRIB/fuzzysets.h SRC/include.h
        $(cc) CONTRIB/fuzzysets.c -o OBJ/fuzzy.o

user1.o: user1.c user.h user_info.h include.h
        $(cc) USER/user1.c -o OBJ/user1.o

user2.o: user2.c user.h user_info.h include.h
        $(cc) USER/user2.c -o OBJ/user2.o

interface.o: CONTRIB/interface.c CONTRIB/flc.h robot.h include.h
        $(cc) CONTRIB/interface.c -o OBJ/interface.o

inference.o: CONTRIB/inference.c CONTRIB/flc.h robot.h include.h
        $(cc) CONTRIB/inference.c -o OBJ/inference.o

header.h: types.h graphics.h sim.h robot.h world.h
        touch SRC/header.h

clean:
        rm -f sim OBJ/*.o
```

The following section describes our computer simulations using the resulting FBM-FLC executable file.

3.6.3. Computer Simulations

Several computer simulations were needed (analysed) before establishing a heuristic verbalisation of our Collision Avoidance FBM. The first simulations were carried out with a crisp version of a control program based on the FBM described in section 3.3. Then it was observed that Khepera needed more complete statements in order to gradually avoid collisions depending on the degree of activation of the *perceiving_obstacles* node (as discussed in section 3.2). Finally, when the FBM-FLC executable file was built, several trials were also necessary to tune the first prototype of fuzzy control rules.

Figure 3.9 shows Khepera's main window and the graphical interface of our FBM-FLC program (displayed the simulator when it calls our *user2* source file and related functions which write fuzzy values to data files). This window shows Khepera robot placed in a world, the robot and, a graphical look-up table of I/O fuzzy data that we have built within Khepera system to show the results of our control architecture. Basically, this look-up table displays the progressive (step by step) performance of the FBM-FLC program. From one controlled movement to the next, it shows:

- the number of current simulation steps,
- the position of the Khepera simulated robot,
- the fuzzified distances and speed motor values,
- the levels of perception read from *R1* and,
- the fuzzy variations inferred by the Decision Making Logic Unit.

Later crisp motor speeds obtained when adding the output of the Defuzzification Interface to previously achieved motions can be observed in the plan view of the simulated robot (top-right side of Figure 3.9).

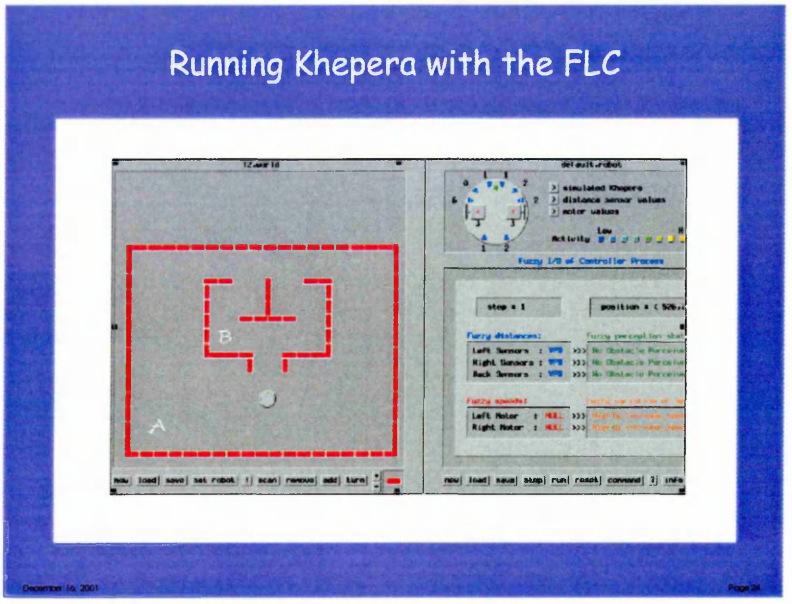


Figure 3.9: Running Khepera, Simulated Environment and Initialised I/O Fuzzy Data of the FBM-FLC.

In addition, our FBM-FLC program uses Khepera's facilities to plot the paths followed by the simulated robot after any desired number of steps and provides the following commands (specially useful for driving our pilot tests):

- reading a new set of fuzzy control rules,
- modifying fuzzy partitions and,
- displaying 3D views of control surfaces.

Our computer simulations driven with the FBM-FLC program were focused on two main objectives. First, testing the general fitness of Fuzzy Control techniques to develop collision avoidance behaviours. Second, investigating the fuzzy inference mechanisms (described in section 3.5.4) that would be more suitable for the simulated robot to optimally avoid collisions with stable obstacles. Our intention was that the robot should follow smooth parabolic trajectories rather than make inflexible angle turn movements.

The first goal was satisfactorily achieved even before the Sub-Rule Base *R2* was accurately tuned since Khepera was observed to avoid obstacles effectively. Regarding the second objective, we evaluated the shapes of the *paths followed by the robot within a number of environments* (worlds built on bricks).

We executed early experiments using simple worlds, not only to test our FBM-FLC program and to gradually fix an optimal set of fuzzy partitions but also, to tune the whole Rule Base (*R1* and *R2*) described in section 3.5.3 (Tables 3.2 and 3.3). Then we carried out our final computer simulations on the environment (world) shown on the left side of Figure 3.9.

Figures 3.10 to 3.12 show the results of some of our early simulations using an "empty world". Here we gradually fixed the fuzzy control rules set until observing that Khepera was able to avoid the wall of the environment while producing large right turn movements (following smooth parabolic trajectories). For all these simulations we placed the robot in the middle of the world.

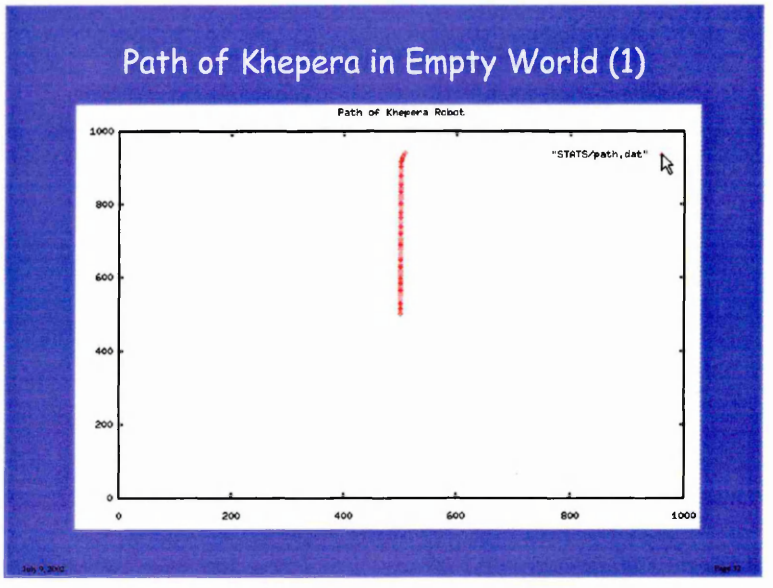


Figure 3.10: Running Khepera in an empty world (1).

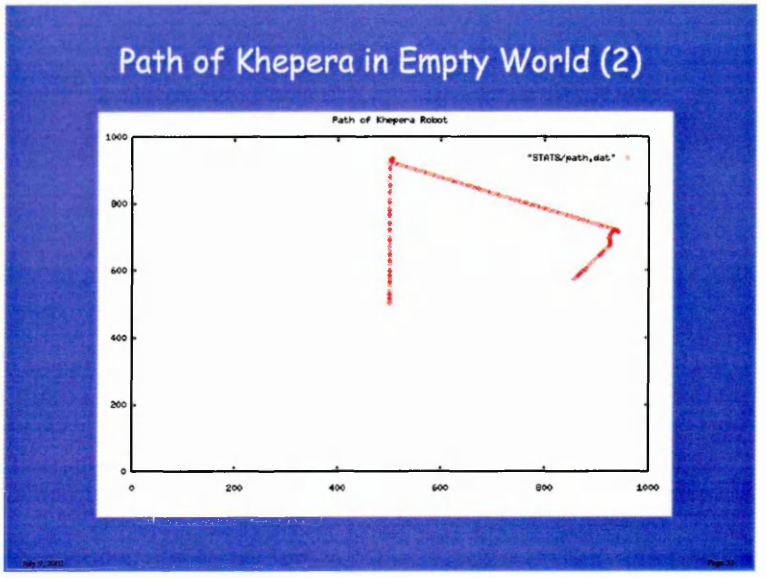


Figure 3.11: Running Khepera in an empty world (3).

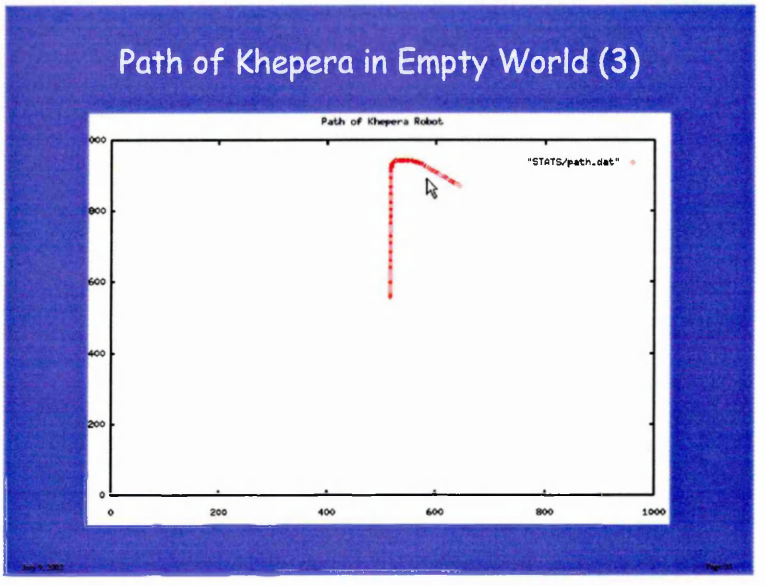


Figure 3.12: Running Khepera in an empty world (3).

The resulting (first) collision avoidance policy (mainly defined in *flc.h*) was achieved after several trials involving our FBM-FLC program built on Khepera system. Basically, we achieved a first *R2* prototype selecting the speeds at which the simulated robot was desired to avoid the wall and, modifying the control actions (fuzzy variation of speed) required to achieve those speeds.

Figures 3.13 & 3.14 illustrates the path followed by robot while we were trying to fix the first set of fuzzy controls rules (*flc.h* constants) so that the robot could avoid the two worlds of one of the top-left corners of the empty world. Basically, we had to train all the levels of perception (producing a first *R1* prototype) and slightly modify the first *R2* prototype until the robot was able to negotiate the corner.

For the experiment driven for Figure 3.13, we focused on turn right movements for avoiding the two walls of the corner. The path displayed in Figure 3.14 corresponds to a very parabolic turn left movements.

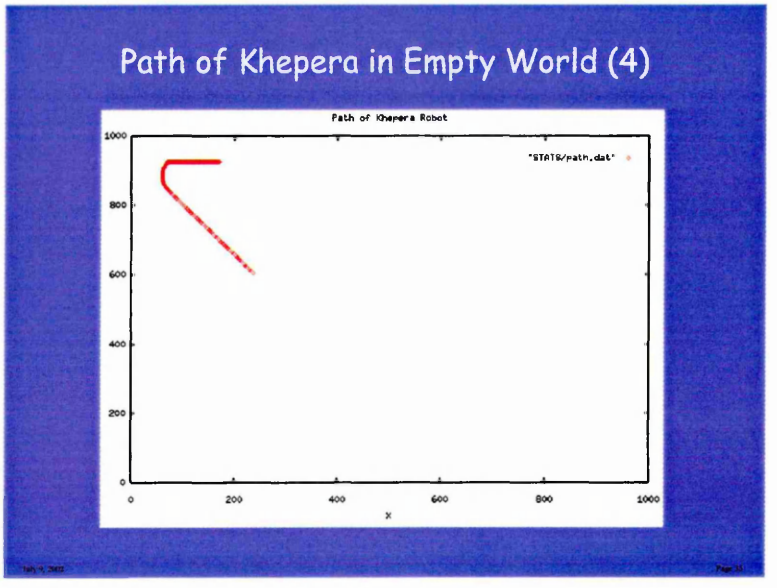


Figure 3.13: Running Khepera in an empty world (4).

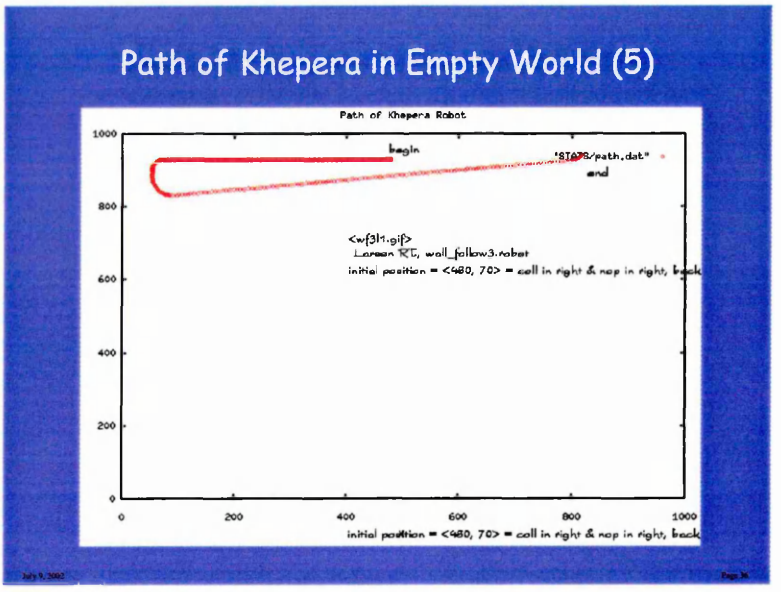


Figure 3.14: Running Khepera in an empty world (5).

3.6.4. Experimental Results

We built the *collision avoidance* environment shown in Figure 3.9 to offer diverse tests of our FBM-FLC architecture equipped with different types of inference mechanisms. Starting with the distribution of obstacles in the sub-environment A and finishing with the corridor B, we carried out around 50 pilot tests following four major steps:

- selecting the inference mechanism and installing this within the FBM-FLC program,
- initiating the FBM-FLC program within Khepera system,
- running the simulator until the robot was observed to achieve an exit of the corridor and, finally,
- saving and studying the shape of the path followed by the robot, taking note of the number of simulation steps needed for the current test.

For each selected fuzzy inference technique, our FBM-FLC program computes and stores the *firing strengths that evaluate the contribution of the fuzzy control rules* (the Collision Avoidance FBM) in the product space of the supports of *perception* and *speed* ($D \times S$). For each pair (do, so) with distance $do \in D$ and crisp speed motor values $so \in S$, Figure 3.15 shows the corresponding firing strengths αi obtained from $\mu Ap(do)$ & $\mu As(so)$ by applying the rule of inference related to one of the fuzzy inference techniques.

From this figure (3.15), it is clear that values $\alpha i(do,so)$ calculated by Mamdani's minioperation rule of inference (first type of fuzzy inference shown in sub-Figure 3.15(a)) differ significantly from those corresponding to Larsen's product operation rule (second type of fuzzy inference shown in sub-Figure 3.15(b)). Thus, for any particular environmental situation, the control actions inferred by the FBM-FLC may also differ from one type of inference mechanism to another.

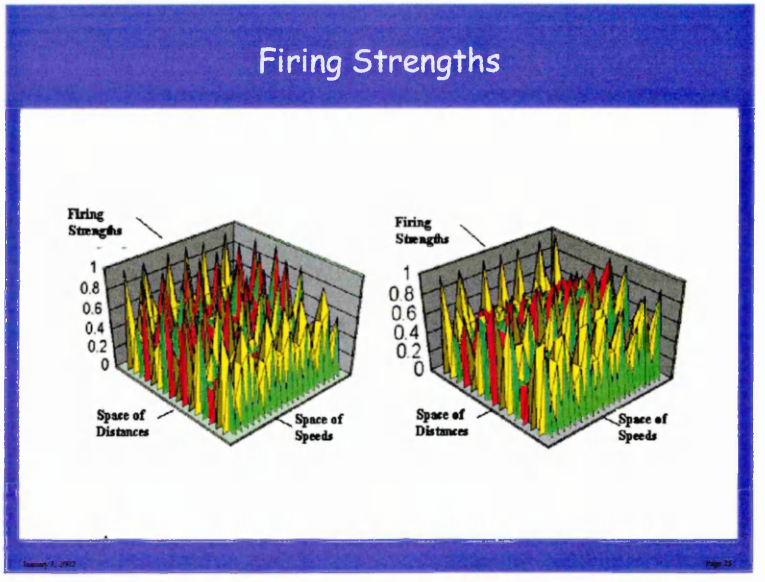


Figure 3.15: Firing Strengths of the FBM-FLC with (a) Mamdani's and (b) Larsen's Fuzzy Implications.

Some representative results of our pilot tests using (Mamadani´s and Larsen's) fuzzy inference techniques while Khepera was controlled in sub-environments A and B are shown in Figure 3.16 and Figure 3.17,

respectively. Basically, these pictures show that Khepera was observed to follow different paths when the type of fuzzy inference was modified within the inference mechanism of our FBM-FLC. This is due, as was commented before, to the differences among the firing strengths that contribute to the control actions inferred by the FLC.

In any particular simulation step, the fuzzy variation of speed computed for the next run of the system varies from one fuzzy inference mechanism to another. Therefore, the motions and resulting directions of movement are also different when adding the defuzzified variations of speed to the crisp motions corresponding to a previous simulation step.

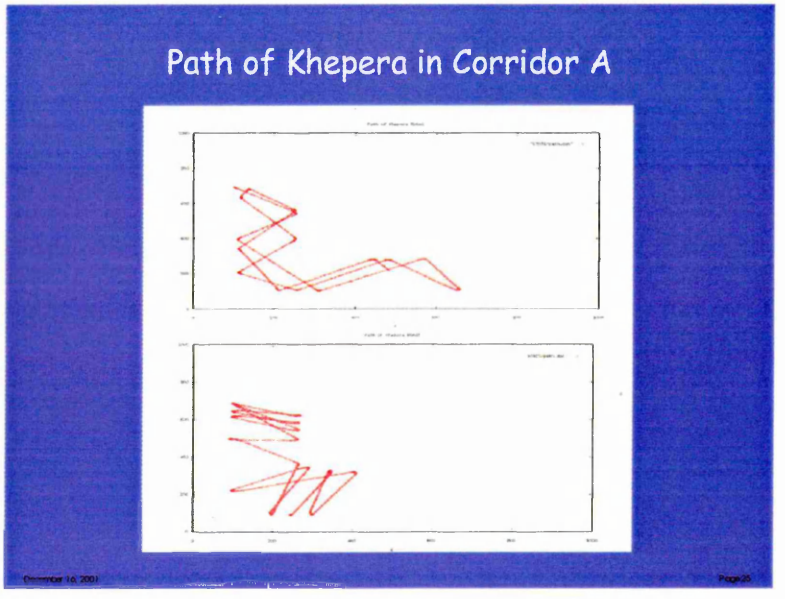


Figure 3.16: Paths of the Khepera robot in corridor A for: (a) the Mamdani's fuzzy inference technique, (b) the Larsen's fuzzy inference technique.

As Figure 3.16 shows, it does not matter where Khepera is located within the world before a control process starts. The final positions attained after several simulation steps differ from one mechanism to another.

The paths in Figure 3.16(a) and Figure 3.16(b) show the results for the robot initialised in the top-left corner of the corridor A and directed to move at 320 degrees to the motor axis.

In the case of Mamdani's minioperation rule (see Figure 3.16(a)), the corresponding firing strengths control movements whose directions vary between 35 and 40 degrees of the corridor walls. When any internal side-perception is built as "obstacle moderately perceived", the fuzzy variations produce large turn movements rather than U-turns (20 to 45 degrees) as happens with Larsen's product rule of inference. Thus, after a certain number of simulation steps, the first fuzzy inference technique allows Khepera to go further than when it is controlled by the second fuzzy inference technique (see Figure 3.16(b)).

The pilot tests carried out within sub-environment B are shown in Figure 3.17. We built this *world* to investigate the fitness of fuzzy inference techniques to allow Khepera to move through a T-corridor.

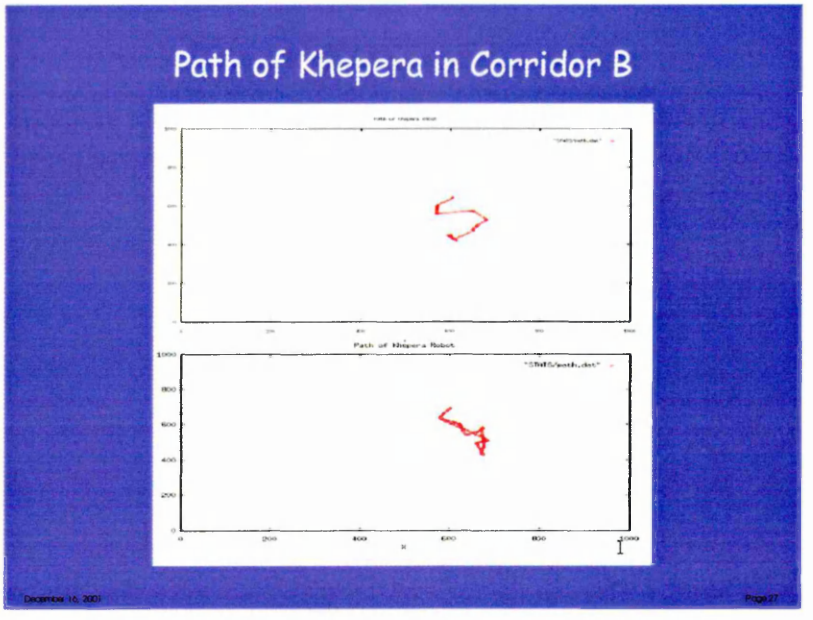
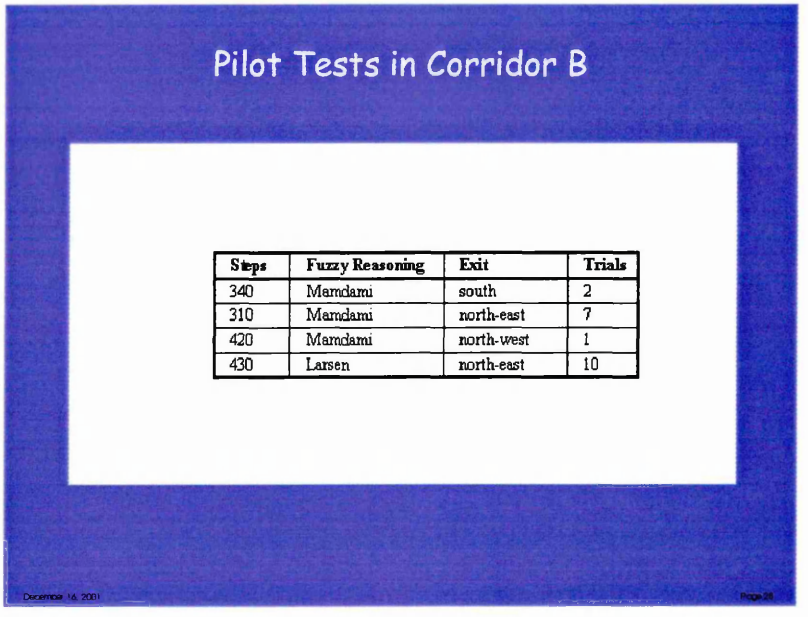


Figure 3.17: Paths of the Khepera robot in corridor B for: (a) the Mamdani's fuzzy inference technique, (b) the Larsen's fuzzy inference technique.

It follows from the previous analysis that the second inference mechanism makes the robot run into a wall almost immediately. Thus, the number of times that the levels of perception (derived from the Sub-Rule Base *R1*) are equal to "coll" remains, in the second inference mechanism, bigger than in the first. Higher fitness levels in a T-corridor deal with a lower number of collisions in order to allow a robot to negotiate away from the walls. This means, frequently following the corridor's middle line so that the robot can proceed easily down the sub-environment B. Therefore, according to this measure of fitness, the inference mechanism based on Mamdani's technique can be expected to perform at a higher level than the inference mechanism based on Larsen's technique.

Table 3.4 summarises the results of 10 runs of each fuzzy inference method. We have achieved all of them by initialising robot's position in the top-right entrance and pointing in a "south-west" direction (230 degrees).



Steps	Fuzzy Reasoning	Exit	Trials
340	Mamdami	south	2
310	Mamdami	north-east	7
420	Mamdami	north-west	1
430	Larsen	north-east	10

Table 3.4: Pilot Tests in Corridor B.

Note in the Table 3.4 that only 2 runs achieve the "south" exit of the T-corridor. Note also that both pilot tests correspond to a control program based on the first fuzzy inference techniques. An example of this behaviour is shown in Figure 3.17(a). However, most of the trials in Table 3.4 show Khepera's tracks going out of the T-corridor (B) through the same entrance selected for the initialisation of robot's position. This sort of performance can be seen and understood in Figure 3.17(b) for motion controlled by the second approach. In this case, and after the first 270 simulation steps, the mobile robot was observed to impact into the second wall opposite the entrance following a consequent turn back movement (160 degrees) to finally arrive at the "north-east" exit.

All these results for the sub-environments A and B appear to indicate that the FLC built on *Mamdani's inference technique behaves with a higher level of fitness than Larsen's inference technique.* Thus, Mamdani's fuzzy implication scheme seems to provide more reliable control actions than those conferred by Larsen's product operation.

The firing scheme derived from the intersection fuzzy conjunction of perception and motion (Equ. 3.14 & Figure 3.15(a)) ensures the avoidance of a higher percentage of collisions than those related to the algebraic sum (Equ. 3.15 & Figure 3.15(b)). Additionally, from one simulation step to another, there exists a smooth transition of membership functions induced by Mamdani's minioperation rule of inference. The evidence refers to control membership values (μAv_{left} & μAv_{right} from Equ. 3.16) stored and analysed by our control program. These control actions were observed to vary less strongly for Mamdani's minioperation rule of inference than in the case of Larsen's product operation. Thus, the resulting directions of movement acquired by Khepera allow it, most of the time, to follow the middle line of the corridor rather than bumping into the brick walls.

Figures 3.18 to 3.20 illustrate some other collision avoidance behaviours displayed by the robot involving the Mandani's fuzzy implication scheme.

For these last pilot tests, we initialised the robot in the down-middle side of the A sub-environment and directed it to move towards the down entrance of the T-corridor (see Figure 3.9). The final objective of these last runs was observing that the robot was able to negotiate the T-corridor and reach any of the top (right or left) exits. But our results where not very successful in the sense that the robot could only get any of the top exists in few runs (1 out of approximately 30).

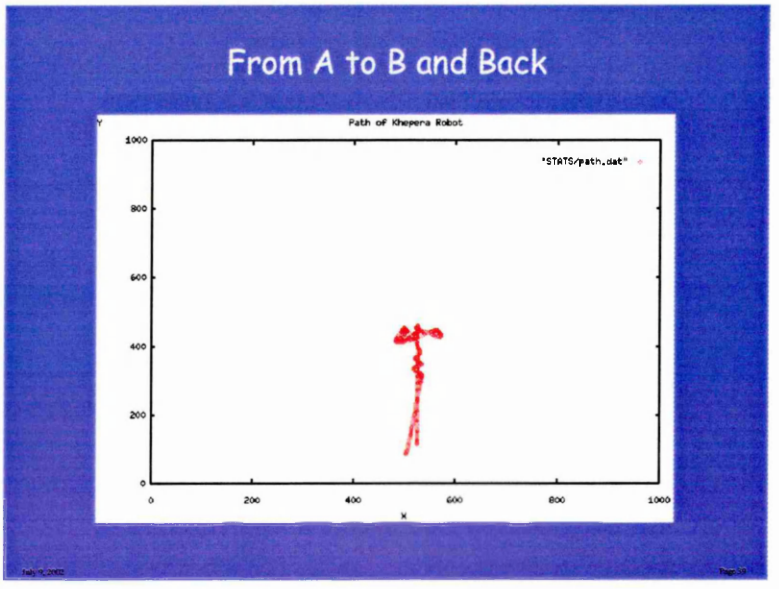


Figure 3.18: Path of Khepera robot passing through the T-corridor.

In Figure 3.18, Khepera reaches the entrance to the T-corridor and makes some rounds (collisioning into the walls) until it finally comes back passing through the same entry point. Figures 3.19 and 3.20 show a couple of runs where the robot was observed to move towards the exit points. In the first trial (3.19), the robot tracks going through the T-corridor to try to reach the right exit. By contrast, in Figure 3.20, the robot passes through the left side of the B sub-environment to finally reach the left exit.

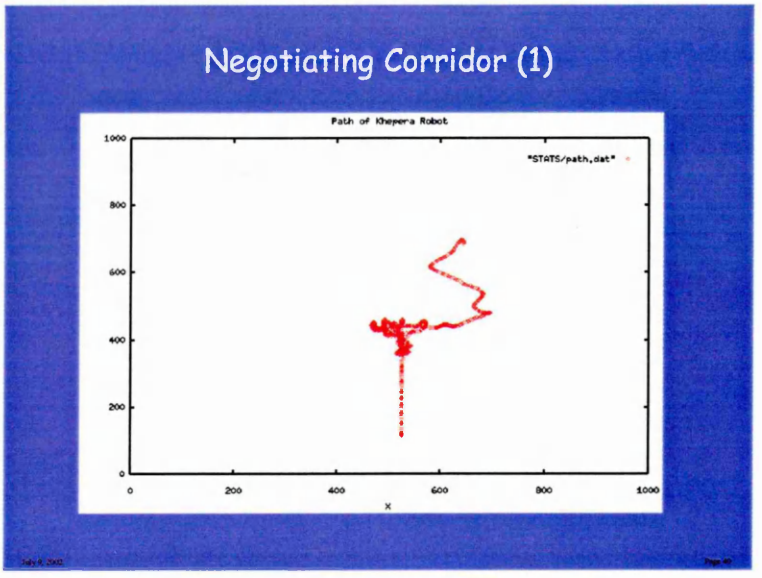


Figure 3.19: Path of Khepera robot negotiating the T-corridor (1).

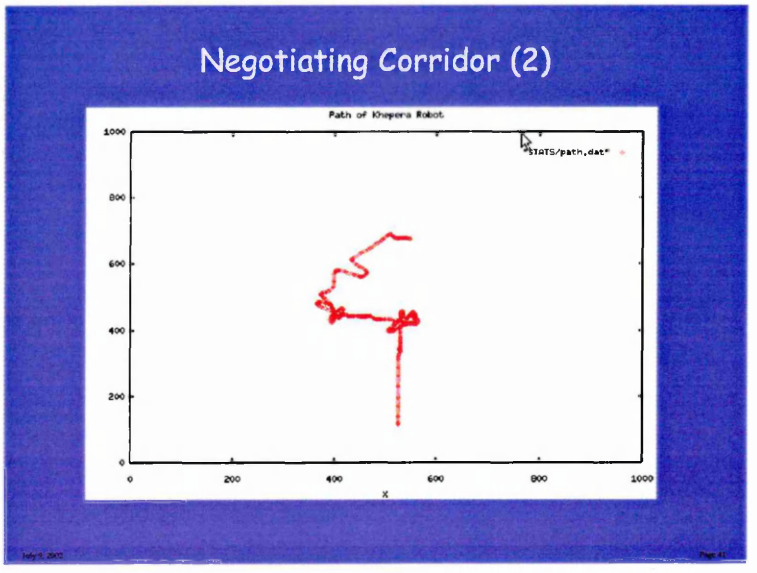


Figure 3.20: Path of Khepera robot negotiating the T-corridor (2).

Chapter 4

The FBMs Framework

Abstract

"As the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour dimishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics" (*Principle of Incompatibility*, by L. Zadeh).

This chapter presents a FBM Framework based on the use of more generic FBMs, *levels of abstraction* and *refinement stages* to develop BBSs. This work extends the definition and use of FBMs earlier discussed and, concentrates on the "development of large-scale BBSs" while introducing some more general ideas to solve the complexity issues outlined in chapter 1.

4.1. Generic FBMs

We can make FBMs driving *several interaction types*. We can add connection links to these maps as to represent the following two *models of interaction*:

- a *top-view model of interaction* between the levels of activation of the behavioural nodes and,
- a *lower model of interaction* between some internal components of the behavioural nodes and external elements/mechanisms.

The problem with this *type of FBM* is that it becomes necessary identifying *what the behaviours are* or at least, those specific components / parameters of the behaviours that should be closely related at a lower level of design. This is, we need *refining* the design of the structure and performance of the FBM (see further discussions in chapter 4). Though we can also be flexible in the way we address this. For example, we can extend the number and type of its nodes and causal interactions as to become useful tools for the modelling of more natural behaviours (e.g., animal behaviour)

Following the ethological studies outlined in section 1.2.2.1, living organisms are observed to perform different sort of behaviours according to their environmental stimuli and their internal (physiological and motivational) states. Then, once the agent accomplishes behaviour, the environmental and/or internal states change accordingly.

Using FBMs for modelling animal behaviours, we must refine the number and type of components of the behaviour producing modules (nodes) so that some of these (e.g., *actions*) become active and causally affected by the changes (e.g. *state transition*) of others. The causal flow of some behaviour modules can be fired and causally affected by other modules such as:

- *environmental stimuli* of the agent (e.g. IR sensor readings of mobile robots) that might become active once the agent starts interacting with its environment; they can engage the agent to perform the behaviours that they causally affect,
- *internal components/mechanisms* of the agent (e.g. physiological, motivational states or physical resources of its morphological set-up) that can also activate the FBM while determining which behaviours are more relevant in a given situation and,
- *resources* (e.g. energy levels of the agent, fuel, temperature, etc.) that might be required (and could be consumed) during the performance (accomplishment, execution) of behaviours or, during specific behavioural change.

These components or, their changes (e.g., state transitions) can directly cause the activation of a FBM and so, engage the agent to perform the behaviour(s) that it represents. Some of these components can also be considered as part of the internal causal flow of the map; they can increase/decrease the degrees of activation of the behaviours (at any instant in time). Further, these external components can also be affected by the FBM. This map can be designed to drive feedback connections to external conditions and / or implement possible consequences of behaviour activations (executions). This is, the flow of an FBM can be extended as to include (and drive) some feedback connections (and related computations) that allow changes on the components it is affected by.

In essence, what all these behaviour-oriented modules (sub-modules) and interaction processes tell us is that we can address the overall performance of the FBM through a number of stages that might include (but are not limited to):

1. Wait for *firing conditions* from external components and modify degrees of activation accordingly
2. Drive the internal causal flow until *terminating condition* is reached

3. Wait for *behavioural performance of the agent* ("behaviour execution" that could be part of the computations of the map)
4. *Drive feedback connections* to external components and modify them (execute "consequences of behaviours") accordingly
5. *Wait for the completion of all consequences* of behaviour execution and, return to 0.

During step 1, the FBM is in a resting position (state). All its nodes have the potential of being activated or, causally effected, by the external components that determine the *current situation of the agent.* Then, once *something happens*, it starts driving the causal links that its edges represent and keeps on swirling until a terminating condition (e.g. maximum degree of activation in one of the behaviours) is reached. In the next step (step number 2), the FBM waits for the execution of behaviour.

In practice, the designer of the FBM might map the behaviour nodes of the FBM onto specific tasks or, non-central control actions that allow the agent exhibiting the required functionality (e.g. we can model task-achieving behaviours). These computations can be time consuming and make the behaviour map stopping from further evolution. More importantly, all the processes must be completed in such a way that the causal flow and components of the FBM are appropriately changed.

Following [McFarland & Bosser-93], a transition in the internal state of an agent is a logical consequence of the accomplishment of a specific behaviour. This has to be reflected in the corresponding components of the FBM. Some behaviour nodes will be responsible for changing some specific parameters of the environmental stimuli (i.e., pushing an object). Finally, some resources of behaviours will be consumed during the step number 1, in time of interaction of the behavioural map with the environment of the agent.

For example, if a behaviour such as *moving around* gets maximum degree of activation (the agent performs this behaviour) it is necessary updating the FBM so that the level of a *fuel resource* is modified (decreased) accordingly.

The same sort of consequence might be necessary during a *transition stage* (not only when one or more behaviour modules are ready to perform some actions). Thus, the number and complexity of causal links will depend on the number and complexity of behaviours (and external components) that are designed using the FBM.

Figure 4.1 represents a generic (and dynamic) FBM with a *set of behaviours* (*b1*, ..., *b4*) which degrees of activation are causally related to a *set of external resources* (*e1*, ..., *e4*) considered during the design process of such behaviour producing modules.

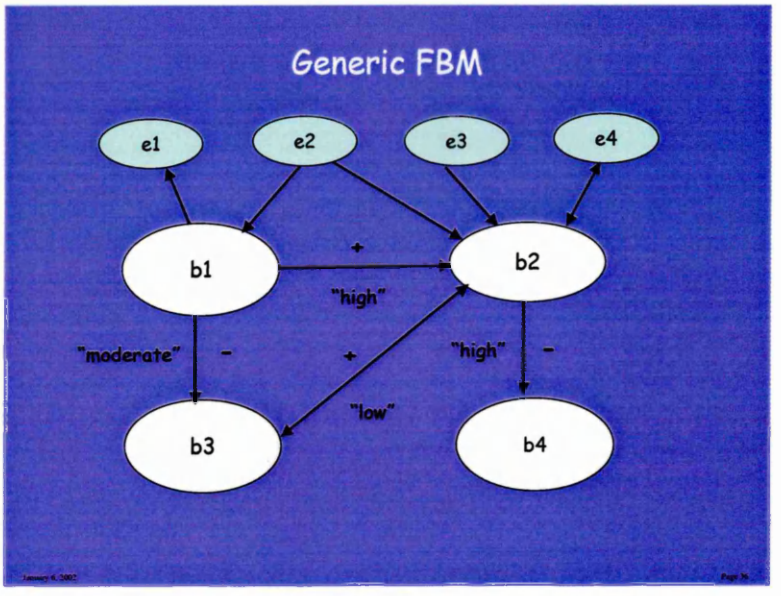


Figure 4.1: "Generic FBM": External Effectors, Internal Flow and Consequences of Behaviours.

Note that the strengths of the connections used in this map are *fixed* in the sense that they do not change when the map is running. Further modifications on the definitions of these (fuzzy) connections might be considered in a future work (when we try to accomplish the design and implementation of Learning FBMs). Some examples of these generic FBMs are described below.

Figure 4.2 illustrates a *top-view model of interaction* of a FBM that has been largely inspired by Maes' Action Selection Mechanisms [Maes-89a; Maes-89b; Maes-90a; Maes-90b; Maes-91]. A *low-view model of interaction* for this map is represented in Figure 4.3.

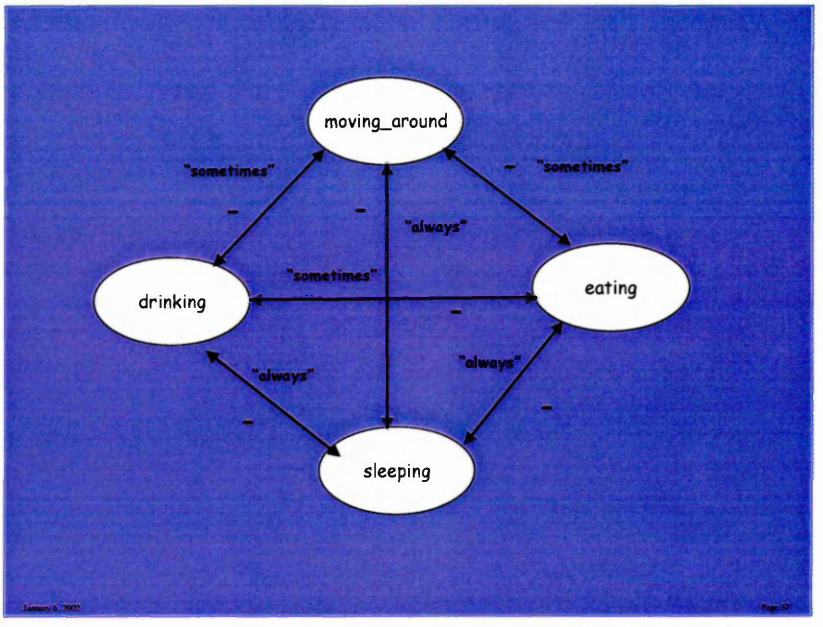


Figure 4.2: Top-View Model of Interaction of a FBM.

The high-level network (Figure 4.2) represents four behaviour nodes (*moving_around*, *drinking*, *eating* and *sleeping*) with negative causal connections among them (i.e. they cannot be displayed by the agent at the same time). However, we can be flexible in the way we design this FBM. In fact, we can refine its behaviour nodes down to a number of behaviour-oriented mechanisms.

Figure 4.3 represents one of these low-level mechanisms. Here the high-level behaviours have been decomposed into two types of components (*actions* and *external resources*) so that an agent can display a high-level behaviour by means of *action selection*. This low-level behaviour network illustrates an *action selection mechanism* as a continuous process. It takes place once the level of activity of an action reaches a maximum value (i.e. "1").

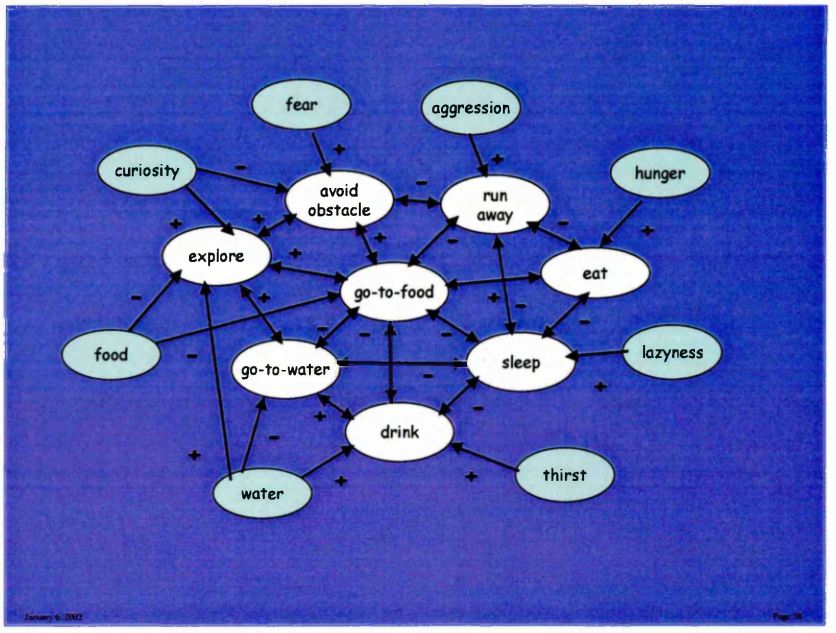


Figure 4.3: Low-View Model of Interaction of a FBM.

All the *actions* (white nodes) are connected using either positive causal links (meaning that they change their levels of activity in the same direction) or, negative causal links (meaning that they change their levels of activity in opposite directions). For example, as the degree to which the selection of the *go-to-food* action increases (decreases), the tendency to which the *eat* action can be selected increases (decreases) too. By contrast, if the level of activity of the *run_away* action increases (decreases), then the level of activity of the *sleep* action decreases (increases). Therefore, the agent can display a high-level behaviour (or another) by means of increasing (decreasing) the selection of one or more actions.

The *external resources* (green nodes) represent inputs to the behaviour network. These causally influence the action selection processes. More particularly, there are two types of external resources: *external stimuli* (from sensors of the environment) and *motivations* (derived from internal stimuli/mechanisms of the agent). The external stimuli correspond to *water* and *food*. The rest of the *external resources* (*curiosity*, *fear*, *aggression*, etc) represent motivations of the agent that can also influence the behaviour network.

The external stimuli become active once the agent starts interacting with its environment. They can engage the agent to select any of the actions that contribute to the high-level behaviours. The motivations also activate the action selection process but these are derived from internal stimuli of the agent. All these external resources influence the levels of activity of the actions. They can directly cause the action selection mechanism. For example, if the level of *food* increases (decreases) then the tendency to which the *go-to-food* action can be selected decreases (increases). Similarly, if the level of *water* increases (decreases) then the tendency to which the *drink* action can be selected increases (decreases) too.

Once this low-level FBM becomes active (some threshold is reached in any of the external stimuli and/or motivations), such activation is spread inside the network. Further, we can address the overall procedure through the following stages:

1. Calculate the level of activity of the external stimuli and motivations
2. Spread activation along all the action nodes
3. Let the most internal map (the one that communicates all the action nodes) evolve until some maximum level of activity (activities) is reached
4. If no action(s) is selected, repeat the cycle
5. Wait for the agent to complete action(s) and, repeat cycle.

4.2. The FBMs Framework

As we have demonstrated so far, we can develop FBMs using two major approaches. First, we can design a FBM and implement this using a FLC architecture (chapter 3). Second, we can design a generic high-level FBM and refine this down to a low-level behaviour network (section 4.1) that can be developed as any other BB Network (e.g. using ANNs).

Using the first development approach, we have designed a Collision Avoidance FBM at a *first level of abstraction* (identifying sensorimotor activities that help to perform collision avoidance behaviour) and mapped the fuzzy relations of this map onto fuzzy control rules. Further, we have developed a complete FLC to store the fuzzy control rules, fire these (using some flexible inference mechanisms also incorporated into the FLC architecture) and optimise the behaviour performance of the robot. This is, we have refined the FBM down to an *implementation level,* using some fuzzy inference mechanisms and related functional units (fuzzification and defuzzification interfaces, etc) that help to optimise the first abstraction.

Following the second approach, we have started with a high-level FBM and refined this down to an action selection mechanism (section 4.1) that can be developed using some principles found in BB Networks (section 1.4.1.2).

In this section, we go a bit further regarding the development of FBMs. We discuss how FBMs can be developed using many different techniques (not only Fuzzy Systems or, BB Networks).

The FBM Framework (development framework) that we present here provides new fundamentals about how to develop FBMs using two keywords *levels of abstraction* and *refinement*. Its principal scope is to help on the design of large-scale and complex behaviour-based models where dynamic elements and related cause-effect relations are needed. Further, this FBM Framework can be taken into the implementation details that are required to develop physical BB architectures like, for example, Subsumption Architectures [Brooks-86].

The following sections describe these ideas while introducing some technical aspects about how to develop FBM-based architectures and, more particularly, a complex multi-agent BB Systems based on non-hierarchical Subsumption Architectures.

4.2.1. Levels of Abstraction of FBMs

FBMs can be used to adopt abstract views of behaviours as to design very low level behaviour-oriented processes such as the sensorimotor activities. That is, we can start up the development process of a BBS with a high level abstraction of behaviour-oriented modules and their interactions (a first FBM) and, move down to lower level abstractions by a number of refinement stages (see Figure 4.4).

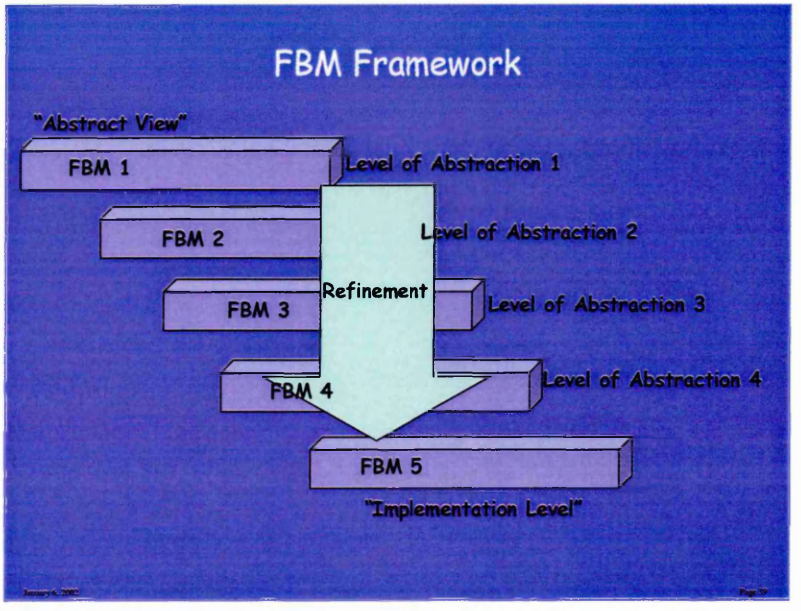


Figure 4.4: FBM Framework. Levels of Abstraction and Refinement.

The first level of abstraction represents a *qualitative net* of some behavioural nodes. It is a top-level, dynamic, and interaction-based model of *potential behaviours* based on the behaviour-oriented structures

that we have identified for abstract, simple FBMs. The complexity of this (first) FBM model depends on the decisions made by the designer. But it is important to emphasise the fact that it is not intended to be a physical model, namely, because this is the purpose of the refinement stages that characterise the framework.

An FBM is abstracted from the selection of behaviours and the first (top level) interactions among these. The designer does not need to worry about what the behaviour producing modules should be (i.e. a goal-attainment modules) or, what the agent is equipped with. Most BB Systems are designed so that the morphological set up of the agent and the control actions (that are required to perform the required functionality) are identified first. This is not considered using FBMs as abstract views of behaviour systems. The morphological set up of the agent(s), control actions and other *parameters* can be identified in the latest stages of development (once the FBM is targeted for a specific creature and/or control situation).

The abstraction of environment-related mechanisms can also be avoided to some extent. The environmental resources can be used to see how to activate the behavioural nodes of the map. Environmental conditions effecting on the degree of activation of these nodes might also be identified at lower levels abstraction. Some of these can be treated as constants over certain periods of time. Other environmental conditions might be dynamic ones (processes / mechanisms).

In summary, the development framework that we propose uses abstractions and refinement stages to find out those behaviour-oriented parameters that are relevant for the activation of behaviour-producing modules and their causal interactions.

4.2.2. The Refinement of FBMs

We exploit the mapping of FBMs onto BB architectures searching for the realisation of *refined behaviour-oriented systems* where it is possible (in latter stages of design) identifying node strings with behaviour-oriented commands that in turn might control specific components of an agent (or, external resources).

Therefore, we use this refinement of FBMs to identify lower design parameters (rather than fuzzy sets or variables as it was to the fuzzy implementation provided in chapter 4 while helping on the definition of the behaviour-oriented processes identified at first levels of abstraction.

The nodes and edges can change from one level of abstraction to another. Indeed, once one behavioural node and its interactions with others have been refined, it might occur that the structure changes completely. One node might be divided into a number of sub-nodes. One connection link might also be divided into different links adding new qualitative and quantitative information. Further, the changes in the connection links might come from the identification of either *co-operative or competitive behaviours* (their activities change in either the same or opposite directions).

The behavioural interactions are refined so that these become either implicitly or explicitly specified. In implicit specifications of behavioural interaction, the activity of one behaviour increases or decreases the activity of another. In the explicit description of behavioural interactions, information exchange and, parameter passing mechanisms is taken into account. This last refinement has been inspired by the work presented in [Brooks-94].

Further, the refinement is based on different criteria. The most obvious refinement process is based on identifying *basic abilities* that are relative to the selected behaviours. For example, if we want to design a "seek for light" behaviour, it is necessary refining this behaviour into basic tasks such as "moving_around", "light identification" or, "detecting the proximity of the lights". The ability to move around is independent of that of identifying the light sources. The agent needs to perform both at the same time. However, if the agent strongly identifies a light source, it should check the proximity of this. In fact, a detection of the light might cause the measurement of distance to this. Then, as the detection of proximity to the light sources increases (it is very close to the light), the agent should decrease the activity of moving around. Therefore, these two abilities might change in opposite directions. Similarly, if the agent strongly identifies a light source, it

should stop moving around and make a movement a head (rotation angle). This provides us with a final FBM that we can try to implement with some sort of sensorimotor connections and control actions.

Another possible refinement is that based on *states of the agent*. Notice that this state-based refinement gives us a final FBM that could be implemented using Finite State Automata techniques.

Finally, we can use a *neural-based refinement* identifying the neural structures (sub-systems) that are associated to the selected behaviours. This would be the most biological approach to refine FBMs.

As we can see, the FBMs can be easily scaled up and refined so that it becomes a large-scale system where numerous interaction processes can take place. The final level of abstraction that is achieved can be ready for implementation or not.

The fact is that only when deciding for the implementation level, the map might be able to operate so that the behaviours become activated and operate according to the interactions that have been designed. The most general FBM-architecture, as suggested earlier, can also incorporate some external modules that might be behaviour-independent.

4.3. An Application Case Study

This section provides some initial steps about how to use the FBM Framework to derive a complex, multi-agent and non-hierarchical BB System presented in [Mataric-92b] (largely inspired by Brook's Subsumption Architecture).

Basically, the Subsumption Architectures [Brooks-86] are developed in an *incremental fashion*. These BB Systems are built by means of implementing basic behaviours (as layers of agent competences that couple sensing and acting capabilities) and aggregating new ones (based on the same design principles). Each behaviour unit (identified first) hides low-level design details. It is designed to drive some control actions

from the sensors to the actuators of the agent while being able to stop other behaviour units from performing any computation. The overall system (control mechanism) is based on parallel operations and subsumption relations. Priority schemes are used so that it is possible to arbitrate which behaviour unit (control layer) can route its outputs to the actuators of the agent.

By contrast, using the FBM Framework, we start at a high level of abstraction (with an abstract view about behaviours and their interactions) and try to design low-level behaviour-oriented processes (e.g. sensorimotor activities) that can be implemented using, for example, the strengths of the behaviour units that characterize the Subsumption Architectures. The main idea, as described in section 4.2, is following as many refinement stages as required to complete the development process.

[Mataric-92b] uses Brook's approach (the Subsumption Architecture) to develop a multi-agent BB System. This author builds five basic behaviours (*safe-wondering*, *following*, *dispersion*, *aggregation* and *homing*) as building blocks for complex group behaviour in a multi-agent system. Basically, the development process that is followed in [Mataric-92b] consist of:

- *selecting* the basic behaviours (control laws) following some special criteria (i.e. being able to generate the required complex group behaviour),
- preparing *formal specifications* for all the basic behaviours (using position, distance and threshold parameters),
- *designing appropriate algorithms* for each basic behaviour and *implementing* these,
- *evaluating* all the basic behaviours (optimisation)
- combining all the basic behaviours (using some kind of combination operator) so that the complex group behaviours can be displayed by all the agents.

The basic development process that we follow to derive a BB System that is able to display the same target behaviours (group behaviours) using the FBM Framework can be summarized as follows:

- *analysing* the target behaviours (group behaviours)
- *abstracting* the target behaviours and high-level interactions among these (first FBM based on group behaviours and causal influences among these)
- *refining* first FBM until implementation details of each basic behaviour can be specified (second FBM with more refined group behaviours and causal influences among these)
- *designing appropriate algorithms* for each basic behaviour and *implementing* these,
- *evaluating* all the target behaviours (optimisation)

[Mataric-92b] investigates four group behaviours that are mutually exclusive. Figure 4.5 represents a first level of abstraction for these mutually exclusive behaviours. Following the experiments reported in Mataric-92b], *flocking* behaviour can be designed, as a direct combination of *safe_wandering, aggregation* and *homing* basic behaviours. In our FBM, direct combination means that all the basic behaviours must have an appropriate level of activation to reproduce the high level one. *Foraging* is initiated by *dispersion*, then *safe_wandering*. This temporal composition requires a positive influence between these two basic behaviours. *Surrounding* can be obtained from a direct combination between *aggregation* and *following*. *Herding* can also be displayed from a direct combination between *flocking* and *surrounding*.

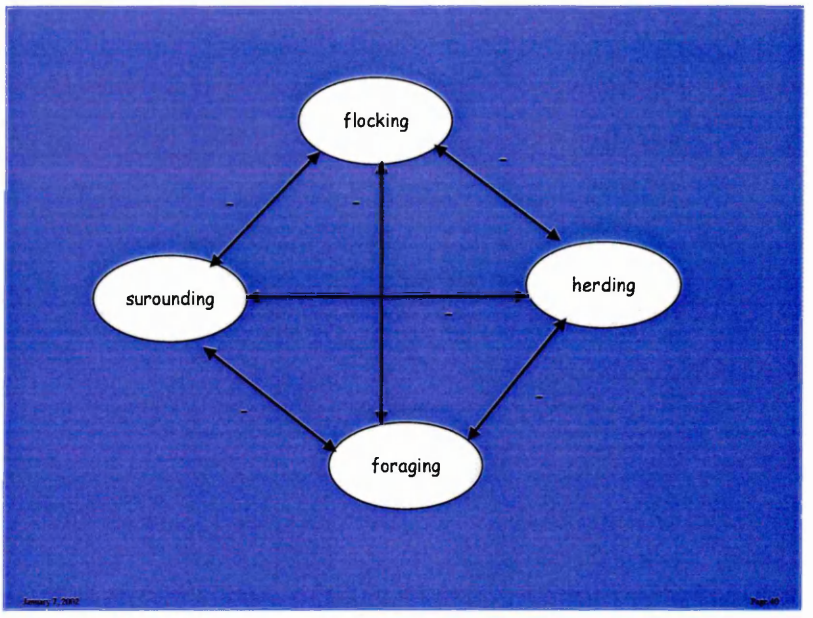


Figure 4.5: First Level of Abstraction.

The following FBM (Figure 4.6) represents the resulting second level of abstraction.

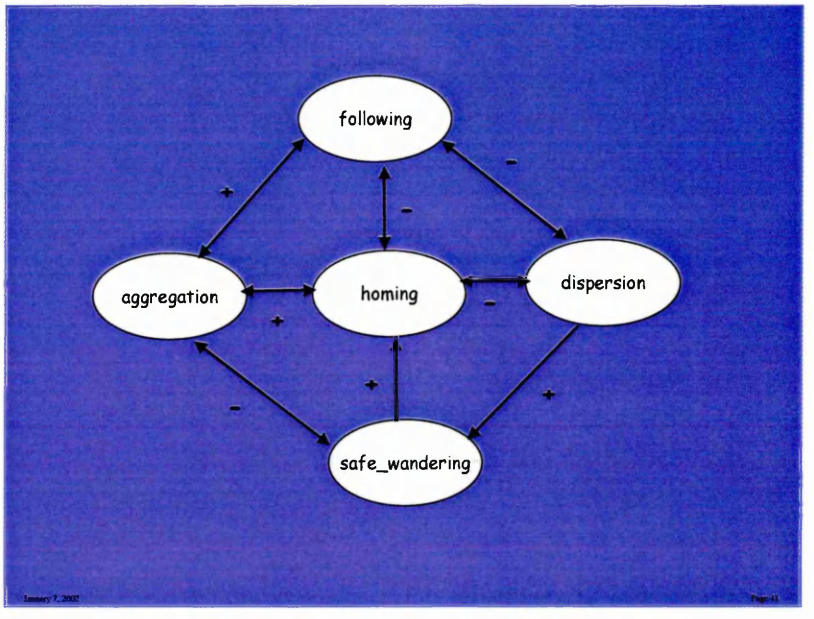


Figure 4.6: Second Level of Abstraction.

Figure 4.7 represents a final abstraction that can be taken into implementation details. Here the low-level behaviours (represented in Figure 4.6) have been decomposed into some actions that can be specified in terms of control parameters (distance, position and threshold values). Next step is using the formal specifications and algorithms described in [Mataric-92b]. Complete experimental results must be obtained for prototyping more behaviour networks. This evaluation process helps optimising the lowest-level of abstraction in terms of agent competences and interactions. New, high-level behaviours can be integrated at any of the levels of abstraction that have been obtained.

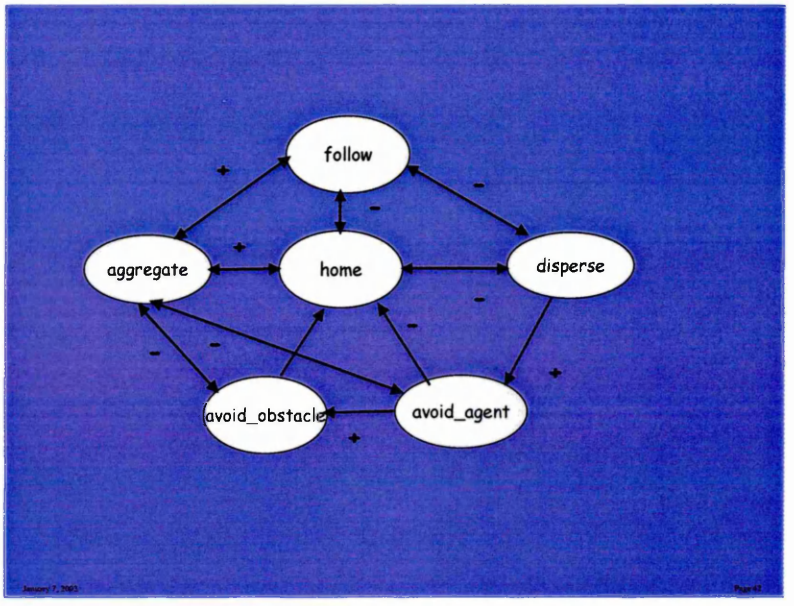


Figure 4.7: Third Level of Abstraction.

4.4. Complexity Considerations

This section summarises the most general ideas explored in the thesis to cope with some complexity issues (discussed in chapter 1) using the FBM Framework.

4.4.1. Abstraction and Description of Behaviours

The nature of behaviours is not well understood. It is difficult to abstract and describe what is observed in natural environments. The ideas summarised here refer to some clues about the way behaviours could be modelled through abstraction, description and organisation mechanisms.

Multiple, Heterogeneous Abstractions

We use levels of abstraction to produce behaviour-based models in terms of: behaviours, their causal interactions and, some possible external and/or internal mechanisms (generic FBMs). Complex behavioural dynamics could be modelled through the identification of these heterogeneous components. But not all behaviours can be abstracted in the same way; they might not be associated to the same neural control structures. It is necessary focusing on different abstraction levels.

Implementation Details at the End of the Abstraction Processes

The design process presented in this dissertation consists of identifying a number of refinement stages leading to different possible levels of abstraction that should be tested before actually moving into implementation details (e.g., control parameters).

No matter, for example, if part of the design process is automated using ANN models and their learning algorithms or, any other implementation technique. There will also be decisions such as *number of neurons* or *which inputs might fire the activity of a particular behaviour* that must be carefully considered before moving any further. Especially in the case of complex behaviours that are not well understood (cannot be observed).

Goal Oriented Abstraction is not Necessary

Following [Maes-89b; Maes-92a], goal-directed behaviours can be designed without need of goals or, copies of the world [Pfeifer & Verschure-95]. The FBM framework does not introduce any goal-based design.

External and Internal Factors

As we discussed earlier (in chapter 1), BB AI rejects the use of conventional world modelling techniques. This rejection differs from one approach to another. For instance, in using Dynamical BB Systems, there is a two-component dynamic system (agent & environment). In the application of Action Selection Mechanisms, on the other hand, the environment seems to be characterised as a set of external conditions that switch on the behaviour network. Thus, even if the environment is not engineered, it seems to be analysed and described in such a way that some information is passed onto the behaviour system. This is the type of external component that we have introduced in the generic version of FBMs. It is some sort of useful condition for designing stages (first levels of abstraction) that might not be implemented when the map is built on an agent.

Simplicity

Using multiple abstraction levels does not mean that we try to complicate things. Simple neural-based systems can be developed to drive the behaviours and, without need of complicating the abstraction of the system by introducing *intentions*, *goals*, *motivations* and the like. Biologists tend to complicate the description of behaviours and the way these are linked to neural activity. Some of them assume that most behaviours of animals need of *intentions* (i.e. recognition and goal-seeking) that might be *located* somewhere in their nervous systems. But they cannot find such intention-like neural activities maybe because they do not exist [Rowe-98].

No Abstraction of Functionalities

This is emergent from the performance of the FBMs that have been introduced. Following [Brooks-91b], intelligent functionality might emerge from the interaction of the behaviours.

Analysis and Descriptive Levels

We need to analyse and try to describe the behaviours of living and artificial organisms. The analysis of behaviours is more close to the Nature side than to the engineering objectives [Mataric-94a]. We analyse behavioural dynamics and try to abstract this using the nodes and edges of FBMs. The engineering part does not need to take place until we arrive to an implementation or, control level (the lowest level of design).

Multiple Neural Controllers

Some behaviours cannot be associated to one single neural control systems; it might be necessary designing the same behaviour-producing module from the use of more than one neural controller.

Descriptive Variables Cannot be the Same for Animals and Robots

We cannot characterise all behaviours with on single set of *variables*. Following [McFarland and Bosser-93], animals have time-dependent physiological states and their brains govern their behaviours according to these states. But robots have no physiology as such. They can have some state variables (e.g., fuel supply, operating temperature, etc.) that might be influenced by their own behaviours.

4.4.2. Organisation, Activity and Execution of Behaviours

No External Organism Scheme

We try to drive the interaction among the behaviour-producing modules and, more particularly, the cause-effect links among their levels of activity. We tend to avoid driving the *selection of these modules* from an external organisation scheme.

The FBM Framework lets the designer of the BB System establishing the functionality of the behaviour-producing module in such a way that no external mechanism is required to control the selection of the behaviours. As described in chapter 3, each behaviour-producing module controls the actions of the agent (robot) having its own control mechanisms. This implementation technique (based on the use of FLC implementation parameters) lets developing behaviour modules that are independent in terms of low level actions and control mechanisms.

The same implementation technique can be applied so that several behaviour maps (FBMs) are causally communicated to influence the level of activity of each other without need of adding any central reasoning unit (special unit to select the behaviour that has to be displayed by the creature).

Not Static Sequencing

Activating what and when? In KB AI, the activities were designed as a result of a deliberative thinking process developed by a central system (engine). Most approaches to develop BBSs tend to reproduce sequences of behaviours. Independent (self-contained) behaviours can be activated one at a time so that there is only one which drives the control actions to the actuators of the agent. However, living organisms are not observed to follow sequences of behaviours.

Flexible Interactions

Some approaches use interaction links among the behaviour-producing modules. For example, in Action Selection Mechanisms, a behaviour can activate / inhibit other behaviours. The higher level behaviours of a Subsumption Architecture also subsume (inhibit) other lower level ones. FBMs incorporate causal interaction links. Using FBMs, we try to address causal flows of behavioural activities; flexible combinations of behavioural activities where more complex behaviours can take place from the causal-effects of others. [Hercock & Barnes-96] implements an FLC to give negative feedback to the behaviours of the architecture. This dynamically adjusts some contributions of the behaviours incorporated into their architecture. In this way, the robot is said "to focus its response". Similarly, FBMs implement both positive and negative feedbacks between the levels of activities of the behavioural nodes.

Scaling Up the System

New behavioural nodes can be added on an FBM either in first levels of abstraction or, through refinement stages. We can augment the FBMs in a similar fashion than FCMs i.e., the FBM can scale up to become a large scale model. This represents an incremental approach to design the behaviour-oriented system.

Degrees of Activity and Situations

The level of activity of one behaviour must be a matter of degree. It depends on a number of behaviour-oriented components and interaction processes; it cannot be just either activated or not. Fuzzy granularity seem to be appropriate to design these degrees of activation. Additionally, the activation of one or more behaviour must be relevant to the situation of the agent and, this situation cannot be designed (cannot be incorporated into the FBM).

Execution, Control and Resting

Once we move onto the implementation level, the behaviour activities might be mapped onto specific control actions i.e., a behaviour strategy associated to each particular node might be executed so that the agent can act accordingly. But not all levels of activities of a behaviour node need to be associated to the same control actions. Some behavioural activities imply *resting* or very little actions.

4.4.3. Evaluation and Optimisations

Evaluation of Behavioural Activities: Fitness Functions and Decision

BBSs have to be reactive. Following [McFarland & Bosser-93], the behaviour-producing modules should extract information from the environment and from their interaction with other behaviours. Consequences of behaviours (lower level actions) should be evaluated by the agent before it actually decides for specific behaviours. It is difficult to see, then, that each behaviour has its own control parameters and decides on its. Hierarchical models of behaviour decision processes do not seem to be appropriate either [McFarland & Bosser-93]. Living organisms behave in such a way that certain behaviours are more relevant than others depending on the environmental situations they encounter. They do not decide for certain behaviours because these are more beneficial. They decide depending on many different factors and these might not be resolved from centralised decision units. The decision making processes have to be able to balance conflicting situations. All behaviours must have the potential of being activated and focus on the experience of the agent [Maes-92a].

Optimisation and Learning

The performance of the behaviours has to be also considered when these become activated. This is strongly related to learning processes.

4.4.4. Technical Aspects to Design and Implement the Systems

Refinement

This is the key aspect we consider to design large-scale BBSs using FBMs. In KB AI, refinement was used to translate plans (provided by plan programs operating on symbolic representations) into atomic plans. We look at refinement stages to help on the design and computational/physical implementation of behaviour producing modules and their cause-effect interactions. Doing so, we intend to find relevant parameters (variables) that can describe FBMs as dynamic models to control the behavioural activities of the agent.

Asynchronous and Parallel Models of (Dynamic) Systems

In parallel architectures, all behaviour-producing modules operate at the same time. Their levels of activity are increased/decreased at the same time. Parallel interaction of behaviours requires all connection links providing input signals (to the effected nodes) at the same time. This allows for faster adaptation [Maes-92b].

Useful Mechanisms

After the behavioural map has been refined, sensor signal accumulators and motor action accumulators can be shared by different behaviours. These sensorimotor mechanisms can be seen as useful mechanisms for behavioural performance.

Computational Requirements

The key principle is KISS. However, this cannot be taken too far, specially, when the behaviours being designed and implemented are too complex (cannot be easily analysed).

Chapter 5

Conclusions

This document has presented a thesis on the definition, exploration and evaluation of Fuzzy Behavioural Maps providing the following contributions to knowledge:

A) The Investigation into the Behaviour-Oriented Nature of Intelligence

The thesis has explored many investigations around the behaviour-oriented nature of intelligence. It has been focused on the problem of building behaviour-based models and identified many research issues of philosophical, biological and technical nature that need to be resolved in order to validate the BB approach.

However, the thesis has not been intended to discuss the philosophical and biological questions around the behaviour oriented views of intelligence. It has been assumed, following [Brooks-91a], that discussion is such points is "untimely unless the technical ones concerning the viability of the behaviour viewpoint can be resolved". The thesis has been concerned with the technological part of this viewpoint and identified those technical issues by classifying the work on the development of BBSs (presenting the classification of approaches to the development of BBSs) and bringing to knowledge the major problems found in the application of these approaches.

The fundamentals of the BB approach have been explored in terms of the models, systems, techniques and tools being investigated by a wide variety of research communities. Many features of the behaviour-based models have been identified as well as some practical ideas for building these models with BBSs (e.g.

designing and implementing self-contained behavioural modules, using distributed and open architectures, using bottom-up development methods, not using world models, etc.). A broader view on the behaviour-oriented nature of intelligence has been identified with the multi-disciplinary groundwork of AAs. Indeed, the thesis has explored some AAs tendencies (e.g., some cognitive, biological and technological ones) that influence the thinking in BB AI and help to consolidate this approach by re-defining the ideas, concepts and methods firstly introduced by the researches of BB AI and AR.

Further, the investigation has outlined some key principles about the development of BBSs and presented a classification of the current approaches to development these systems (Subsumption Architectures, BB Networks, Dynamical BB Systems, Evolutionary Approaches and Hybrid Paradigms) based on some development decisions raised with the proposed FBM Framework. Finally, the major technical problems found with the application of these approaches (to develop the BBSs) have been discussed to conclude the survey of the investigations around the behaviour-oriented nature of intelligence that have motivated the definition and use of FBMs.

B) The Definition, Exploration and Evaluation of FBMs

The thesis has presented several types of FBMs of varying degrees of complexity and established some of their uses within the BB viewpoint i.e., as designing tools of behaviour-based models. Simple FBMs have been introduced as directed graphs which nodes represent behaviour-producing modules the designer is hoping to implement within an AA; behaviour networks drawing pictures about the behaviours of AAs and how the firing degrees of these can be causally related with symbolic edges showing either positive or negative cause-effect relation. More complex FBMs have been defined as dynamic behaviour-oriented networks that incorporate feedback cause-effect connections with behaviour-activation patterns being propagated until some terminating condition is satisfied.

The most generic FBMs have been defined as networks divided into behavioural nodes that can be casually affected by external components relative to the activities, morphological set-up and environmental situations of the agent. Further, it has been suggested that time-dependent changes of variables (processes) feeding FBMs can be considered as conditions that activate the operation of these maps (not a world mode, of course). A specific behavioural node can be executed making the agent to experience natural consequences such that other behaviour and/or internal resources can become also affected. That is, the use of FBMs has been extended as to implement possible consequences of the execution of the behavioural nodes such as the modification of the causation of the behaviours (e.g. increasing, decreasing or producing the state transition of environmental stimuli, internal states and/or the resources that feed its nodes).

Finally, the explorations around the use of these FBMs have shown the differences among the design and the implementation of FBMs. An FBM (either simple or complex) serves to design behaviour-producing modules. Then, it is possible using any suitable implementation technique (e.g., FLC) to map the FBM (set of FBMs) onto a functional architecture.

C) The Collision Avoidance FBM: Development, Pilot Tests and Results

The work presented in chapter 3 has shown how to use FBMs to design an avoidance behaviour-producing module based on sensorimotor connections for a mobile robot. Our final intention of this practical work was to obtain a proof-of-concept of simple FBMs (i.e., to demonstrate that these maps can be used to design behaviour-oriented systems) but this has also served us for three major purposes. First, to demonstrate that FBMs can be implemented using Fuzzy Logic Control parameters (e.g., fuzzy sets, linguistic variables, fuzzy implications, fuzzy inference mechanisms and others). Second, to show how FBMs can be mapped onto functional control architectures such as FLCs. Third, the work has shown the flexibility of the resulting BB and Fuzzy architecture.

This work has presented several fundamentals of mobile robots and Control Theories including some studies on perception, motion and, the basic avoidance behaviour to be exhibited by an architecture based on the current Behaviour-Based Robotics. It has been focused on the use of FBMs, Fuzzy Logic Control Theory and related concepts from Fuzzy Logic (its mathematical framework).

The FBMs have been used in combination with FLC techniques to control the avoidance behaviour of mobile robots. A simple FBM has been used to design an avoidance behaviour-producing module while an FLC architecture has been implemented and tested to drive the functionality of the map and, more particularly, to fire the level of activation of its nodes and edges (converted into fuzzy control rules).

The use of this (fuzzy) methodology has being presented for implementing and testing an avoidance strategy designed with the FBM and fired within an FLC using two different types of fuzzy inference techniques: Mamdani's minioperation rule of inference and, Larsen's product operation. The control policy has been obtained through heuristic verbalisation and analysis of the dynamic features of the FBM. This map has been divided into two sub-units and converted into two sets of fuzzy control rules manipulating the levels of perception and fuzzified motor values of a simulated robot. Levels of perception have been treated in the form of fuzzy disjunction relations of fuzzified sensor signals. Fuzzy partitions have been designed and implemented for manipulating both fuzzy sets of sensor and motor signals plus fuzzy control actions in terms of motor speed variations.

The flexibility provided by this FBM-FLC architecture has been analysed through pilot tests and carried out using computer simulations. These have revealed that (in simulation) a mobile robot can be able to follow different types of trajectories depending on the underlying formalism of the inference mechanism and without even modifying the fuzzy control policy (the FBM) stated at the beginning of the development process.

Regarding the accuracy of this FBM-FLC architecture and, from the measurement of the contribution of its Rule Base, the pilot tests have shown a higher level of fitness for the construction of firing strengths

(contribution of fuzzy control rules) using Mamdani's minioperation than provided by Larsen's product operation.

D) The FBMs Framework: Definition and Application Case Study

The thesis has presented our initial (first intuitive) ideas around the Framework of FBMs and the development of large-scale BBSs, assuming that it is necessary identifying different levels of abstraction and refinement stages to cope with the technical complexity issues found with the current approaches to develop BBSs.

In chapters 2 and 4, we have described how the nodes and external components / mechanisms of a simple / complex FBMs can let the designer establishing a number of interaction processes that are observed with natural behaviours. The thesis has introduced the concept of *generic FBM* to mean that, FBM-based architectures can incorporate a number of internal and/or external behaviour-oriented modules and causal links whose number and complexity depend on the morphological set up of the agent(s) as on the desired behavioural performance.

The proposed Framework of FBMs has been defined to capture and model behavioural dynamics (behaviour producing modules and their internal/external interactions) at different levels of abstraction (through different levels of refinement). Its major scope is the design of dynamic BB networks. Most obviously, this framework maps some required behaviours into connection structures of behaviour-producing modules that are causally linked.

Here the concept *refinement* is used to identify lower design parameters (i.e. control actions) rather than linguistic variables or, fuzzy sets (as it corresponds to a fuzzy implementation technique like the FLC described in chapter 3). This refinement helps on the definition of the behaviours selected from *first levels of abstraction.*

The thesis has also proposed taking the FBM Framework into the *implementation levels* that are required to build (derive) behaviour-based control architectures like SAs [Brooks-86]. Indeed, the framework has been intended to allow the refinement of FBMs (from one level of abstraction to another) for identifying lower level behaviour-oriented design parameters (instead of the fuzzy implementation parameters identified in the mapping of the Avoidance FBM onto the FLC). Final levels of abstraction correspond to implementation ones. Using the FBM Framework (its levels of abstraction, refinement stages and complexity considerations), it is possible developing large-scale BBSs.

The thesis has also described a case study to validate our initial steps towards the development of complex BB Systems using the FBM Framework. More particularly, it has been described how to develop a complex, non-hierarchical, multi-agent behaviour system using refinement capabilities of the FBM Framework.

Finally, the thesis has introduced some general ideas about the use of this framework to cope with some, current complexity issues around the behaviour-oriented nature of intelligence.

Chapter 6

Future Research Work

In a near future we plan to continue our research work around the definition and use of FBMs having in mind that it is worth working on the following tasks.

A) Using Simple FBMs to Design and Control Behaviour-Producing Modules

Our main intention is to evaluate large-scale FBMs designing numerous behaviours and the interaction among these. Then we shall follow implementations and pilot tests in a similar way to that exposed in chapter 3. This is, selecting an implementation technique, developing the resulting architecture and testing this with the appropriate agent / robot.

B) Implementing Simple FBMs using other possible Fuzzy Systems

The results of our pilot tests have revealed a strong potential for the application of Fuzzy Logic to control movements of mobile robots. More particularly, this work has shown the flexibility provided by the fuzzy inference (and weighting) methods in different simulations of obstacle avoidance as well as the possible manipulation of levels of perception under the Fuzzy approach. We aim to carry on our studies in these Fuzzy Logic applications.

Fuzzy weighting factors and fuzzy methods will be an important support for this research. Regarding the levels of perception handled by the FLC, we believe that this is still a research issue that is open for further developments.

Considering these ideas, we intend to focus our practical work on the relevance of other Fuzzy Systems such as Fuzzy Cognitive Maps, Hybrid NN models or Recurrent Fuzzy Systems to possibly implement FBMs. In doing so, we would like to contribute to the development of new BB architectures.

C) A Proof-of-Concept for More Complex FBMs

The thesis has introduced the concept of "more complex FBMs". In a near feature, we shall provide more examples of these structures as well as some possible implementations and evaluations.

D) Prototyping more BB Systems using either Simple or More complex FBMs

The thesis has presented an implementation of a simple FBM using FLC implementation techniques. One of our motivations for future explorations is that we can prototype other possible FBM-based architectures. This might also follow the general ideas the thesis has introduced regarding the FBM framework of development. For example, it would be possible trying to derive some of the BBSs reviewed in chapter 1 (section 1.4).

E) Defining, Using and Evaluating Learning FBMs

Some interesting (cognitive) processes introduced in this chapter 1 (e.g. learning, assimilation and adaptation) will also be considered in our future work due to their particular relevance in the BB viewpoint we have investigated. Indeed, we also intend to explore the definition and use of Learning FBMs from a more detailed study of the learning approaches within BB AI and AAs.

F) The Development of Large-Scale BBSs Using the FBMs Framework

Finally, we shall extend the work around the FBM Framework. In Chapter 4, we have introduced our preliminary ideas about how to develop large-scale BBSs using FBMs. In a near future, we shall complete this work while deriving more complex BBSs.

Appendix A

Fuzzy Sets and Fuzzy Relations

Classical Set Theory ("black or white") assumes that an element either does or does not belong to a set. *Fuzzy Set Theory* ("grey is better") extends this idea by considering that an element can belong to a *fuzzy set* to a certain degree (or membership value). A *classical set* is characterised with membership functions that map elements from a given universe of discourse to a two-valued set (bi-valued logic with crisp distinction between 0 and 1; between non-membership and whole membership). By contrast, the membership functions of the fuzzy sets map the same universe of discourse to a multi-valued set (multi-valued logic between 0 and 1; fuzzy granularity with infinite degrees of membership [Zadeh-65]).

Zadeh states that *Fuzzy Logic* is used to mean Fuzzy Sets Theories [Zadeh-72b; Zadeh-72d; Zadeh-92]. He introduced numerous fuzzy logic methods from the need of non-classical logical connectives to handle fuzzy sets [Zadeh-73b; Zadeh-75]. Each fuzzy logic connective (e.g. Zadeh's fuzzy implication) is associated with a mapping by which the truth value of a combined proposition can be determined by the truth values of atomic propositions. These mappings have been termed *fuzzy relations*.

Some definitions of fuzzy sets and fuzzy relations are listed below (see also [Zadeh-65; Zadeh & Kacprzyk-92; Dubois & Prade-83; Wang & Loe-93]).

Universe of Discourse

A universe of discourse is a discrete or continuous collection of objects denoted as:

$$U = \{u\} \qquad \text{(Def. A.1)}$$

Fuzzy Set

A fuzzy set is a class of objects with a continuum of grades of memberships [Zadeh-65; Wang & Loe-93]. Such a set on a universe of discourse U is defined as a collection of ordered pairs of the form

$$A = \{(u, \mu_A(u)) \text{ such that } u \in U\} \qquad \text{(Def. A.2)}$$

where u represents a generic (crisp) element of U and $\mu_A(u)$ is the degree of membership to which u belongs to A i.e. a characteristic function:

$$\mu_A : U \longrightarrow [0,1] \qquad \text{(Def. A.3)}$$

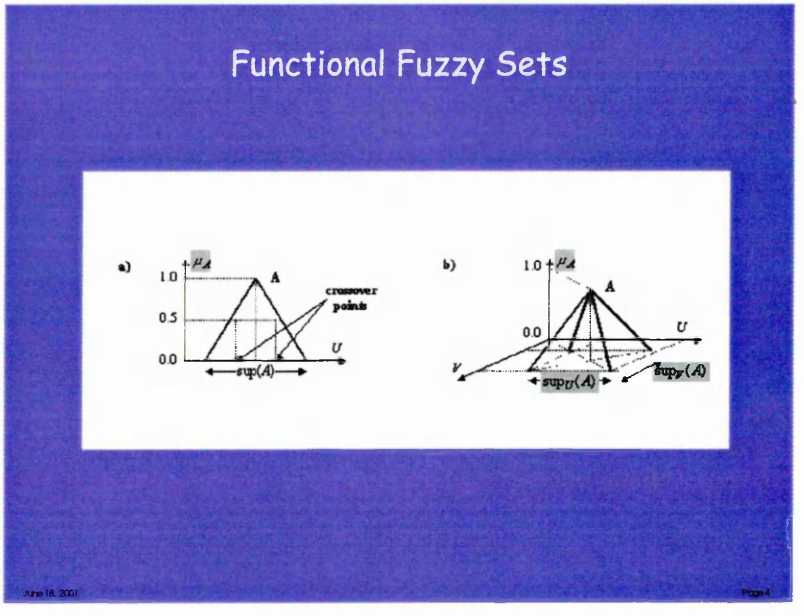


Fig. A.1: Diagrammatic Representation of a (Triangular) Functional (a) 1D fuzzy set A, (b) 2D fuzzy set A.

Support of a Fuzzy Set

The support of the fuzzy set A given in Def. A.2 is the crisp interval (see Fig. A.1(a)) containing elements of U which have non-zero membership degrees. This is denoted by:

$$\sup(A) = \{u \in U; \mu_A(u) > 0\} \qquad \text{(Def. A.4)}$$

Fuzzy Singleton

Fuzzy sets of cardinality equal to 1 are referred to as fuzzy singletons.

Crossover Point

A crossover point (see Fig. A.1(a)) of a fuzzy set A is an element $u \in U$ such that $\mu_A(u) = 0.5$.

In general, fuzzy sets can be either numerically or functionally represented. The following definition expresses the differences among these representations.

Numerical and Functional Fuzzy Set

A numerical fuzzy set (fuzzy number) corresponds to the interval of its support. A functional fuzzy set can be differently shaped (e.g. in triangular, bell or trapezoidal form). Fig. A.1(a) shows a simple example of a one-Dimensional (1D) triangular (functional) fuzzy set.

Dimension of a Fuzzy Set

The dimension of a fuzzy set A is the dimension of the universe of discourse U on which it is supported. Hence, a two-dimensional (2D) fuzzy set A (see Fig. A.1(b)) defined on a generic 2D (product space) universe of discourse U×V can be defined as

$$A = \{(u, v, \mu_A(u,v)\} \text{ such that } (u,v) \in U \times V \qquad \text{(Def. A.5)}$$

Fuzzy Set Theoretic Operations

The fuzzy set-theoretic operations of union ($\cup$), intersection ($\cap$) and complement ($\approx$) are extensions (in terms of membership functions) of the same classical binary operations.

Considering two fuzzy sets A and B defined over the same universe of discourse U, these extensions are given by the following formulas

$$\forall u \in U; \mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u)) \qquad \text{(Def. A.6.1)}$$

$$\forall u \in U; \mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u)) \qquad \text{(Def. A.6.2)}$$

$$\forall u \in U; \mu_{A\approx}(u) = 1 - \mu_A(u) \qquad \text{(Def. A.6.3)}$$

where $\mu_{A \cup B}$, $\mu_{A \cap B}$ and $\mu_{A\approx}$ are the membership functions (see Def. A.2) of the fuzzy sets corresponding to $A \cup B, A \cap B$ and $A \approx$, respectively.

Cartesian Product of Fuzzy Sets

The Cartesian product of the fuzzy sets $A_1, A_2, \ldots, A_n$ on $U_1, U_2, \ldots, U_n$, respectively,

is a fuzzy set defined on the n-Dimensional (nD) product space of universes $U_1 \times U_2 \times \ldots \times U_n$ with a membership function

$$\mu_{A_1 \times A_2 \times \ldots \times A_n}(u_1, u_2, \ldots, u_n) = \{\mu_{A_1}(u_1) * \mu_{A_2}(u_2) * \ldots * \mu_{A_n}(u_n)\} \qquad \text{(Def. A.7)}$$

where * could be any triangular operator (see Table A.1) such as the intersection (triangular norm) or union (triangular co-norm).

Linguistic Variable

A linguistic variable is a fuzzy concept [Zadeh-73b] characterised by: its name "x", a universe of discourse U and a set of terms, or linguistic values, T("x"). For instance, considering a space of distances U = [0,1023], the following set

$$T(\text{"distance"}) = \{\text{"big"}, \text{"medium"}, \text{"small"}\}.$$

defines three possible terms (linguistic values) of the linguistic variable "distance" which, in practice, correspond to three fuzzy sets (see Def. A.1 & Def. A.2) on the 1D support U.

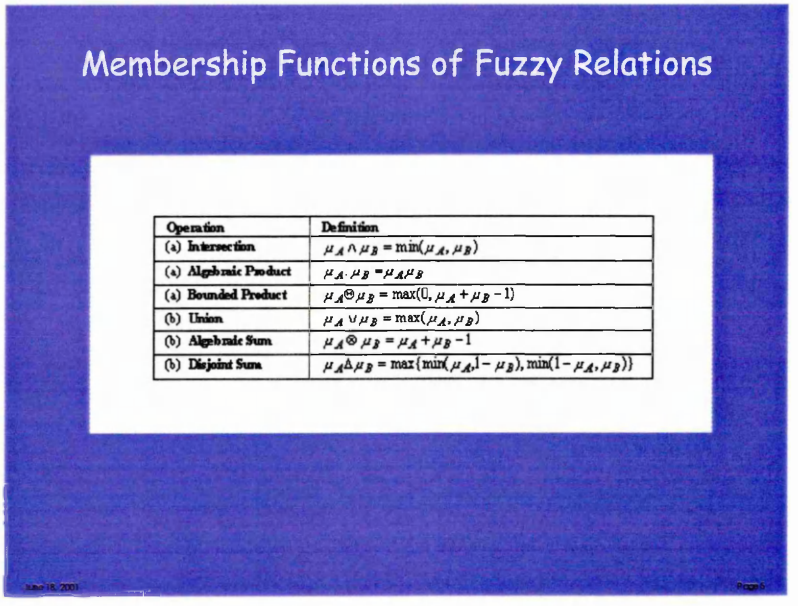


Table A.1: Membership Functions of (a) Triangular Norms, (b) Triangular Co-Norms.

Fuzzy Relations: Conjunction, Disjunction & Implication

A fuzzy relation (in the context of the fuzzy sets) is a natural extension of a classical relation (in the context of the crisp sets). By definition, an n-ary fuzzy relation R between n fuzzy sets (see Def. A.7) is another fuzzy set given by

$$R_{U_1 \times U_2 \times \ldots \times U_n} = \{(u_1, u_2, \ldots, u_n), \mu_R(u_1, u_2, \ldots, u_n)\} \qquad \text{(Def. A.8)}$$

where $(u_1, u_2, \ldots, u_n)$ corresponds to any point in the n-dimensional product space $U_1 \times U_2 \times \ldots \times U_n$.

Different fuzzy relations are obtained [Lee I-90; Wang & Loe-93] depending on which triangular (×) operator is applied between their membership functions. The three categories that have received most attention in Fuzzy Theory are:

- fuzzy conjunctions or fuzzy relations defined with triangular norms (see Table A.1(a)),
- fuzzy disjunctions or fuzzy relations defined with triangular co-norms (see Table A.1(b)),
- fuzzy implications which are defined on both types of triangular operators (see Table A.2).

Let us consider the fuzzy sets A and B defined on the universes of discourse U and V, respectively. Then a fuzzy conjunction between A and B is a fuzzy relation ($A \times B$) or 2D fuzzy set on the product space $U \times V$ with a membership function related to those of A and B by the formula

$$\mu_{A \times B}(u, v) = \mu_A(u) * \mu_B(v) \qquad \text{(Def. A.9.1)}$$

and, a fuzzy disjunction between the same fuzzy sets A and B is another fuzzy relation ($A \times B$) or 2D fuzzy set on $U \times V$ with a membership function defined as

$$\mu_{A\times B}(u,v) = \mu_A(u) \mp \mu_B(v) \qquad \text{(Def. A.9.2)}$$

The relevance of these categories (of a fuzzy relation) is in their application in the implementation of FLC architectures. Indeed the sort of fuzzy control rules (see Def. A.10.1 & Def. A.10.2) used in any FLC consist of fuzzy implications which link antecedents and consequents, both being defined as either the fuzzy disjunction or the fuzzy conjunction of different fuzzy sets. Furthermore, nearly 40 distinct fuzzy implications [Lee I-90] have been described in the literature of Fuzzy Logic Control for the manipulation of the fuzzy control rules that characterise the inference mechanisms of these architectures (see appendix B).

Table A.2 shows the membership functions of the fuzzy implications most commonly used in fuzzy inference mechanism. These are: Mamdani's mini-operation rule, Larsen's product operation rule, Zadeh's (I) max-min rule and Zadeh's (II) arithmetic rule.

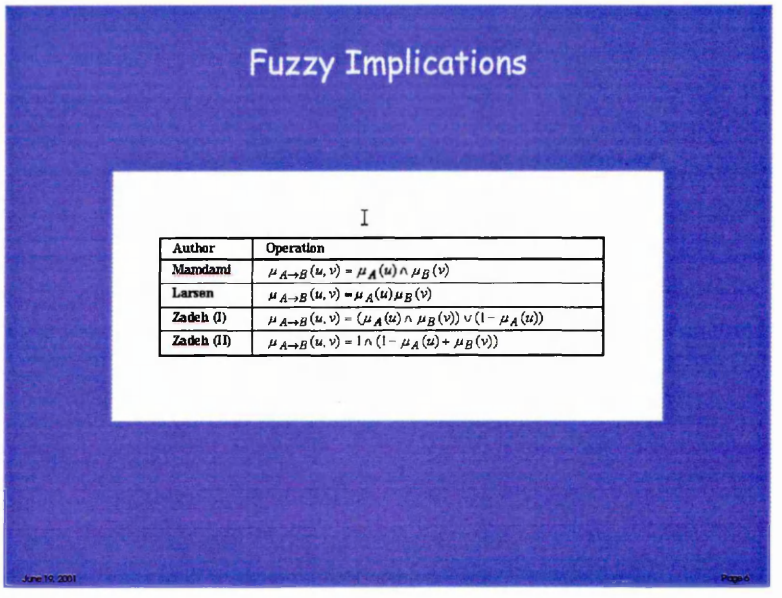


Table A.2: Fuzzy Implications.

The operations carried out with the fuzzy implication techniques of Mamdani and Larsen are explained in more detail within chapter 3. More information about the fuzzy implication operators can be seen in the corresponding implementation parameter included in appendix B.

Fuzzy Control Rule

A fuzzy control rule (or fuzzy inference rule) is a conditional proposition which can be represented as

IF "x is *A*" THEN "y is *B*" (Def. A.10.1)

or also

$$A \rightarrow B \qquad \text{(Def. A.10.2)}$$

where A and B are fuzzy sets (terms) of the linguistic variables "x" and "y", respectively, (i.e. $A \in$ T("x") and $B \in$ T("y")) and, the sentence connective THEN ($\rightarrow$) represents a generic fuzzy implication operator.

Sup-Star Composition and Sup-Star Compositional Rule of Inference

The sup-star composition $R \circ S$ between two fuzzy relations R, on $U \times V$ and S, on $V \times W$, is another fuzzy relation given by the formula

$$R \circ S_{U \times W} = \{((u,w), \sup_V (\mu_R(u,v) * \mu_S(v,w)))\} \qquad \text{(Def. A.11.1)}$$

where * represents any of the triangular norms (Table A.1(a)). As an extension of this definition, the sup-star compositional rule of inference, expressed as,

$$Y = X \circ R \qquad \text{(Def. A.11.2)}$$

corresponds to the inference of the fuzzy set $Y \in$ T("y") by the $\circ$ sup-star composition applied to the fuzzy set $X \in$ T("x") and the fuzzy relation (or set of fuzzy relations) R.

For instance, considering a fuzzy set "small" $\in$ T("distance"), a fuzzy set "fast" $\in$ T("speed") and a fuzzy relation R (which could be the whole fuzzy control rules set of a FLC), the following formula

$$\text{"fast"} = \text{"small"} \circ R$$

produces the inference of the fuzzy set "fast" by the sup-star composition of "small" and the fuzzy relation R.

Appendix B

Fuzzy Logic Control

Since Zadeh established the fundamentals of both Fuzzy Sets and Fuzzy Systems, the area of Fuzzy Logic Control has emerged as one of the most productive applications of his pioneering work. The spectrum of applications includes cement-kiln process control, robot control, image processing, motor control, automatic train operation, servo loop control, aircraft control and many others.

FLC Design

In designing a Fuzzy Logic Controller (FLC), the principal factors include:

- the inputs and outputs and, their universes of discourse,
- the scale factors of all the input-output variables,
- the membership functions of all the fuzzy sets and,
- the fuzzy control rule base.

The construction of the fuzzy sets (their membership functions) and the fuzzy control rules represent the key issues in designing a FLC. Depending on these, the resulting architecture may also have self-organising and learning capabilities.

Basic Structure of FLCs

The purpose of a FLC is to compute values of control action (output) variables from the observation of input variables of the process(es) under control. Here the relation between the input and output variables is viewed as a set of logical rules that the FLC architecture evaluates for every single input set.

The main functional components of a FLC are: a fuzzification unit, a decision making logic unit, a data base, a rule base and, a defuzzification unit.

Depending on the design objectives and the type of process(es) that need being controlled, different types of FLCs can be constructed. For instance, the FLC can have a fixed number of fuzzy control rules or, it may have learning capabilities through the modification of its data base.

The data base of the FLC have to be designed so that it contains all the membership functions defining the fuzzy sets of the input/output variables. The rule base maps all the fuzzy values of the inputs to the fuzzy values of the outputs. The input values are measured from the controlled process and, the output values are used by the FLC to control the process(es). The definition of all the fuzzy sets is one of the most important steps in the design of a FLC. Finally, the fuzzification and defuzzification operations are needed to map the crisp values of the input/output variables to and from the fuzzy sets that uses the FLC.

For some crisp variable, a fuzzification strategy involves:

- acquiring the crisp values of the input values,
- mapping the crisp values of the input values into the corresponding universe of discourse and,
- converting the mapped data into fuzzy sets.

For the output values, the defuzzification usually occurs as part of the fuzzy differencing. This last mechanism involves weighting and combining a number of fuzzy sets resulting from the fuzzy inference process that characterizes the decision making logic unit.

The rule base of the FLC defines the fuzzy control policy i.e. the control relationships among input and output fuzzy sets. These rules are usually expressed in the form "if-then" format.

The data base may be static or, dynamic. A dynamic data base is required to permit a learning FLC. The fuzzy decision making logic unit uses fuzzy logic operators (selected by the designer) to perform the fuzzy inference mechanism arriving at the fuzzy control actions that have to be executed. This mechanism also involves evaluating the data base for the fuzzified inputs that reach the FLC.

During a fuzzy inference, the following operations have to be performed.

- Determine the degree of match between the fuzzy input data and the defined fuzzy sets for each input variable.
- Carry out the fire strength calculations for each rule based on the degree of match and the fuzzy connectives (e.g. "and", "or") used in the antecedent part of the rule.
- Derive the control (output) based on the calculated fire strength and the defined fuzzy sets for each output variable in the consequent part of the rule.

The final crisp control action is inferred either by selecting or, by combining the calculated control outputs and depends upon the defuzzification strategy that has been selected by the designer of the FLC.

In controlling a process, all the fuzzy control rules are compared to the current inputs (observations) and fired. The action of each rule is weighted by how close the antecedent part of the rule matches the current observation.

Hence, a good way of tuning a FLC is to modify the set of fuzzy control rules i.e. adding or, deleting rules until appropriate outputs (control actions) are observed.

Design of Fuzzy Control Rules

From the design perspective, fuzzy control rules have been constructed based on the following:

Heuristic Approach

This is largely based on qualitative information about the behaviour of the systems (processes) under control. In control engineering, the control rules are often organised by using a closed-loop system trajectory in a phase plane. Then, once the designer observes the effects of parameters adjustment based on phase plane analysis and of system's behaviour, the approach can be applied.

Deterministic Approach

This determines the linguistic structure or parameters of the fuzzy control rules that satisfy the control objectives and constraints.

Fuzzy Modelling Technique

This is used to build qualitative models of the systems under control and without any prior knowledge about it. Using this technique, the controlled process can be directly modelled provided that sufficient input and output data is available for the process (system) under control.

The FLC can be designed once the fuzzy model of the controlled process has been established.

Adaptive FLCs

Adaptive control is identified with the process of accommodating the changes of a process in a dynamic environment.

If a designed FLC does not perform satisfactorily after a time due to some changes in the process dynamics, it is required adapting all its design parameters so that it performs the required control actions.

Initial design parameters such as fuzzy sets, membership functions, universes of discourse and control rules may need to be modified and adjusted to meet the design objectives.

An adaptive FLC is often constructed with self-tuning and learning capabilities so that it is able to:

- generate new fuzzy rules,
- modify the existing fuzzy rules,
- modify the designed fuzzy sets,
- adjust all the membership functions of its data base and,
- adjust the universes of discourse.

This adaptive FLC may include the following elements: a performance measurement module, a supervising and tuning module and, the FLC that has to be adapted.

Appendix C

The FBM-FLC Program

FUZZYSETS.H

```
#ifndef FUZZYSETS_H
#define FUZZYSETS_H
#define MAX_CARD      250
#define MAX_TEXT      50


struct Value1
{
  short int          CrispX;
  float              MembershipValue;
  struct Value1     *Next;
};


struct Value2
{
  short int          CrispX,CrispY;
  float              MembershipValue;
  struct Value2     *Next;
};


struct FuzzyProperties
{
  char               Name[25];
  char               MembershipFunctionType[25];
  short int          Cardinality;
};


struct FuzzySet1              /* 1-Dimensional fuzzy sets */
{
  struct FuzzyProperties   Properties;
  short int                Support[2];
  float                    Center;
  struct Value1            *Values;
  struct FuzzySet1         *Next;
};


struct FuzzySet2              /* 2-Dimensional fuzzy sets */
```

```
{
   struct FuzzyProperties  Properties;
   short int                SupportX[2],SupportY[2];
   float                    CenterX,CenterY;
   struct Value2           *Values;
   struct FuzzySet2        *Next;
};

#endif
```

FUZZYSETS.C

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>
#include "../SRC/include.h"
#include "../SRC/types.h"
#include "fuzzysets.h"
#define FreeValue(x)    free(x);


void Singleton1(short int X1, float X2, short int X3, short int X, float *mf)
{
  if (X = X2)
    *mf = 1.0;
  else
    *mf = 0.0;
}


void Singleton2(short int X1, float X2, short int X3, short int Y1, float Y2, short int Y3, short int X, short int Y,
                float *mf)
{
  if ((X = X2) && (Y = Y2))
    *mf = 1.0;
  else
    *mf = 0.0;
}


void Triangular1(short int X1, float X2, short int X3, short int X, float *mf)
{
   float m1 = 1/(X2-X1),
         b1 = X1/(X1-X2),
         m2 = 1/(X2-X3),
         b2 = X3/(X3-X2);

 if ((X >= X1) && (X <  X2)) *mf =  (m1*X)+b1;
 else if ((X > X2) && (X <= X3)) *mf =  (m2*X)+b2;
 else /*(X = X2)*/  *mf = 1.0;
}
```

```
/*************************************************************/
/*                    1-DIM FUZZY SETS                       */
/*************************************************************/


void AddValues1(struct Value1 *value1, struct Value1 *value2)
{
  value2->Next = value1->Next;
  value1->Next = value2;
}


struct Value1 *CreateValue1(short int X, float m)
{
  struct Value1 *value;

  value               = (struct Value1 *)malloc(sizeof(struct Value1));
  value->CrispX       = X;
  value->MembershipValue  = m;
  value->Next         = NULL;
  return(value);
}


void FreeValues1(struct Value1 *value)
{
  if (value)
  {
    FreeValues1(value->Next);
    FreeValue(value);
  }
}


struct FuzzySet1 *CreateEmptyFuzzySet1()
{
  struct FuzzySet1 *fuzzy_set;

  fuzzy_set = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  strcpy(fuzzy_set->Properties.Name, "new");
  strcpy(fuzzy_set->Properties.MembershipFunctionType, "Singleton1");
  fuzzy_set->Properties.Cardinality = 0;
  fuzzy_set->Values = (struct Value1 *)NULL;
  fuzzy_set->Support[0] = 0;
  fuzzy_set->Support[1] = 0;
  fuzzy_set->Center    = 0.0;
  fuzzy_set->Next = NULL;
  return(fuzzy_set);
}


void FreeFuzzySet1(struct FuzzySet1 *fuzzy_set)
{
  FreeValues1(fuzzy_set->Values);
  free(fuzzy_set);
}
```

```
short int Cardinality1(short int X1, short int X2)
{
 short int card1 = 0,
 num = X1;
 while (num <= X2)
 {
   card1++;
   num++;
 }
 return(card1);
}


struct FuzzySet1 *CreateFuzzySet1(char name[MAX_TEXT],
                                  char membership_function[MAX_TEXT],
                                 short int universe_of_discourse[2],
                                  short int support[2],
                                  float center)
{
 struct FuzzySet1 *fuzzy_set;
 struct Value1 *value1, *value2;

 short int U,
           U1 = universe_of_discourse[0],
           Un = universe_of_discourse[1];
 float mf_valor = 0.0;

 fuzzy_set = CreateEmptyFuzzySet1();
 strcpy(fuzzy_set->Properties.Name, name);
 strcpy(fuzzy_set->Properties.MembershipFunctionType, membership_function);
 fuzzy_set->Properties.Cardinality = Cardinality1(support[0], support[1]);
 fuzzy_set->Support[0]          = support[0];
 fuzzy_set->Support[1]          = support[1];
 fuzzy_set->Center              = center;
 value1 = CreateValue1(U1,0.0);
 fuzzy_set->Values = value1;
 U = U1+1;
 while (U < support[0])
 {
   value2 = CreateValue1(U,0.0);
   AddValues1(value1,value2);
   value1 = value1->Next;
   U++;
 }
 while (U <= support[1])
 {
   if (strcmp(membership_function,"Triangular1")==0)
           Triangular1(support[0],center,support[1],U,&mf_valor);
   else if (strcmp(membership_function,"Bell1")==0)
           Bell1(support[0],center,support[1],U,&mf_valor);
   else /* (strcmp(membership_function,"Singleton1")==0) */
           Singleton1(support[0],center,support[1],U,&mf_valor);
   value2 = CreateValue1(U,mf_valor);
   AddValues1(value1,value2);
   value1 = value1->Next;
   U++;
 }
```

```
  while (U <= Un)
  {
    value2 = CreateValue1(U,0.0);
    AddValues1(value1,value2);
    value1 = value1->Next;
    U++;
  }
  return(fuzzy_set);
}


struct Value1 *FindValue1(struct FuzzySet1 *fuzzy_set, short int X)
{
  struct Value1 *search, *found;

  search = fuzzy_set->Values;
  found = NULL;
  while(search)
  {
    if((search->CrispX-X)==0)
    {
      found = search;
      search = NULL;
    }
    else
      search = search->Next;
  }
  return(found);
}


short int DecFindCrispValue1(struct FuzzySet1 *fuzzy_set, float mf,
                             short int init, short int end)
{
   struct Value1 *search;
   short int found;
   search = FindValue1(fuzzy_set, init);
   found = init;
   while(search)
   {
     if((search->MembershipValue-mf)<=0)
     {
       found = search->CrispX;
       search = NULL;
     }
     else
       search=search->Next;
   }
   return(found);
}


short int IncFindCrispValue1(struct FuzzySet1 *fuzzy_set, float mf, short int init, short int end)
{
  struct Value1 *search;
  short int found;
  search = FindValue1(fuzzy_set,init);
```

```
 found = init;
 while(search)
 {
   if((search->MembershipValue-mf)>=0)
   {
    found = search->CrispX;
    search = NULL;
   }
   else
    search=search->Next;
 }
 return(found);
}


float Sum(struct FuzzySet1 *fuzzy_set)
{
 float sum;
 struct Value1 *value;

 value = FindValue1(fuzzy_set,fuzzy_set->Support[0]);
 sum = (value->MembershipValue)*(value->CrispX);
 value = value->Next;
 while(value)
 {
  sum = sum+((value->MembershipValue)*(value->CrispX));
  if(value->CrispX==fuzzy_set->Support[1])
   value = NULL;
  else
   value=value->Next;
 }
 return(sum);
}


void PrintValue1(struct Value1 *val)
{
 char        buffer1[15],buffer2[15];

 sprintf(buffer1,"X%7d",val->CrispX);
 sprintf(buffer2,"mf%8d",val->MembershipValue);
 printf("%s",buffer1);printf("%s",buffer2);
}


void PrintFuzzySet1(struct FuzzySet1 *fuzzy_set)
{
 struct Value1 *value;

 printf("Primary Fuzzy Sets.\n\n");
 printf("  -Name:%s",fuzzy_set->Properties.Name);
 printf("  -Membership Function Type:%s",
          fuzzy_set->Properties.MembershipFunctionType);
 printf("  -Cardinality:%d\n",fuzzy_set->Properties.Cardinality);
 printf("  -XSupport:[%d,%d,%d]\n",fuzzy_set->Support[0], fuzzy_set->Center, fuzzy_set->Support[1]);
 value = fuzzy_set->Values;
 while(value)
 {
```

```
   printf("\t");
   PrintValue1(value);
   value = value->Next;
   if(value==fuzzy_set->Values)
    value=NULL;
 }
}

void WriteFuzzySet1ToFile(struct FuzzySet1 *fuzzy_set,FILE *file)
{
  struct Value1    *value;

  fprintf(file,"# File  : PRIMARY FUZZY SETS.\n");
  fprintf(file,"# Author: Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date  :      \n");
  fprintf(file,"-----------------------------------------------------\n\n\n");
  fprintf(file,"  -Name: %s\n",fuzzy_set->Properties.Name);
  fprintf(file,"  -Membership Function Type: %s\n", fuzzy_set->Properties.MembershipFunctionType);
  fprintf(file,"  -Cardinality: %d\n", fuzzy_set->Properties.Cardinality);
  fprintf(file,"  -XSupport: [ %d,%d]\n", fuzzy_set->Support[0],fuzzy_set->Support[1]);
  fprintf(file,"  -XCenter: %f\n",fuzzy_set->Center);
  fprintf(file,"  -Values:\n\n");
  value = FindValue1(fuzzy_set,fuzzy_set->Support[0]);
  while((value) && (value->CrispX<=fuzzy_set->Support[1]))
  {
    fprintf(file,"\tX = %d   \t",value->CrispX);
    fprintf(file,"mf = %f\n",value->MembershipValue);
    value = value->Next;
  }
}



void ReadFuzzySet1FromFile(struct FuzzySet1 *fuzzy_set,FILE *file)
{
}


/*************************************************************************/
/*                    2-DIM FUZZY SETS                                   */
/*************************************************************************/


void AddValues2(struct Value2 *value1, struct Value2 *value2)
{
 value2->Next = value1->Next;
 value1->Next = value2;
}


struct Value2 *CreateValue2(short int X, short int Y, float m)
{
 struct Value2 *value;

 value              = (struct Value2 *)malloc(sizeof(struct Value2));
 value->CrispX              = X;
 value->CrispY              = Y;
```

```
 value->MembershipValue = m;
 value->Next            = NULL;
 return(value);
}


void FreeValues2(struct Value2 *value)
{
 if (value)
 {
  FreeValues2(value->Next);
  FreeValue(value);
 }
}


struct FuzzySet2 *CreateEmptyFuzzySet2()
{
 struct FuzzySet2 *fuzzy_set;
 short int i;

 fuzzy_set = (struct FuzzySet2 *)malloc(sizeof(struct FuzzySet2));
 strcpy(fuzzy_set->Properties.Name, "new");
 strcpy(fuzzy_set->Properties.MembershipFunctionType, "Singleton2");
 fuzzy_set->Properties.Cardinality    = 0;
 fuzzy_set->Values                    = (struct Value2 *) NULL;
 for(i = 0;i < 2;++i)
 {
  fuzzy_set->SupportX[i]              = 0;
  fuzzy_set->SupportY[i]              = 0;
 }
 fuzzy_set->CenterX                   = 0;
 fuzzy_set->CenterY                   = 0;
 fuzzy_set->Next                      = NULL;
 return(fuzzy_set);
}


void FreeFuzzySet2(struct FuzzySet2 *fuzzy_set)
{
 FreeValues2(fuzzy_set->Values);
 free(fuzzy_set);
}



struct FuzzySet2 *CreateFuzzySet2(char name[MAX_TEXT], char membership_function[MAX_TEXT],
                                  short int universe_of_discourseX[2], short int universe_of_discourseY[2],
                                   short int supportX[2], short int supportY[2],
                                   float centerX, float centerY)
{
 struct FuzzySet2 *fuzzy_set;
 struct Value2 *value1, *value2;

 short int i, Ux, Uy,
           Ux1 = universe_of_discourseX[0],
           Uxn = universe_of_discourseX[1],
```

```
            Uy1 = universe_of_discourseY[0],
            Uyn = universe_of_discourseY[1];
 float mf_valor = 0.0;

 fuzzy_set = CreateEmptyFuzzySet2();
 strcpy(fuzzy_set->Properties.Name, name);
 strcpy(fuzzy_set->Properties.MembershipFunctionType, membership_function);
 fuzzy_set->Properties.Cardinality  = Cardinality1(supportX[0], supportX[1])*
                                      Cardinality1(supportY[0], supportY[1]);
 for(i = 0; i< 2; ++i)
 {
  fuzzy_set->SupportX[i]        = supportX[i];
  fuzzy_set->SupportY[i]        = supportY[i];
 }
 fuzzy_set->CenterX             = centerX;
 fuzzy_set->CenterY             = centerY;
 value1 = CreateValue2(Ux1,Uy1,0.0);
 fuzzy_set->Values = value1;
 Ux = Ux1;
 Uy = Uy1;
 while (Ux < supportX[0])
 {
  while(Uy < supportY[0])
  {
   value2 = CreateValue2(Ux,Uy,0.0);
   AddValues2(value1, value2);
   value1 = value1->Next;
   Uy++;
  }
  Ux++;
 }
 while (Ux <= supportX[1])
 {
  while(Uy <= supportY[1])
  {
   if (strcmp(membership_function,"Triangular2")==0)
          Triangular2(supportX[0],centerX,supportX[1],
                      supportY[0],centerY,supportY[1],
                      Ux,Uy,&mf_valor);
   else if (strcmp(membership_function,"Bell2")==0)
          Bell2(supportX[0],centerX,supportX[1],
                supportY[0],centerY,supportY[1],
                Ux,Uy,&mf_valor);
   else /* (strcmp(membership_function,"Singleton2")==0) */
          Singleton2(supportX[0],centerX,supportX[1],
                     supportY[0],centerY,supportY[1],
                     Ux,Uy,&mf_valor);
   value2 = CreateValue2(Ux,Uy,mf_valor);
   AddValues2(value1,value2);
   value1 = value1->Next;
   Uy++;
  }
  Ux++;
 }
 while (Ux <= Uxn)
 {
  while(Uy <= Uyn)
  {
```

```
      value2 = CreateValue2(Ux,Uy,0.0);
      AddValues2(value1, value2);
      value1 = value1->Next;
      Uy++;
    }
    Ux++;
  }
  return(fuzzy_set);
}

struct Value2 *FindValue2(struct FuzzySet2 *fuzzy_set, short int X, short int Y)
{
  struct Value2 *search, *found;

  search = fuzzy_set->Values;
  found = NULL;
  while(search)
  {
    if(((search->CrispX-X)==0) && ((search->CrispY-Y)==0))
    {
      found = search;
      search = NULL;
    }
    else
      search = search->Next;
  }
  return(found);
}


void PrintValue2(struct Value2 *val)
{
  char        buffer1[15],buffer2[15],buffer3[15];

  sprintf(buffer1,"X%d",val->CrispX);
  sprintf(buffer2,"Y%d",val->CrispY);
  sprintf(buffer3,"mf%d",val->MembershipValue);
  printf("%s",buffer1);
  printf("%s",buffer2);
  printf("%s",buffer3);
}

void PrintFuzzySet2(struct FuzzySet2 *fuzzy_set)
{
 struct Value2 *value;

 printf("Primary Fuzzy Sets.\n\n");
 printf("  -Name:%s",fuzzy_set->Properties.Name);
 printf("  -Membership Function Type:%s\n",
          fuzzy_set->Properties.MembershipFunctionType);
 printf("  -Cardinality:%d\n",
          fuzzy_set->Properties.Cardinality);
 printf("  -XSupport:[%d,%d,%d]\n",
                    fuzzy_set->SupportX[0],
                    fuzzy_set->CenterX,
                fuzzy_set->SupportX[1]);
 printf("  -YSupport:[%d,%d,%d]\n",
```

```
                        fuzzy_set->SupportY[0],
                        fuzzy_set->CenterY,
                        fuzzy_set->SupportY[1]);
 printf("  -Values:\n\n");
 value = fuzzy_set->Values;
 while(value)
 {
  printf("\t");
  PrintValue2(value);
  printf("\n");
  value = value->Next;
 }
}


void WriteFuzzySet2ToFile(struct FuzzySet2 *fuzzy_set,FILE *file)
{
  struct Value2   *value;

  fprintf(file,"#FILE: PRIMARY 2-DIM. FUZZY SETS.\n");
  fprintf(file,"# Author: Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date  :     \n");
  fprintf(file,"-----------------------------------------------------\n\n\n");
  fprintf(file,"  -Name: %s\n",fuzzy_set->Properties.Name);
  fprintf(file,"  -Membership Function Type: %s\n",
                        fuzzy_set->Properties.MembershipFunctionType);
  fprintf(file,"  -Cardinality: %d\n",
                        fuzzy_set->Properties.Cardinality);
  fprintf(file,"  -XSupport: [%d,%d]\n",
                        fuzzy_set->SupportX[0],fuzzy_set->SupportX[1]);
  fprintf(file,"  -YSupport: [%d,%d]\n",
              fuzzy_set->SupportY[0],fuzzy_set->SupportY[1]);
  fprintf(file,"  -XCenter: %f\n",fuzzy_set->CenterX);
  fprintf(file,"  -YCenter: %f\n",fuzzy_set->CenterY);
  fprintf(file,"  -Values:\n\n");
  value = FindValue2(fuzzy_set,
                        fuzzy_set->SupportX[0],
                        fuzzy_set->SupportY[0]);
  while((value) && (value->CrispX<=fuzzy_set->SupportX[1]) &&
                  (value->CrispY<=fuzzy_set->SupportY[1]))
  {
    fprintf(file,"\tX = %d   \t",value->CrispX);
    fprintf(file,"\tY = %d   \t",value->CrispY);
    fprintf(file,"mf = %f\n",value->MembershipValue);
    value = value->Next;
  }
}


void ReadFuzzySet2FromFile(struct FuzzySet2 *fuzzy_set,FILE *file)
{
}



/*****************************************/
/*          FUZZY OPERATORS          */
/*****************************************/
```

```
struct FuzzySet1 *Union1(struct FuzzySet1 *A,
                         struct FuzzySet1 *B,
                         char name[MAX_TEXT],
                         short int univ[2])
{
 struct FuzzySet1 *C;
 struct Value1 *value1, *value2;

 short int U,
           X1 = A->Support[0],
           Xn = B->Support[1],
           U1 = univ[0],
           Un = univ[1];
 float mf_value = 0.0;

 C = CreateEmptyFuzzySet1();
 strcpy(C->Properties.Name, name);
 strcpy(C->Properties.MembershipFunctionType,
        A->Properties.MembershipFunctionType);
 C->Properties.Cardinality = Cardinality1(X1,Xn);
 C->Support[0] = X1;
 C->Support[1] = Xn;
 C->Center    = X1 + ((Xn - X1)/2);
 value1 = CreateValue1 (U1,0.0);
 C->Values = value1;
 while (U < X1)
 {
   value2 = CreateValue1(U,0.0);
   AddValues1(value1,value2);
   value1 = value1->Next;
   U++;
 }
 while (U <= Xn)
 {
   if (strcmp( A->Properties.MembershipFunctionType,"Triangular1")==0)
          Triangular1(X1,C->Center,Xn,U,&mf_value);
   else if (strcmp( A->Properties.MembershipFunctionType,"Bell1")==0)
          Bell1(X1,C->Center,Xn,U,&mf_value);
   else /* (strcmp( A->Properties.MembershipFunctionType,"Singleton1")==0) */
          Singleton1(X1,C->Center,Xn,U,&mf_value);
   value2 = CreateValue1(U,mf_value);
   AddValues1(value1,value2);
   value1 = value1->Next;
   U++;
 }
 while (U <= Un)
 {
   value2 = CreateValue1(U,0.0);
   AddValues1(value1,value2);
   value1 = value1->Next;
   U++;
 }
 return(C);
}
```

```
struct FuzzySet1 *Complement1(struct FuzzySet1 *A,
                              short int Univ[2])
{
 struct FuzzySet1 *complement;
 struct Value1 *value;
 short int X,
           X1 = A->Support[0],
           Xn = A->Support[1];
 short int supportC[2] = {X1,Xn};
 float mf_value;

 complement = CreateFuzzySet1(A->Properties.Name, A->Properties.MembershipFunctionType,
                              Univ,supportC, A->Center);
 value = FindValue1(complement,X1);
 X = X1;
 while (X <= Xn)
 {
  mf_value = value->MembershipValue;
  value->MembershipValue = 1.0 - mf_value;
  value = value->Next;
  X++;
 }
 return(complement);
}


struct FuzzySet2 *Complement2(struct FuzzySet2 *A, short int UnivX[2], short int UnivY[2])
{
 struct FuzzySet2 *complement;
 struct Value2 *value;
 short int X,Y,
           X1 = A->SupportX[0],
           Xn = A->SupportX[1],
           Y1 = A->SupportY[0],
           Yn = A->SupportY[1];

 short int XsupportC[2] = {X1,Xn},
           YsupportC[2] = {Y1,Yn};
 float mf_value;
 complement = CreateFuzzySet2(A->Properties.Name,
                              A->Properties.MembershipFunctionType,
                              UnivX,UnivY,
                              XsupportC, YsupportC,
                              A->CenterX,A->CenterY);
 value = FindValue2(complement,X1,Y1);
 X = X1;
 Y = Y1;
 while (X <= Xn)
 {
  while (Y <= Yn)
  {
   mf_value = value->MembershipValue;
   value->MembershipValue = 1.0 - mf_value;
   value = value->Next;
   Y++;
  }
  X++;
 }
```

```
  return(complement);
}


struct FuzzySet2 *CreateNullCartesianProduct11(short int univU[2],
                                               short int univV[2])
{
  struct FuzzySet2 *cart_product;
  struct Value2 *value1, *value2;
  short int i, u, v;


  cart_product = CreateEmptyFuzzySet2();
  for(i = 0; i< 2; ++i)
  {
    cart_product->SupportX[i]      = univU[i];
    cart_product->SupportY[i]      = univV[i];
  }
  value1 = CreateValue2(univU[0],univV[0],0.0);
  cart_product->Values = value1;
  for(u = univU[0];u<=univU[1];++u)
  for(v = univV[0];v<=univV[1];++v)
  {
     value2 = CreateValue2(u,v,0.0);
     AddValues2(value1, value2);
     value1 = value1->Next;
  }
  return(cart_product);
}




/*************************************************************************/
/*                          TRIANGULAR NORMS                             */
/*************************************************************************/


void TNormIntersection(float mf1, float mf2, float *mf)
{
  if ((mf1 - mf2)>= 0.0)
    *mf = mf2;
  else
    *mf = mf1;
}


void TNormProduct(float mf1, float mf2, float *mf)
{
  *mf = mf1*mf2;
}


void TNormBoundedProduct(float mf1, float mf2, float *mf)
{
```

```
  if ((mf1+mf2-1)<=0)
   *mf = 0.0;
  else
   *mf = mf1+mf2-1;
}

void TNormDrasticProduct(float mf1, float mf2, float *mf)
{
  if (mf2 == 1.0)
   *mf = mf1;
  else if (mf1 == 1.0)
   *mf = mf2;
  else
   *mf = 0.0;
}


/*****************************************************************************/
/*                          TRIANGULAR CO-NORMS                              */
/*****************************************************************************/


void TCoNormUnion(float mf1, float mf2, float *mf)
{
  if (mf1-mf2>=0)
    *mf = mf1;
  else
    *mf = mf2;
}

void TCoNormSum(float mf1, float mf2, float *mf)
{
  *mf = (mf1+mf2-mf1*mf2);
}

void TCoNormBoundedSum(float mf1, float mf2, float *mf)
{
  if ((mf1-mf2)<=1)
   *mf = mf1-mf2;
  else
   *mf = 1;
}

void TCoNormDrasticSum(float mf1, float mf2, float *mf)
{
  if (mf2 == 0.0)
    *mf = mf1;
  else if (mf1 == 0.0)
    *mf = mf2;
  else
    *mf = 1.0;
}
```

```
void TCoNormDisjointSum(float mf1, float mf2, float *mf)
{
 float min1, min2;

 min1 = 1-mf2;
 if (mf1<min1)
   min1 = mf1;
 min2 = 1-mf1;
 if (mf2<min2)
   min2 = mf2;
 if (min1>min2)
   *mf = min1;
 else
   *mf = min2;
}
```

FLC.H

```
#ifndef FLC_H
#define FLC_H
#define MAX_TEXT              50
#define MIN_DIST             -101          /* universe of distances */
#define  MAX_DIST            1101
#define NUM_DIST_SETS          9
#define NUM_IRSENSORS          8
#define VPBmin               -100          /* min IRS = max distance */
#define VPBct                 0.0          /* very positive big */
#define VPBmax                100
#define PBmin                 75
#define PBct                 150.0         /* positive big */
#define PBmax                 225
#define PMmin                 200
#define PMct                 300.0         /* positive medium */
#define PMmax                 400
#define PSmin                 375
#define PSct                 450.0         /* positive small */
#define PSmax                 525
#define ZEmin                 475
#define ZEct                 550.0         /* zero */
#define ZEmax                 625
#define NSmin                 575
```

```
#define NSct                    650.0           /* negative small */
#define NSmax                    725
#define NMmin                    700
#define NMct                    750.0           /* negative medium */
#define NMmax                    800
#define NBmin                    775
#define NBct                    850.0           /* negative big */
#define NBmax                    925
#define VNBmin                   900
#define VNBct                  1000.0            /* very negative big */
#define VNBmax                   1100            /* max IRS  = min distance */
#define MIN_SPEED                -11             /* universe of speeds */
#define MAX_SPEED                 11
#define NUM_SPEED_SETS            7
#define NUM_MOTORS                2
#define VFmax                     11
#define VFct                    10.0              /* very fast */
#define VFmin                     9
#define Fmax                      10
#define Fct                     7.0               /* fast */
#define Fmin                      4
#define PModmax                   5
#define PModct                  3.0               /* positive moderate */
#define PModmin                   1
#define NULLmax                   2
#define NULLct                  0.0               /* null */
#define NULLmin                  -2
#define NModmin                  -1
#define NModct                  -3.0              /* negative moderate */
#define NModmax                 -5
#define Smin                    -4
#define Sct                     -7.0              /* slow */
#define Smax                    -10
#define VSmin                   -9
#define VSct                    -10.0             /* very slow */
#define VSmax                   -11
#define HDSmin                  -20
#define HDSct                   -30               /* highly decrease speed */
#define HDSmax                  -40
#define MDSmin                  -10
#define MDSct          -20                  /* moderatly decrease speed */
#define MDSmax                  -30
#define LDSmin                   0
#define LDSct                   -10               /* lightly decrease speed */
#define LDSmax                  -20
#define NMSmin                  -10
#define NMSct                    0                /* no modify speed */
#define NMSmax                   10
#define LISmin                   0
#define LISct                   10                /* lightly increase speed */
#define LISmax                  20
#define MISmin                  10
#define MISct                   20                /* moderatly increase speed */
#define MISmax                  30
#define HISmin                  20
#define HISct                   30                /* highly increase speed */
#define HISmax                  40
```

```
struct PartitionMember
{
  short int              IntervalMember[2];
  struct FuzzySet1       *FuzzySet;
};


struct PartitionOfDistances
{
  char                   Name[MAX_TEXT];
  short int              Interval[2];
  struct PartitionMember Members[NUM_DIST_SETS-1];
};


struct PartitionOfSpeeds
{
 char                    Name[MAX_TEXT];
 short int               Interval[2];
 struct PartitionMember  Members[NUM_SPEED_SETS-1];
};



struct PerceptionStates
{
 struct FuzzySet1 *LeftState;
 struct FuzzySet1 *RightState;
 struct FuzzySet1 *BackState;
};


struct SpeedVariation
{
 struct FuzzySet1 *LeftVariation;
 struct FuzzySet1 *RightVariation;
};


struct RuleComponent
{
 struct FuzzySet1     *Antecedent1;
 struct FuzzySet1     *Antecedent2;
 struct FuzzySet1     *Consequent;
 struct RuleComponent *Next;
};



struct RuleBase
{
 struct RuleComponent  *Left;
 struct RuleComponent  *Right;
 struct RuleComponent  *Back;
 char                  Name[MAX_TEXT];
```

```
};


extern struct FuzzySet1 *CreateEmptyFuzzySet1();
extern struct FuzzySet1 *CreateFuzzySet1(char name[MAX_TEXT],
                                         char membership_function[MAX_TEXT],
                                        short int universe_of_discourse[2],
                                        short int support[2],
                                         float center);
extern void PrintFuzzySet1(struct FuzzySet1 *fuzzy_set);
extern FreeFuzzySet1(struct FuzzySet1 *fuzzy_set);
extern void WriteFuzzySet1ToFile(struct FuzzySet1 *fuzzy_set,FILE *file);
extern void ReadFuzzySet1FromFile(struct FuzzySet1 *fuzzy_set,FILE *file);
extern struct Value1 *FindValue1(struct FuzzySet1 *fuzzy_set,
                                 short int X);
extern short int DecFindCrispValue1(struct FuzzySet1 *fuzzy_set, float mf,
                                    short int init, short int end);
extern short int IncFindCrispValue1(struct FuzzySet1 *fuzzy_set, float mf,
                                    short int init, short int end);
extern float Sum(struct FuzzySet1 *fuzzy_set);
extern struct FuzzySet2 *CreateFuzzySet2(char name[MAX_TEXT],
                                         char membership_function[MAX_TEXT],
                                        short int universe_of_discourseX[2],
                                         short int universe_of_discourseY[2],
                                         short int supportX[2],
                                         short int supportY[2],
                                         float centerX,
                                         float centerY);
extern void PrintFuzzySet2(struct FuzzySet2 *fuzzy_set);
extern void FreeFuzzySet2(struct FuzzySet2 *fuzzy_set);
extern struct FuzzySet2 *CreateEmptyFuzzySet2();
extern void WriteFuzzySet2ToFile(struct FuzzySet2 *fuzzy_set,FILE *file);
extern void ReadFuzzySet2FromFile(struct FuzzySet2 *fuzzy_set,FILE *file);
extern struct Value2 *FindValue2(struct FuzzySet2 *fuzzy_set,
                                 short int X,
                                 short int Y);
extern struct FuzzySet1 *Union1(struct FuzzySet1 *A,
                                 struct FuzzySet1 *B,
                                char name[MAX_TEXT],
                                 short int univ[2]);
extern struct FuzzySet2 *CreateNullCartesianProduct11(short int UnivU[2],
                                                      short int UnivV[2]);
extern void TNormIntersection(float mf1, float mf2, float *mf);
extern void TNormProduct(float mf1, float mf2, float *mf);
extern void TNormBoundedProduct(float mf1, float mf2, float *mf);
extern void TNormDrasticProduct(float mf1, float mf2, float *mf);
extern void TCoNormUnion(float mf1, float mf2, float *mf);
extern void TCoNormSum(float mf1, float mf2, float *mf);
extern void TCoNormBoundedSum(float mf1, float mf2, float *mf);
extern void TCoNormDrasticSum(float mf1, float mf2, float *mf);
extern void TCoNormDisjointSum(float mf1, float mf2, float *mf);

#endif
```

FLC.C

```
#include "../SRC/include.h"
#include "../SRC/types.h"
#include "flc.h"

struct Robot *robot;

boolean PositiveBigSpeedMotor(struct Robot *r, short int motor)
{
  return( r->Motor[motor].Value <= SPBmax &&
          r->Motor[motor].Value >= SPBmin );
}


boolean PositiveBigSpeed(struct Robot *r)
{
  return(PositiveBigSpeedMotor(r,LEFT) ||
         PositiveBigSpeedMotor(r,RIGHT) );
}


boolean PositiveMediumSpeedMotor(struct Robot *r, short int motor)
{
   return( r->Motor[motor].Value <= SPMmax  &&
             r->Motor[motor].Value >= SPMmin );
}


boolean PositiveMediumSpeed(struct Robot *r)
{
   return( PositiveMediumSpeedMotor(r,LEFT) ||
           PositiveMediumSpeedMotor(r,RIGHT) );

}


boolean PositiveSmallSpeedMotor(struct Robot *r, short int motor)
{
   return( r->Motor[motor].Value <= SPSmax &&
           r->Motor[motor].Value >= SPSmin );
}


boolean PositiveSmallSpeed(struct Robot *r)
{
    return( PositiveSmallSpeedMotor(r,LEFT) ||
            PositiveSmallSpeedMotor(r,RIGHT) );
}


boolean ZeroSpeedMotor(struct Robot *r, short int motor)
{
   return( r->Motor[motor].Value <= SZEmax &&
           r->Motor[motor].Value >= SZEmin );
}
```

```
boolean ZeroSpeed(struct Robot *r)
{
     return( ZeroSpeedMotor(r,LEFT) ||
             ZeroSpeedMotor(r,RIGHT) );
}


boolean NegativeSmallSpeedMotor(struct Robot *r, short int motor)
{
    return( r->Motor[motor].Value <= SNSmin &&
            r->Motor[motor].Value >= SNSmax );
}


boolean NegativeSmallSpeed(struct Robot *r)
{
     return( NegativeSmallSpeedMotor(r,LEFT) ||
             NegativeSmallSpeedMotor(r,RIGHT) );
}


boolean NegativeMediumSpeedMotor(struct Robot *r, short int motor)
{
    return( r->Motor[motor].Value <= SNMmin  &&
            r->Motor[motor].Value >= SNMmax );
}



boolean NegativeMediumSpeed(struct Robot *r)
{
     return( NegativeMediumSpeedMotor(r,LEFT) ||
             NegativeMediumSpeedMotor(r,RIGHT) );
}


boolean NegativeBigSpeedMotor(struct Robot *r, int motor)
{
     return( r->Motor[motor].Value <= SNBmin  &&
             r->Motor[motor].Value >= SNBmax );
}


boolean NegativeBigSpeed(struct Robot *r)
{
    return( NegativeBigSpeedMotor(r,LEFT) ||
            NegativeBigSpeedMotor(r,RIGHT) );
}


boolean PositiveBigDistance(struct Robot *r, short int sensor)
{
     return( r->IRSensor[sensor].DistanceValue <= MPBmax  &&
             r->IRSensor[sensor].DistanceValue >= MPBmin  );
}
```

```
boolean PositiveMediumDistance(struct Robot *r, short int sensor )
{
    return( r->IRSensor[sensor].DistanceValue <= MPMmax &&
            r->IRSensor[sensor].DistanceValue >= MPMmin );
}

boolean ObstaclePerceivedOnRight(struct Robot *r)
{
    return( PositiveBigDistance(r,3)   ||
            PositiveBigDistance(r,4)   ||
            PositiveBigDistance(r,5)   ||
            PositiveMediumDistance(r,3) ||
            PositiveMediumDistance(r,4) ||
            PositiveMediumDistance(r,5) );
}


boolean ObstaclePerceivedOnLeft(struct Robot *r)
{
        return( PositiveBigDistance(r,0)   ||
                PositiveBigDistance(r,1)   ||
                PositiveBigDistance(r,2)   ||
                PositiveMediumDistance(r,0) ||
                PositiveMediumDistance(r,1) ||
                PositiveMediumDistance(r,2) );
}

boolean ObstaclePerceivedInBack(struct Robot *r)
{
        return( PositiveBigDistance(r,6)   ||
                PositiveBigDistance(r,7)   ||
                PositiveMediumDistance(r,6) ||
                PositiveMediumDistance(r,7) );
}

boolean PositiveSmallDistance(struct Robot *r, short int sensor )
{
        return( r->IRSensor[sensor].DistanceValue <= MPSmax  &&
              r->IRSensor[sensor].DistanceValue >= MPSmin );
}

boolean ZeroDistance(struct Robot *r, short int sensor )
{
        return( r->IRSensor[sensor].DistanceValue <= MZEmax  &&
              r->IRSensor[sensor].DistanceValue >= MZEmin );
}

boolean ObstacleLightlyPerceivedOnRight(struct Robot *r)
{

        return( PositiveSmallDistance(r,3) ||
                PositiveSmallDistance(r,4) ||
```

```
                PositiveSmallDistance(r,5) ||
                ZeroDistance(r,3)          ||
                ZeroDistance(r,4)          ||
                ZeroDistance(r,5) );
}

boolean ObstacleLightlyPerceivedOnLeft(struct Robot *r)
{
        return( PositiveSmallDistance(r,0) ||
                PositiveSmallDistance(r,1) ||
                PositiveSmallDistance(r,2) ||
                ZeroDistance(r,0)          ||
                ZeroDistance(r,1)          ||
                ZeroDistance(r,2) );
}

boolean ObstacleLightlyPerceivedInBack(struct Robot *r)
{
        return( PositiveSmallDistance(r,6) ||
                PositiveSmallDistance(r,7) ||
                ZeroDistance(r,6)          ||
                ZeroDistance(r,7) );
}

boolean NegativeSmallDistance(struct Robot *r, short int sensor)
{
        return( r->IRSensor[sensor].DistanceValue <= MNSmin  &&
             r->IRSensor[sensor].DistanceValue >= MNSmax );
}

boolean NegativeMediumDistance(struct Robot *r, short int sensor)
{
        return( r->IRSensor[sensor].DistanceValue <= MNMmin &&
             r->IRSensor[sensor].DistanceValue >= MNMmax );
}

boolean NegativeBigDistance(struct Robot *r, short int sensor)
{
        return( r->IRSensor[sensor].DistanceValue <= MNBmin &&
                r->IRSensor[sensor].DistanceValue >= MNBmax );
}

boolean NoObstaclePerceivedOnRight(struct Robot *r)
{
        return( NegativeSmallDistance(r,3)  ||
                NegativeSmallDistance(r,4)  ||
                NegativeSmallDistance(r,5)  ||
                NegativeMediumDistance(r,3) ||
                NegativeMediumDistance(r,4) ||
                NegativeMediumDistance(r,5) ||
                NegativeBigDistance(r,3)    ||
                NegativeBigDistance(r,4)    ||
                NegativeBigDistance(r,5) );
```

```
}

boolean NoObstaclePerceivedOnLeft(struct Robot *r)
{
        return( NegativeSmallDistance(r,0)  ||
                NegativeSmallDistance(r,1)  ||
                NegativeSmallDistance(r,2)  ||
                NegativeMediumDistance(r,0) ||
                NegativeMediumDistance(r,1) ||
                NegativeMediumDistance(r,2) ||
                NegativeBigDistance(r,0)    ||
                NegativeBigDistance(r,1)    ||
                NegativeBigDistance(r,2) );
}

boolean NoObstaclePerceivedInBack(struct Robot *r)
{
        return( NegativeSmallDistance(r,6)  ||
                NegativeSmallDistance(r,7)  ||
                NegativeMediumDistance(r,6) ||
                NegativeMediumDistance(r,7) ||
                NegativeBigDistance(r,6)    ||
                NegativeBigDistance(r,7) );
}

boolean CollisionOnLeft(struct Robot *r)
{
        return( PositiveBigDistance(r,0) ||
                PositiveBigDistance(r,1) ||
                PositiveBigDistance(r,2) );
}

boolean CollisionOnRight(struct Robot *r)
{
        return( PositiveBigDistance(r,3) ||
                PositiveBigDistance(r,4) ||
                PositiveBigDistance(r,5) );
}

boolean CollisionInBack(struct Robot *r)
{
        return( PositiveBigDistance(r,6) ||
                PositiveBigDistance(r,7) );
}

void LightlyIncreaseSpeed(struct Robot *robot)
{
        robot->Motor[LEFT].Value  += 1;
        robot->Motor[RIGHT].Value += 1;
}
```

```
void IncreaseSpeed(struct Robot *robot)
{
        robot->Motor[LEFT].Value  += 2;
        robot->Motor[RIGHT].Value += 2;
}


void LightlyDecreaseSpeed(struct Robot *robot)
{
        robot->Motor[LEFT].Value  -= 1;
        robot->Motor[RIGHT].Value -= 1;
}

void DecreaseSpeed(struct Robot *robot)
{
        robot->Motor[LEFT].Value  -= 2;
        robot->Motor[RIGHT].Value -= 2;
}


void TurnRight(struct Robot *robot)
{
        robot->Motor[LEFT].Value  = SPSmax;
        robot->Motor[RIGHT].Value = SNMmin;
}


void TurnLeft(struct Robot *robot)
{
        robot->Motor[LEFT].Value  = SNMmin;
        robot->Motor[RIGHT].Value = SPSmin;
}


boolean DifferentEffectorsDirection(struct Robot *robot)
{
        return (( robot->Motor[LEFT].Value ) != ( robot->Motor[RIGHT].Value ));

}


void CorrectDirection(struct Robot *robot)
{
        if (( robot->Motor[LEFT].Value ) > ( robot->Motor[RIGHT].Value ))
                robot->Motor[RIGHT].Value = robot->Motor[LEFT].Value;
        if (( robot->Motor[RIGHT].Value ) > ( robot->Motor[LEFT].Value ))

                robot->Motor[LEFT].Value = robot->Motor[RIGHT].Value;
}


void SpeedDownMovement(struct Robot *r)
{
        if ( DifferentEffectorsDirection(r) )
                if ( PositiveSmallSpeed(r) || ZeroSpeed(r) )
                        CorrectDirection(r);
                else
                        LightlyDecreaseSpeed(r);
```

```
        else
                DecreaseSpeed(r);
}

void SpeedUpMovement(struct Robot *r)
{
        if ( DifferentEffectorsDirection(r) )
                if ( PositiveSmallSpeed(r) || ZeroSpeed(r) )
                        CorrectDirection(r);
                else
                        LightlyIncreaseSpeed(r);
        else
                IncreaseSpeed(r);
}

void SolveCollisionOnLeft(struct Robot *r)
{
        if ( ZeroSpeed(r) || PositiveSmallSpeed(r) )
                TurnRight(r);
        else
                LightlyDecreaseSpeed(r);
}

void SolveCollisionOnRight(struct Robot *r)
{
        boolean ans = FALSE;
        ans = ( ZeroSpeed(r) || PositiveSmallSpeed(r));
        if ( ans = TRUE )
                TurnLeft(r);
        if ( ans = FALSE )
                DecreaseSpeed(r);
}

void CorrectRobotEffectors(struct Robot *r)
{
        int i;
        for(i=0; i<=1; ++i)
        {
                if (r->Motor[i].Value > SPBmax)
                        r->Motor[i].Value = SPBmax;
                if (r->Motor[i].Value < SNBmax)
                        r->Motor[i].Value = SNBmax;
        }
}
```

DATABASE.C

```
#include "../SRC/include.h"
#include "../SRC/types.h"
#include "flc.h"
#include "fuzzysets.h"
struct Robot *robot;


/* Primary Fuzzy Distances */

struct FuzzySet1 *vpb, *pb, *pm, *ps, *ze, *ns, *nm, *nb, *vnb;

/* Primary Fuzzy Speeds */

struct FuzzySet1 *vf, *f, *pmod, *nulla, *nmod, *s, *vs;

/* Supports of primary Fuzzy Sets */

short int univ_distances[2] = {MIN_DIST,MAX_DIST};
short int     supportVPB[2] = {VPBmin,VPBmax};
short int     supportPB[2] = {PBmin,PBmax};
short int     supportPM[2] = {PMmin,PMmax};
short int     supportPS[2] = {PSmin,PSmax};
short int     supportZE[2] = {ZEmin,ZEmax};
short int     supportNS[2] = {NSmin,NSmax};
short int     supportNM[2] = {NMmin,NMmax};
short int     supportNB[2] = {NBmin,NBmax};
short int     supportVNB[2] = {VNBmin,VNBmax};
short int     univ_speeds[2] = {MIN_SPEED,MAX_SPEED};
short int     supportVF[2] = {VFmin,VFmax};
short int     supportF[2] = {Fmin,Fmax};
short int     supportPMod[2] = {PModmin,PModmax};
short int     supportNULL[2] = {NULLmin,NULLmax};
short int     supportNMod[2] = {NModmax,NModmin};
short int     supportS[2] = {Smax,Smin};
short int     supportVS[2] = {VSmax,VSmin};


/*****************************************************************************/
/* PRIMARY FUZZY SETS: generation of fuzzy distances and fuzzy speeds */
/*****************************************************************************/


void CreatePrimaryFuzzyDistances()
{
  vpb = CreateFuzzySet1("VPB",
                        "Triangular1",univ_distances,supportVPB,VPBct);
  pb  = CreateFuzzySet1("PB",
                        "Triangular1",univ_distances,supportPB,PBct);
  pm  = CreateFuzzySet1("PM",
                      "Triangular1",univ_distances,supportPM,PMct);
  ps  = CreateFuzzySet1("PS",
                        "Triangular1",univ_distances,supportPS,PSct);
  ze  = CreateFuzzySet1("ZE",
                        "Triangular1",univ_distances,supportZE,ZEct);
  ns  = CreateFuzzySet1("NS",
```

```
                              "Triangular1",univ_distances,supportNS,NSct);
  nm  = CreateFuzzySet1("NM",
                              "Triangular1",univ_distances,supportNM,NMct);
  nb  = CreateFuzzySet1("NB",
                              "Triangular1",univ_distances,supportNB,NBct);
  vnb = CreateFuzzySet1("VNB",
                              "Triangular1",univ_distances,supportVNB,VNBct);
}


void CreatePrimaryFuzzySpeeds()
{
  vf   = CreateFuzzySet1("VF",
                              "Triangular1",univ_speeds,supportVF,VFct);
  f    = CreateFuzzySet1("F",
                              "Triangular1",univ_speeds,supportF,Fct);
  pmod  = CreateFuzzySet1("PMod",
                              "Triangular1",univ_speeds,supportPMod,PModct);
  nulla = CreateFuzzySet1("NULL",
                              "Triangular1",univ_speeds,supportNULL,NULLct);
  nmod  =  CreateFuzzySet1("NMod",
                               "Triangular1",univ_speeds,supportNMod,NModct);
  s    = CreateFuzzySet1("S",
                              "Triangular1",univ_speeds,supportS,Sct);
  vs   = CreateFuzzySet1("VS",
                              "Triangular1",univ_speeds,supportVS,VSct);
}



/*****************************************************************/
/* FUZZY PARTITIONS: completeness of the database */
/*****************************************************************/


short int LimitPartitionMember(struct FuzzySet1 *fs1,
                               struct FuzzySet1 *fs2)
/* fs1 and fs2 may be ordered */
{
 struct Value1 *search1, *search2;
 short int element, intersection_1, intersection_2;

 search1 = (struct Value1 *)malloc(sizeof(struct Value1));
 search2 = (struct Value1 *)malloc(sizeof(struct Value1));
 intersection_1=fs2->Support[0];
 intersection_2=fs1->Support[1];
 element = intersection_1;
 if ((abs(intersection_2) - intersection_1) > 0)
 {
   search1 = FindValue1(fs1,intersection_1);
   search2 = FindValue1(fs2,intersection_1);
   while (search1 && search2)
   {
    if ((search2->MembershipValue - search1->MembershipValue) > 0.0 )
    {
       element = (search1->CrispX) - 1;
       search1 = NULL;
       search2 = NULL;
```

```
      }
      else
      {
        search1 = search1->Next;      /* next value of the support of fs1 */
        search2 = search2->Next;
      }
     }
    }
    return(element);
}


struct PartitionOfDistances *CreateEmptyPartitionOfDistances()
{
  struct PartitionOfDistances *partition;
  short int num;

  partition = (struct PartitionOfDistances *)
              malloc(sizeof(struct PartitionOfDistances));
  strcpy(partition->Name,"new");
  partition->Interval[0] = 0;
  partition->Interval[1] = 0;
  for(num = 0;num < NUM_DIST_SETS;++num)
  {
    partition->Members[num].IntervalMember[0] = 0;
    partition->Members[num].IntervalMember[1] = 0;
    partition->Members[num].FuzzySet = (struct FuzzySet1 *)
                                       malloc(sizeof(struct FuzzySet1));
  }
  return(partition);
}


void FreeDistanceMembers(struct PartitionOfDistances *partition)
{
  short int num;

  for(num = 0;num < NUM_DIST_SETS - 1;++num)
   FreeFuzzySet1(partition->Members[num].FuzzySet);
}


struct PartitionOfDistances *CreatePartitionOfDistances(short int interval[2])
{
  struct PartitionOfDistances *partition;

  partition = CreateEmptyPartitionOfDistances();
  strcpy(partition->Name,"FUZZY PARTITIONS OF DISTANCE VALUES");
  partition->Interval[0] = interval[0];
  partition->Interval[1] = interval[1];     /* to be modified with previous*/
                           /* connection of fuzzy distances */

  partition->Members[0].FuzzySet        = vpb;
  partition->Members[0].IntervalMember[0] = interval[0];
  partition->Members[0].IntervalMember[1] = LimitPartitionMember(vpb,pb);
```

```
 partition->Members[1].FuzzySet        = pb;
 partition->Members[1].IntervalMember[0] = partition->Members[0].IntervalMember[1]+1;
 partition->Members[1].IntervalMember[1] = LimitPartitionMember(pb,pm);
 partition->Members[2].FuzzySet     = pm;
 partition->Members[2].IntervalMember[0] = partition->Members[1].IntervalMember[1]+1;
 partition->Members[2].IntervalMember[1] = LimitPartitionMember(pm,ps);
 partition->Members[3].FuzzySet     = ps;
 partition->Members[3].IntervalMember[0] = partition->Members[2].IntervalMember[1]+1;
 partition->Members[3].IntervalMember[1] = LimitPartitionMember(ps,ze);
 partition->Members[4].FuzzySet     = ze;
 partition->Members[4].IntervalMember[0] = partition->Members[3].IntervalMember[1]+1;
 partition->Members[4].IntervalMember[1] = LimitPartitionMember(ze,ns);
 partition->Members[5].FuzzySet     = ns;
 partition->Members[5].IntervalMember[0] = partition->Members[4].IntervalMember[1]+1;
 partition->Members[5].IntervalMember[1] = LimitPartitionMember(ns,nm);
 partition->Members[6].FuzzySet     = nm;
 partition->Members[6].IntervalMember[0] = partition->Members[5].IntervalMember[1]+1;
 partition->Members[6].IntervalMember[1] = LimitPartitionMember(nm,nb);
 partition->Members[7].FuzzySet     = nb;
 partition->Members[7].IntervalMember[0] =  partition->Members[6].IntervalMember[1]+1;
 partition->Members[7].IntervalMember[1] = LimitPartitionMember(nb,vnb);
 partition->Members[8].FuzzySet     = vnb;
 partition->Members[8].IntervalMember[0] = partition->Members[7].IntervalMember[1]+1;
 partition->Members[8].IntervalMember[1] = partition->Interval[1];

 return(partition);

}


struct PartitionOfSpeeds *CreateEmptyPartitionOfSpeeds()
{
 struct PartitionOfSpeeds *partition;
 short int num;

 partition = (struct PartitionOfSpeeds *)malloc(sizeof(struct PartitionOfSpeeds));
 strcpy(partition->Name,"new");
 partition->Interval[0] = 0;
 partition->Interval[1] = 0;
 for(num = 0;num < NUM_SPEED_SETS;++num)
 {
  partition->Members[num].IntervalMember[0] = 0;
  partition->Members[num].IntervalMember[1] = 0;
  partition->Members[num].FuzzySet = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 }
 return(partition);
}



struct PartitionOfSpeeds *CreatePartitionOfSpeeds(short int interval[2])
{
 struct PartitionOfSpeeds *partition;

 partition = CreateEmptyPartitionOfSpeeds();
 strcpy(partition->Name,"FUZZY PARTITIONS OF SPEED VALUES");
 partition->Interval[0] = interval[0];
 partition->Interval[1] = interval[1];
```

```
  partition->Members[0].FuzzySet        = vs;
  partition->Members[0].IntervalMember[0] = interval[0];
  partition->Members[0].IntervalMember[1] = LimitPartitionMember(vs,s);
  partition->Members[1].FuzzySet        = s;
  partition->Members[1].IntervalMember[0] = partition->Members[0].IntervalMember[1]+1;
  partition->Members[1].IntervalMember[1] = LimitPartitionMember(s,nmod);
  partition->Members[2].FuzzySet     = nmod;
  partition->Members[2].IntervalMember[0] = partition->Members[1].IntervalMember[1]+1;
  partition->Members[2].IntervalMember[1] = LimitPartitionMember(nmod,nulla);
  partition->Members[3].FuzzySet     = nulla;
  partition->Members[3].IntervalMember[0] = partition->Members[2].IntervalMember[1]+1;
  partition->Members[3].IntervalMember[1] = LimitPartitionMember(nulla,pmod);
  partition->Members[4].FuzzySet     = pmod;
  partition->Members[4].IntervalMember[0] = partition->Members[3].IntervalMember[1]+1;
  partition->Members[4].IntervalMember[1] = LimitPartitionMember(pmod,f);
  partition->Members[5].FuzzySet     = f;
  partition->Members[5].IntervalMember[0] = partition->Members[4].IntervalMember[1]+1;
  partition->Members[5].IntervalMember[1] = LimitPartitionMember(f,vf);
  partition->Members[6].FuzzySet     = vf;
  partition->Members[6].IntervalMember[0] = partition->Members[5].IntervalMember[1]+1;
  partition->Members[6].IntervalMember[1] = interval[1];

  return(partition);
}


void WritePartitionOfDistancesToFile(struct PartitionOfDistances *partition, FILE *file)
{
  short int num;
  fprintf(file,"#File      :FUZZY|distancespartition\n");
  fprintf(file,"#Description: %s\n",partition->Name);
  fprintf(file,"# Author    : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date      :      \n");
  fprintf(file,"-----------------------------------------------------\n\n\n");

  fprintf(file," - Interval of crisp distances: [%d,%d]\n",partition->Interval[0],
                                                          partition->Interval[1]);
  fprintf(file," - Members:\n\n");
  for(num = 0; num < NUM_DIST_SETS; ++num)
  {
    fprintf(file," \t-Number of fuzzy distance member : %d\n",num);
    fprintf(file," \t-Interval of crisp distances: [%d,%d]\n",
                                          partition->Members[num].IntervalMember[0],
                                          partition->Members[num].IntervalMember[1]);
    fprintf(file," \t-Fuzzy distance set member: %s\n\n",
                                          partition->Members[num].FuzzySet);
    fprintf(file," \t\t-XSupport: [%d,%d]\n",
                                      partition->Members[num].FuzzySet->Support[0],
                                      partition->Members[num].FuzzySet->Support[1]);
    fprintf(file," \t\t-Center:   %f\n", partition->Members[num].FuzzySet->Center);
    fprintf(file," \t\t-Membership Function Type:  %s\n\n\n\n",
            partition->Members[num].FuzzySet->Properties.MembershipFunctionType);
  }
}



void WritePartitionOfSpeedsToFile(struct PartitionOfSpeeds *partition, FILE *file)
```

```
{
  short int num;
  fprintf(file,"# File      : FUZZY|speedspartition\n");
  fprintf(file,"# Description: %s\n",partition->Name);
  fprintf(file,"# Author    : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date      :     \n");
  fprintf(file,"-------------------------------------------------\n\n\n");
  fprintf(file,"  - Interval of crisp speed values: [%d , %d]\n",partition->Interval[0],
                                                                  partition->Interval[1]);
  fprintf(file,"  - Members:\n\n");
  for(num = 0; num < NUM_SPEED_SETS; ++num)
  {
    fprintf(file," \t-Number of fuzzy speed member: %d\n",num);
    fprintf(file," \t-Interval of crisp speed values: [%d,%d]\n",
                                          partition->Members[num].IntervalMember[0],
                                          partition->Members[num].IntervalMember[1]);
    fprintf(file," \t-Fuzzy speed set member: %s\n\n",
                                              partition->Members[num].FuzzySet);
    fprintf(file," \t\t-XSupport: [%d,%d] \n",
                                        partition->Members[num].FuzzySet->Support[0],
                                        partition->Members[num].FuzzySet->Support[1]);
    fprintf(file," \t\t-Center:   %f\n",
                                        partition->Members[num].FuzzySet->Center);
    fprintf(file," \t\t-Membership Function Type:  %s\n\n\n\n",
            partition->Members[num].FuzzySet->Properties.MembershipFunctionType);
  }
}


void FreeSpeedMembers(struct PartitionOfSpeeds *partition)
{
 short int num;

  for(num = 0;num < NUM_SPEED_SETS - 1;++num)
  FreeFuzzySet1(partition->Members[num].FuzzySet);
}


short int FindMaxPartitionMember(short int IR1, short int IR2,
                                 short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
 short int max_partition_member, i;

 max_partition_member = KheperaSensorFuzzyValues[IR1];
 for(i = IR1; i <= IR2; ++i)
  if((KheperaSensorFuzzyValues[i] - max_partition_member ) > 0)
    max_partition_member = i;
 return(max_partition_member);
}


/***************************************/
/*  FUZZIFICATION STRATEGY  */
/***************************************/



short int  FindDistancePartitionMember(short int crisp_distance,
```

```
                                    struct PartitionOfDistances *partition)

{
  struct PartitionMember member;
  short int found, num;

  num = 0;
  found = -1;
  member = partition->Members[num];
  while((found == -1) && ((NUM_DIST_SETS - num) > 0 ))
  {
    if(((crisp_distance - member.IntervalMember[0]) >= 0) &&
          ((crisp_distance - member.IntervalMember[1]) <= 0))
          found = num;
    else
    {
      num++;
      member = partition->Members[num];
    }
  }
  return(found);
}


short int FindSpeedPartitionMember(short int crisp_speed,
                                   struct PartitionOfSpeeds *partition)

{
  struct PartitionMember member;
  short int num, found;
  num = 0;
  found = -1;
  member = partition->Members[num];
  while((found == -1) && ((NUM_SPEED_SETS - num) > 0 ))
  {
    if(((crisp_speed - member.IntervalMember[0]) >= 0) &&
          ((crisp_speed - member.IntervalMember[1]) <= 0))
          found = num;
    else
    {
      num++;
      member = partition->Members[num];
    }
  }
  return(found);
}


short int  SensorValueFuzzification( struct Robot *robot,
                                     short int sensor,
                                     struct PartitionOfDistances *partition)
{
  short int dist_member;

  dist_member =FindDistancePartitionMember(robot->IRSensor[sensor].DistanceValue, partition);
  return(dist_member);
}
```

```
short int MotorValueFuzzification(struct Robot *robot,
                                  short int motor,
                                  struct PartitionOfSpeeds *partition)
{
 short int speed_member;

 speed_member = FindSpeedPartitionMember(robot->Motor[motor].Value, partition);

 return(speed_member);
}


void CompleteDistanceFuzzification(struct Robot *robot,
                                   struct PartitionOfDistances *partition,
                                   short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
 short int num_sensor;

 for(num_sensor = 0; num_sensor < NUM_IRSENSORS; ++num_sensor)
 {
  KheperaSensorFuzzyValues[num_sensor] = SensorValueFuzzification(robot, num_sensor, partition);
 }
}



void CompleteSpeedFuzzification(struct Robot *robot,
                                struct PartitionOfSpeeds *partition,
                                short int KheperaMotorFuzzyValues[NUM_MOTORS-1])
{
 short int num_motor;

 for(num_motor = 0; num_motor < NUM_MOTORS; ++num_motor)
 {
  KheperaMotorFuzzyValues[num_motor] = MotorValueFuzzification(robot, num_motor, partition);
 }
}







void WriteCurrentFuzzyDistancesToFile(FILE *file,
                                      short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1],
                                      struct PartitionOfDistances *partition,
                                      short int step)
{
  short int sensor,num;
  struct FuzzySet1 *member;
  fprintf(file,"# File       : FUZZY\fuzzy_distancessensors\n");
  fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Description : Current FUZZY DISTANCES of Khepera\n");
  fprintf(file,"# Date       :      \n");
```

```
  fprintf(file," ----------------------------------------------------\n");
  fprintf(file,"  *STEP: %d\n",step);
  fprintf(file," ----------------------------------------------------\n\n\n");
  fprintf(file,"  *Khepera Infra Red Sensors:\n\n");
  for(sensor = 0; sensor < NUM_IRSENSORS; ++sensor)
  {
    num = KheperaSensorFuzzyValues[sensor];
    if (sensor==0 || sensor==1 || sensor==2)
      fprintf(file,"  IR Sensor %d LEFT\n",sensor);
    else if (sensor==3 || sensor==4 || sensor==5)
      fprintf(file,"  IR Sensor %d RIGHT\n",sensor);
    else /* (sensor==6 || sensor==7) */
      fprintf(file,"  IR Sensor %d BACK\n",sensor);
    fprintf(file," \t-Number of fuzzy distance member: %d\n",num);
    fprintf(file," \t-Interval of crisp sensor values: [%d,%d]\n",
                                          partition->Members[num].IntervalMember[0],
                                          partition->Members[num].IntervalMember[1]);
    member = partition->Members[num].FuzzySet;
    fprintf(file," \t-Fuzzy distance set member: %s\n\n", member->Properties.Name);
    fprintf(file," \t\t-XSupport: [%d,%d]", member->Support[0], member->Support[1]);
    fprintf(file," \t-Center:   %f\n\n", member->Center);
  }
}


void WriteCurrentFuzzySpeedsToFile(FILE *file,
                                   short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                   struct PartitionOfSpeeds *partition,
                                   short int step)
{
 short int motor,num;
  struct FuzzySet1 *member;

  fprintf(file,"# File       : FUZZY\fuzzy_speedsmotors\n");
  fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Description : Current FUZZY SPEEDS of Khepera\n");
  fprintf(file,"# Date       :      \n");
  fprintf(file," ----------------------------------------------------\n");
  fprintf(file,"  *STEP: %d\n",step);
  fprintf(file," ----------------------------------------------------\n\n\n");
  fprintf(file,"  *Khepera motors:\n\n");
  for(motor = 0; motor < NUM_MOTORS; ++motor)
  {
    num = KheperaMotorFuzzyValues[motor];
    if (motor == 0)
      fprintf(file,"  Motor %d: RIGHT\n",motor);
    else /*(motor = 1)*/
      fprintf(file,"  Motor %d: LEFT\n",motor);
    fprintf(file," \t-Number of fuzzy speed member: %d\n",num);
    fprintf(file," \t-Interval of crisp motor values: [%d,%d]\n",
                                          partition->Members[num].IntervalMember[0],
                                          partition->Members[num].IntervalMember[1]);
    member = partition->Members[num].FuzzySet;
    fprintf(file," \t-Fuzzy speed set member: %s\n\n",
                                           member->Properties.Name);
    fprintf(file," \t\t-XSupport: [%d,%d] ",
                                          member->Support[0],
                                          member->Support[1]);
```

```
    fprintf(file," \t-Center:   %f\n\n",member->Center);
  }
}
```

RULEBASE.C

```
#include "../SRC/include.h"
#include "../SRC/types.h"
#include "fuzzysets.h"
#include "flc.h"

#define NUM_PERCEP_STATES      5
#define NUM_VARIATIONS         7
#define MIN_VARIATION_SPEED   -40
#define MAX_VARIATION_SPEED    40
#define LEFT_SIDE              0
#define RIGHT_SIDE             1
#define BACK_SIDE              2
#define NUM_SIDE_RULES         70

#define FreeComponent(x)    free(x);

struct Robot *robot;

/*********************************************************/
/*   Supports of fuzzy speeds variations                 */
/*********************************************************/

short int supportHIS[2] = {HISmin, HISmax},
          supportMIS[2] = {MISmin, MISmax},
          supportLIS[2] = {LISmin, LISmax},
          supportNMS[2] = {NMSmin, NMSmax},
          supportLDS[2] = {LDSmax, LDSmin},
          supportMDS[2] = {MDSmax, MDSmin},
          supportHDS[2] = {HDSmax, HDSmin};


short int distances[2] = {MIN_DIST, MAX_DIST},
      variations[2] = {MIN_VARIATION_SPEED, MAX_VARIATION_SPEED};

extern struct FuzzySet1 *vpb, *pb, *pm, *ps, *ze, *ns, *nm, *nb, *vnb;
extern struct FuzzySet1 *vf, *f, *pmod, *nulla, *nmod, *s, *vs;



/*****************************************************/
/* FUZZY CONTROL RULES                               */
/*****************************************************/


/* States of perception */

struct FuzzySet1  *collision,
```

```
                *ohp,   /* obstacle highly perceived   */
                *omp,  /*   "    moderatly perceived */
                *olp,      /*   "    lightly perceived   */
                *nop;      /*   "    no perceived        */


/* Speed variations */

struct FuzzySet1 *his,          /* highly increase speed       */
                 *mis,          /* moderatly increase speed    */
                 *lis,          /* lightly increase speed      */
                 *nms,          /* no modify speed             */
                 *lds,          /* lightly decrease speed      */
                 *mds,          /* moderatly increase speed    */
                 *hds;          /* highly increase speed       */


void CreateFuzzySetsOfPerception()
{
 nop      = Union1(vpb,pb,"No Obstacle Perceived",distances);
 nop->Support[0] = 0;
 olp      = Union1(pm,ps, "Obstacle Lightly Perceived",distances);
 omp      = Union1(ze,ns,"Obstacle Moderatly Perceived",distances);
 ohp      = Union1(nm,nb,"Obstacle Lightly Perceived",distances);
 collision = Union1(vnb,vnb,"Collision",distances);
 collision->Support[1] = 1023;
}


short int CountEqualFuzzyDistances(short int s1, short int s2,
                                   short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1],
                                   short int partition_member_1,
                                   short int partition_member_2)
{
 short int number, sensor;

 number = 0;
 for(sensor = s1; sensor <= s2; ++sensor)
  if ((KheperaSensorFuzzyValues[sensor] == partition_member_1) ||
     (KheperaSensorFuzzyValues[sensor] == partition_member_2))
    number++;
 return(number);
}


struct FuzzySet1 *CreateSidePerceptionState(short int s1, short int s2,
                                            short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
 struct FuzzySet1 *perception;
 short int number;

 perception = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,0,1);
 if(number >= 1)
  perception = nop;
 number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,2,3);
```

```
 if(number >= 1)
  perception = olp;
 number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,4,5);
 if(number >= 1)
  perception = omp;
 number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,6,7);
 if(number >= 1)
  perception = ohp;
 number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,8,8);
 if(number >= 1)
  perception = collision;

 return(perception);
}


struct PerceptionStates *CreateFuzzyPerceptionStates(short int
                                                     KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
 struct PerceptionStates *perceptions;

 perceptions = (struct PerceptionStates *)malloc(sizeof(struct PerceptionStates));
 perceptions->LeftState = CreateSidePerceptionState(0,2,KheperaSensorFuzzyValues);
 perceptions->RightState = CreateSidePerceptionState(3,5,KheperaSensorFuzzyValues);
 perceptions->BackState = CreateSidePerceptionState(6,7,KheperaSensorFuzzyValues);

 return(perceptions);
}


void WriteFuzzyPerceptionStatesToFile(struct PerceptionStates *perceptions,
                                      struct Robot *robot,
                                      short int step,
                                      FILE *file)
{

 short int sensor;

 fprintf(file,"# File       : FUZZY|fuzzy.perceptions\n");
 fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel.\n");
 fprintf(file,"# Description : Current FUZZY PERCEPTION states of Khepera\n");
 fprintf(file,"# Date       :     \n");
 fprintf(file," ------------------------------------------------------------\n");
 fprintf(file,"  *STEP: %d\n",step);
 fprintf(file," ------------------------------------------------------------\n\n\n");
 fprintf(file,"    - IR Sensors Crisp Distance Values:\n");
 for(sensor = 0; sensor < NUM_IRSENSORS; ++sensor)
    if (sensor==0 || sensor==1 || sensor==2)
     fprintf(file,"        IR Sensor %d (LEFT) : %d\n",sensor,
                                      robot->IRSensor[sensor].DistanceValue);
    else if (sensor==3 || sensor==4 || sensor==5)
     fprintf(file,"        IR Sensor %d (RIGHT): %d\n",sensor,
                                      robot->IRSensor[sensor].DistanceValue);
    else /* (sensor==6 || sensor==7) */
     fprintf(file,"        IR Sensor %d (BACK) : %d\n",sensor,
                                      robot->IRSensor[sensor].DistanceValue);

 fprintf(file,"    - LEFT IR Sensors interpretation : %s\n",
```

```
                        perceptions->LeftState->Properties.Name);
  fprintf(file,"   - RIGHT IR Sensors interpretation: %s\n",
                        perceptions->RightState->Properties.Name);
  fprintf(file,"   - BACK IR Sensors interpretation : %s\n",
                        perceptions->BackState->Properties.Name);

}


void CreateFuzzySetsOfSpeedVariation()
{
  his = CreateFuzzySet1("Highly_increase_speed","Triangular1",
                        variations,supportHIS,HISct);
  mis = CreateFuzzySet1("Moderatly_increase_speed","Triangular1",
                        variations,supportMIS,MISct);
  lis = CreateFuzzySet1("Lightly_increase_speed","Triangular1",
                        variations,supportLIS,LISct);
  nms = CreateFuzzySet1("No_modify_speed","Triangular1",
                        variations,supportNMS,NMSct);
  lds = CreateFuzzySet1("Lightly_decrease_speed","Triangular1",
                        variations,supportLDS,LDSct);
  mds = CreateFuzzySet1("Moderatly_decrease_speed","Triangular1",
                        variations,supportMDS,MDSct);
  hds = CreateFuzzySet1("Highly_decrease_speed","Triangular1",
                        variations,supportHDS,HDSct);
}


struct SpeedVariation *CreateNullSpeedVariation()
{
   struct SpeedVariation *variations;

   variations = (struct SpeedVariation *)malloc(sizeof(struct SpeedVariation));
   variations->LeftVariation=CreateEmptyFuzzySet1();
   variations->RightVariation=CreateEmptyFuzzySet1();
   return(variations);
}


struct SpeedVariation FreeSpeedVariations(struct SpeedVariation *variations)
{
  FreeFuzzySet1(variations->LeftVariation);
  FreeFuzzySet1(variations->RightVariation);
  free(variations);
}


struct FuzzySet1 *ReadAntecedent1(char ant[4])
{
 struct FuzzySet1 *antecedent1;

  if (strncmp(ant,"nop",3)==0)
          antecedent1 = nop;
  else if (strncmp(ant,"olp",3)==0)
          antecedent1 = olp;
  else if (strncmp(ant,"omp",3)==0)
          antecedent1 = omp;
```

```
  else if (strncmp(ant,"ohp",3)==0)
          antecedent1 = ohp;
  else /*if (strncmp(ant,"col",3)==0)*/
          antecedent1 = collision;
  return(antecedent1);
}


struct FuzzySet1 *ReadAntecedent2(char ant[6])
{
 struct FuzzySet1 *antecedent2;

  if (strncmp(ant,"VS",2)==0)
          antecedent2 = vs;
  else if (strncmp(ant,"S",1)==0)
          antecedent2 = s;
  else if (strncmp(ant,"Nmod",4)==0)
          antecedent2 = nmod;
  else if (strncmp(ant,"Nulla",5)==0)
          antecedent2 = nulla;
  else if (strncmp(ant,"Pmod",4)==0)
          antecedent2 = pmod;
  else if (strncmp(ant,"F",1)==0)
          antecedent2 = f;
  else /* if (strncmp(ant,"VF",2)==0)*/
          antecedent2 = vf;

  return(antecedent2);
}


struct FuzzySet1 *ReadConsequent(char cons[4])
{
  struct FuzzySet1 *consequent;

  if (strncmp(cons,"his",3)==0)
          consequent = his;
  else if (strncmp(cons,"mis",3)==0)
          consequent = mis;
  else if (strncmp(cons,"lis",3)==0)
          consequent = lis;
  else if (strncmp(cons,"nms",3)==0)
          consequent = nms;
  else if (strncmp(cons,"lds",3)==0)
          consequent = lds;
  else if (strncmp(cons,"mds",3)==0)
          consequent = mds;
  else /*if (strncmp(cons,"hds",3)==0)*/
          consequent = hds;

  return(consequent);

}


struct RuleComponent *CreateEmptyRuleComponent()
{
   struct RuleComponent *component;
```

```
  component = (struct RuleComponent *)malloc(sizeof(struct RuleComponent));
  component->Antecedent1 = NULL;
  component->Antecedent2 = NULL;
  component->Consequent  = NULL;
  component->Next        = NULL;

  return(component);
}


struct RuleComponent *CreateRuleComponent(struct FuzzySet1 *ant1,
                                          struct FuzzySet1 *ant2,
                                          struct FuzzySet1 *conseq)
{
  struct RuleComponent *component;

  component = CreateEmptyRuleComponent();
  component->Antecedent1 = ant1;
  component->Antecedent2 = ant2;
  component->Consequent  = conseq;
  component->Next        = NULL;

  return(component);

}



void AddComponents (struct RuleComponent *component1,
                    struct RuleComponent *component2)
{
 component2->Next = component1->Next;
 component1->Next = component2;
}


void FreeComponents(struct RuleComponent *component)
{
 if(component)
 {
  FreeComponents(component->Next);
  FreeComponent(component);
 }

}


void FreeRuleBase(struct RuleBase *rule_base)
{
 FreeComponents(rule_base->Left);
 FreeComponents(rule_base->Right);
 FreeComponents(rule_base->Back);
 free(rule_base);
}
```

```
struct RuleBase *CreateEmptyRuleBase()
{
 struct RuleBase   *rule_base;
 short int         percept;

 rule_base = (struct RuleBase *)malloc(sizeof(struct RuleBase));
 strcpy(rule_base->Name,"new");
 rule_base->Left = NULL;
 rule_base->Right = NULL;
 rule_base->Back = NULL;
 return(rule_base);
}


struct RuleBase *ReadRuleBaseFromFile(FILE *file)
{
  struct RuleBase          *rule_base;
  char                     side[5], s1[2], ant1[4], s2[2],
                           ant2[6], s3[4], cons[4];
  struct FuzzySet1         *a1, *a2, *c;
  struct RuleComponent     *component1, *component2;
  short int                rule, percept;

  rule_base = CreateEmptyRuleBase();
  strcpy(rule_base->Name,"FUZZY CONTROL RULES");
  a1 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  a2 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  c = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1 ));
  component1 = CreateEmptyRuleComponent();
  component2 = CreateEmptyRuleComponent();
  for(percept=LEFT_SIDE;percept<=BACK_SIDE;++percept)
  {
    fscanf(file,"%4s%1s%3s%1s%5s%3s%3s\n",side,s1,ant1,s2,ant2,s3,cons);
    a1 = ReadAntecedent1(ant1);
    a2 = ReadAntecedent2(ant2);
    c = ReadConsequent(cons);
    component1 = CreateRuleComponent(a1,a2,c);
    switch (percept)
    {
          case 0: rule_base->Left = component1;
           break;
          case 1: rule_base->Right = component1;
                  break;
          case 2: rule_base->Back = component1;
           break;
    }
    for(rule=2;rule<=NUM_SIDE_RULES;++rule)
    {
     fscanf(file,"%4s%1s%3s%1s%5s%3s%3s\n",side,s1,ant1,s2,ant2,s3,cons);
     a1 = ReadAntecedent1(ant1);
     a2 = ReadAntecedent2(ant2);
     c = ReadConsequent(cons);
     component2 = CreateRuleComponent(a1, a2, c);
     AddComponents(component1, component2);
     component1 = component1->Next;
    }
  }
```

```
 return(rule_base);
}


struct FuzzySet1 *SelectLeftConsequent (struct RuleBase *rule_base,
                                        struct FuzzySet1 *a1,
                                        struct FuzzySet1 *a2,
                                        short int side)
{
 struct FuzzySet1      *found;
 struct RuleComponent  *search;

 search = (struct RuleComponent *)malloc(sizeof(struct RuleComponent));
 switch (side)
 {
        case 0: search = rule_base->Left;
                break;
        case 1: search = rule_base->Right;
                break;
        case 2: search = rule_base->Back;
                break;
 }
 found = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 found = NULL;
 while (search)
 {
   if (((a1->Center-search->Antecedent1->Center)==0.0)&&
      ((a2->Center-search->Antecedent2->Center)==0.0))
   {
        found = search->Consequent;
        search = NULL;
   }
   else
        search = search->Next;
 }
 return(found);
}


struct FuzzySet1 *SelectRightConsequent (struct RuleBase *rule_base,
                                         struct FuzzySet1 *a1,
                                         struct FuzzySet1 *a2,
                                         short int side)
{
 struct FuzzySet1      *found;
 struct RuleComponent  *search;
 short int              count;

 search = (struct RuleComponent *)malloc(sizeof(struct RuleComponent));
 switch (side)
 {
        case 0: search = rule_base->Left;
                break;
        case 1: search = rule_base->Right;
                break;
        case 2: search = rule_base->Back;
                break;
 }
```

```
 found = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 found = NULL;
 while (search)
 {
  for(count=1;count<=7;++count)
        search = search->Next;
  for(count=1;count<=7;++count)
  {
   if(((a1->Center-search->Antecedent1->Center)==0.0)&&
     ((a2->Center-search->Antecedent2->Center)==0.0))
   {
        found = search->Consequent;
        search = NULL;
        break;
   }
   else
    search = search->Next;
  }
 }
 return(found);
}


struct RuleComponent *FindLeftComponent(struct RuleBase *rule_base,
                                        struct FuzzySet1 *a1,
                                        struct FuzzySet1 *a2)
{
  struct RuleComponent *search, *found;
  struct FuzzySet1    *ant1,*ant2;
  float               center1, center2;

  ant1 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  ant2 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  center1 = a1->Center;
  center2 = a2->Center;
  search = rule_base->Left;
  found = NULL;
  while(search)
  {
        ant1 = search->Antecedent1;
        ant2 = search->Antecedent2;
        if(((ant1->Center-center1)==0.0) && ((ant2->Center-center2)==0.0))
    {
         found = search;
         search = NULL;
    }
    else
        search = search->Next;
  }

  return(found);
}


void WriteRuleBaseToFile(struct RuleBase *rule_base,FILE *file)
{
 struct RuleComponent  *component;
 short int                    percept, rule;
```

```
  fprintf(file,"# File       : EXAMPLES|FL_EX2|FUZZYZ|flc2.rules\n");
  fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel\n");
  fprintf(file,"# Description : FUZZY CONTROL RULES for Khepera mobile robot\n");
  fprintf(file,"# Date       :      \n");
  fprintf(file," ------------------------------------------------------------\n\n");
  fprintf(file,"\t\t** %s **\n\n",rule_base->Name);
  percept = LEFT_SIDE;
  while(percept<=BACK_SIDE)
  {
   fprintf(file,"  PERCEPTION %d:",percept);
   switch(percept)
   {
          case 0: fprintf(file,"LEFT side IR sensors.\n\n");
            component = rule_base->Left;
                  break;
          case 1: fprintf(file,"RIGHT side IR sensors.\n\n");
                  component = rule_base->Right;
                  break;
          case 2: fprintf(file,"BACK side IR sensors.\n\n");
                  component = rule_base->Back;
                  break;
   }
   for(rule=1;rule<NUM_SIDE_RULES;++rule)
   {
          fprintf(file," Rule %3d : a1: %s\n",rule, component->Antecedent1->Properties.Name);
          fprintf(file,"          a2: %s\n", component->Antecedent2->Properties.Name);
          fprintf(file,"          --> %s\n\n", component->Consequent->Properties.Name);
           component = component->Next;
   }
   percept++;
    fprintf(file,"\n\n\n");
  }
}


struct FuzzySet1 *DiscriminatePerceptions(struct FuzzySet1 *f_speed,
                                          struct FuzzySet1 *percept1,
                                          struct FuzzySet1 *percept2,
                                          struct FuzzySet2 *strengths,
                                          char Disjunction[MAX_TEXT],
                                          short int *side)
{

  struct FuzzySet1          *fuzzy_set;
  short int          speed, distance;
  float              c1, c2, max_firing1, max_firing2, max;

  fuzzy_set = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  c1 = percept1->Center;
  c2 = f_speed->Center;
  distance = ( short int ) c1;
  speed = (short int ) c2;
  max_firing1 = FindFiringStrength(distance,speed,strengths);
  c1 = percept2->Center;
  distance = (short int) c1;
  max_firing2 =  FindFiringStrength(distance,speed,strengths);
  if (strncmp(Disjunction,"Union",5)==0)
```

```
    TCoNormUnion(max_firing1,max_firing2,&max);
  else /*Algebraic Sum */
    TCoNormSum(max_firing1,max_firing2,&max);
  if((max - max_firing1)==0.0)
  {
   fuzzy_set = percept1;
   *side = 1;
  }
  else
  {
   fuzzy_set = percept2;
   *side = 2;
  }
  return(fuzzy_set);
}



struct FuzzySet1 *SelectMotorVariation(short int motor,
                                       struct PerceptionStates *perceptions,
                                       short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                       struct FuzzySet2 *strengths,
                                       struct PartitionOfSpeeds *partitions,
                                       struct RuleBase *rule_base,
                                      char Disjunction[MAX_TEXT])
{
  struct FuzzySet1      *speed, *temp, *selected, *variation;
  short int             num,side1,side2;
  struct PartitionMember  member;
  variation  = CreateEmptyFuzzySet1();
  speed     = CreateEmptyFuzzySet1();
  temp      = CreateEmptyFuzzySet1();
  selected   = CreateEmptyFuzzySet1();

  num = KheperaMotorFuzzyValues[motor];
  member = partitions->Members[num];
  speed = member.FuzzySet;
  temp = DiscriminatePerceptions(speed,perceptions->LeftState,
                                 perceptions->RightState,strengths,
                                 Disjunction,&side1);
  if ((side1-1)==0)
    side1 = LEFT_SIDE;
  else
    side1 = RIGHT_SIDE;
  selected = DiscriminatePerceptions(speed,temp,perceptions->BackState, strengths,Disjunction,&side2);
  if ((side2-1)==0)
        side2 = side1;
  else
        side2 = BACK_SIDE;

  switch (motor)
  {
   case RIGHT:
     variation = SelectRightConsequent(rule_base,selected, speed,side2);
     break;
```

```
    case LEFT:
          variation = SelectLeftConsequent(rule_base,selected,speed,side2);
          break;
   }
 return(variation);
}



struct SpeedVariation *FuzzyImplication(struct PerceptionStates *perceptions,
                                        short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                        struct RuleBase *rule_base,
                                        struct FuzzySet2 *strengths,
                                        struct PartitionOfSpeeds *partition,
                                        char Implication[MAX_TEXT])
{
 struct SpeedVariation *variations;

 variations = (struct SpeedVariation *)malloc(sizeof(struct SpeedVariation));
 if (strncmp(Implication,"Mamdami",7)==0)
 {
  variations->RightVariation = SelectMotorVariation(RIGHT,perceptions,
                                                    KheperaMotorFuzzyValues,strengths,
                                                   partition,rule_base,"Union");
  variations->LeftVariation = SelectMotorVariation(LEFT,perceptions,
                                                  KheperaMotorFuzzyValues,strengths,
                                                  partition,rule_base,"Union");

 }
 else /* Larse fuzzy implication */
 {
  variations->RightVariation = SelectMotorVariation(RIGHT,perceptions,
                                                    KheperaMotorFuzzyValues,strengths,
                                                   partition,rule_base,"Algebraic Sum");
  variations->LeftVariation = SelectMotorVariation(LEFT,perceptions,
                                                  KheperaMotorFuzzyValues,strengths,
                                                  partition,rule_base,"Algebraic Sum");
 }
 return(variations);
}



float *ConjunctionAlphaVariation(float alpha, struct FuzzySet1 *variation, char Implication[MAX_TEXT])
{
 short int     U0,U;
 float         mf, *temp,*conjunction;
 struct Value1   *value;

 value = (struct Value1 *)malloc(sizeof(struct Value1));
 U0 = variation->Support[0];
 value = variation->Values;
 *conjunction = alpha;

 for(U=U0;U<=variation->Support[1];++U)
 {
  mf=value->MembershipValue;
  if(strncmp(Implication,"Mamdami",7)==0)
```

```
    {
      TNormIntersection(mf,alpha,temp);
      TNormIntersection(*temp,*conjunction,conjunction);
    }
    else /* Larse fuzzy implication */
    {
      TNormProduct(mf,alpha,temp);
      TNormProduct(*temp,*conjunction,conjunction);
    }
       value = value->Next;
  }
 return(conjunction);
}



float InferControlActionMembership(float alpha,struct FuzzySet1 *variation, char Implication[MAX_TEXT])

{
 float        inference,temp, mf;
 struct Value1  *value;

 value  = (struct Value1 *)malloc(sizeof(struct Value1));
 value = FindValue1(variation,variation->Support[0]);
 inference = alpha;
 while(value)
 {
   mf = value->MembershipValue;
   if(strncmp(Implication,"Mamdami",7)==0)
   {
      TNormIntersection(alpha,mf,&temp);
      TNormIntersection(temp,inference,&inference);

   }
   else /* Larse fuzzy implication */
   {
      TNormProduct(alpha,mf,&temp);
      TNormProduct(temp,inference,&inference);
   }
   value = value->Next;
 }
 return(inference);
}


float InferControlAction(float alpha,struct FuzzySet1 *variation, char Implication[MAX_TEXT])
{
 float        inference,center;
 struct Value1  *value;
 short int      speed;

 value  = (struct Value1 *)malloc(sizeof(struct Value1));
 center = variation->Center;
 speed  = ( short int ) center;
 value = FindValue1(variation,speed);
 inference = value->MembershipValue;
 if(strncmp(Implication,"Mamdami",7)==0)
```

```
    TNormIntersection(alpha,inference,&inference);
  else /* Larse fuzzy implication */
    TNormProduct(alpha,inference,&inference);

  return(inference);
}


void CorrectMotorValues(struct Robot *robot)
{
  short int motor;

  for (motor=0;motor<=1;++motor)
  {
    if (robot->Motor[motor].Value > 10)
       robot->Motor[motor].Value = 10;
    if (robot->Motor[motor].Value < -10)
       robot->Motor[motor].Value = -10;
  }
}
```

INFERENCE.C

```
#include "../SRC/include.h"
#include "../SRC/types.h"
#include "fuzzysets.h"
#include "flc.h"

#define NUM_PERCEP_STATES      5
#define NUM_VARIATIONS         7
#define MIN_VARIATION_SPEED   -40
#define MAX_VARIATION_SPEED    40
#define LEFT_SIDE              0
#define RIGHT_SIDE             1
#define BACK_SIDE              2
#define NUM_SIDE_RULES        70

#define FreeComponent(x)    free(x);

struct Robot *robot;


short int supportHIS[2] =   {HISmin, HISmax},
          supportMIS[2] =  {MISmin, MISmax},
          supportLIS[2] =   {LISmin, LISmax},
          supportNMS[2] = {NMSmin, NMSmax},
          supportLDS[2] =  {LDSmax, LDSmin},
          supportMDS[2] = {MDSmax, MDSmin},
          supportHDS[2] =  {HDSmax, HDSmin};


short int distances[2] = {MIN_DIST, MAX_DIST},
```

```
        variations[2] = {MIN_VARIATION_SPEED, MAX_VARIATION_SPEED};


extern struct FuzzySet1 *vpb, *pb, *pm, *ps, *ze, *ns, *nm, *nb, *vnb;
extern struct FuzzySet1 *vf, *f, *pmod, *nulla, *nmod, *s, *vs;


/* States of perception */

struct FuzzySet1  *collision,
                  *ohp,   /* obstacle highly perceived   */
                  *omp,   /*   "    moderatly perceived  */
                  *olp,   /*   "    lightly perceived    */
                  *nop;   /*   "    no perceived         */



/* Speed variations */

struct FuzzySet1 *his,          /* highly increase speed       */
                 *mis,          /* moderatly increase speed    */
                 *lis,          /* lightly increase speed      */
                 *nms,          /* no modify speed             */
                 *lds,          /* lightly decrease speed      */
                 *mds,          /* moderatly increase speed    */
                 *hds;          /* highly increase speed       */

void CreateFuzzySetsOfPerception()
{
 nop      = Union1(vpb,pb,"No Obstacle Perceived",distances);
 nop->Support[0] = 0;
 olp      = Union1(pm,ps, "Obstacle Lightly Perceived",distances);
 omp      = Union1(ze,ns,"Obstacle Moderatly Perceived",distances);
 ohp      = Union1(nm,nb,"Obstacle Lightly Perceived",distances);
 collision = Union1(vnb,vnb,"Collision",distances);
 collision->Support[1] = 1023;
}


short int CountEqualFuzzyDistances(short int s1, short int s2,
                                   short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1],
                                   short int partition_member_1,
                                   short int partition_member_2)
{
 short int number, sensor;

 number = 0;
 for(sensor = s1; sensor <= s2; ++sensor)
   if ((KheperaSensorFuzzyValues[sensor] == partition_member_1) ||
      (KheperaSensorFuzzyValues[sensor] == partition_member_2))
     number++;
 return(number);
}


struct FuzzySet1 *CreateSidePerceptionState(short int s1, short int s2,
                                            short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
```

```
  struct FuzzySet1 *perception;
  short int number;

  perception = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,0,1);
  if(number >= 1)
    perception = nop;
  number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,2,3);
  if(number >= 1)
    perception = olp;
  number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,4,5);
  if(number >= 1)
    perception = omp;
  number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,6,7);
  if(number >= 1)
    perception = ohp;
  number = CountEqualFuzzyDistances(s1,s2,KheperaSensorFuzzyValues,8,8);
  if(number >= 1)
    perception = collision;

  return(perception);
}



struct PerceptionStates *CreateFuzzyPerceptionStates(short int
                                                     KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
  struct PerceptionStates *perceptions;

  perceptions = (struct PerceptionStates *)malloc(sizeof(struct PerceptionStates));
  perceptions->LeftState = CreateSidePerceptionState(0,2,KheperaSensorFuzzyValues);
  perceptions->RightState = CreateSidePerceptionState(3,5,KheperaSensorFuzzyValues);
  perceptions->BackState = CreateSidePerceptionState(6,7,KheperaSensorFuzzyValues);

  return(perceptions);
}


void WriteFuzzyPerceptionStatesToFile(struct PerceptionStates *perceptions,
                                      struct Robot *robot,
                                      short int step,
                                      FILE *file)
{
  short int sensor;

  fprintf(file,"# File       : FUZZY|fuzzy.perceptions\n");
  fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Description : Current FUZZY PERCEPTION states of Khepera\n");
  fprintf(file,"# Date       :      \n");
  fprintf(file," ----------------------------------------------------------\n");
  fprintf(file,"  *STEP: %d\n",step);
  fprintf(file," ----------------------------------------------------------\n\n\n");
  fprintf(file,"    - IR Sensors Crisp Distance Values:\n");
  for(sensor = 0; sensor < NUM_IRSENSORS; ++sensor)
     if (sensor==0 || sensor==1 || sensor==2)
       fprintf(file,"         IR Sensor %d (LEFT) : %d\n",sensor,robot->IRSensor[sensor].DistanceValue);
     else if (sensor==3 || sensor==4 || sensor==5)
```

```
      fprintf(file,"        IR Sensor %d (RIGHT): %d\n",sensor,robot->IRSensor[sensor].DistanceValue);
    else /* (sensor==6 || sensor==7) */
      fprintf(file,"        IR Sensor %d (BACK) : %d\n",sensor,robot->IRSensor[sensor].DistanceValue);

  fprintf(file,"    - LEFT IR Sensors interpretation : %s\n",perceptions->LeftState->Properties.Name);
  fprintf(file,"    - RIGHT IR Sensors interpretation: %s\n",perceptions->RightState->Properties.Name);
  fprintf(file,"    - BACK IR Sensors interpretation : %s\n",perceptions->BackState->Properties.Name);
}


void CreateFuzzySetsOfSpeedVariation()
{
  his = CreateFuzzySet1("Highly_increase_speed","Triangular1",
                        variations,supportHIS,HISct);
  mis = CreateFuzzySet1("Moderatly_increase_speed","Triangular1",
                        variations,supportMIS,MISct);
  lis = CreateFuzzySet1("Lightly_increase_speed","Triangular1",
                        variations,supportLIS,LISct);
  nms = CreateFuzzySet1("No_modify_speed","Triangular1",
                        variations,supportNMS,NMSct);
  lds = CreateFuzzySet1("Lightly_decrease_speed","Triangular1",
                        variations,supportLDS,LDSct);
  mds = CreateFuzzySet1("Moderatly_decrease_speed","Triangular1",
                        variations,supportMDS,MDSct);
  hds = CreateFuzzySet1("Highly_decrease_speed","Triangular1",
                        variations,supportHDS,HDSct);
}


struct SpeedVariation *CreateNullSpeedVariation()
{
   struct SpeedVariation *variations;

   variations = (struct SpeedVariation *)malloc(sizeof(struct SpeedVariation));
   variations->LeftVariation=CreateEmptyFuzzySet1();
   variations->RightVariation=CreateEmptyFuzzySet1();

   return(variations);

}


struct SpeedVariation FreeSpeedVariations(struct SpeedVariation *variations)
{
  FreeFuzzySet1(variations->LeftVariation);
  FreeFuzzySet1(variations->RightVariation);
  free(variations);
}


struct FuzzySet1 *ReadAntecedent1(char ant[4])
{
 struct FuzzySet1 *antecedent1;

  if (strncmp(ant,"nop",3)==0)
          antecedent1 = nop;
  else if (strncmp(ant,"olp",3)==0)
          antecedent1 = olp;
```

```
  else if (strncmp(ant,"omp",3)==0)
          antecedent1 = omp;
  else if (strncmp(ant,"ohp",3)==0)
          antecedent1 = ohp;
  else /*if (strncmp(ant,"col",3)==0)*/
          antecedent1 = collision;

  return(antecedent1);

}


struct FuzzySet1 *ReadAntecedent2(char ant[6])
{
 struct FuzzySet1 *antecedent2;

  if (strncmp(ant,"VS",2)==0)
          antecedent2 = vs;
  else if (strncmp(ant,"S",1)==0)
          antecedent2 = s;
  else if (strncmp(ant,"Nmod",4)==0)
          antecedent2 = nmod;
  else if (strncmp(ant,"Nulla",5)==0)
          antecedent2 = nulla;
  else if (strncmp(ant,"Pmod",4)==0)
          antecedent2 = pmod;
  else if (strncmp(ant,"F",1)==0)
          antecedent2 = f;
  else /* if (strncmp(ant,"VF",2)==0)*/
          antecedent2 = vf;

  return(antecedent2);

}


struct FuzzySet1 *ReadConsequent(char cons[4])
{
  struct FuzzySet1 *consequent;

  if (strncmp(cons,"his",3)==0)
          consequent = his;
  else if (strncmp(cons,"mis",3)==0)
          consequent = mis;
  else if (strncmp(cons,"lis",3)==0)
          consequent = lis;
  else if (strncmp(cons,"nms",3)==0)
          consequent = nms;
  else if (strncmp(cons,"lds",3)==0)
          consequent = lds;
  else if (strncmp(cons,"mds",3)==0)
          consequent = mds;
  else /*if (strncmp(cons,"hds",3)==0)*/
          consequent = hds;

  return(consequent);
}
```

```
struct RuleComponent *CreateEmptyRuleComponent()
{
  struct RuleComponent *component;

  component = (struct RuleComponent *)malloc(sizeof(struct RuleComponent));
  component->Antecedent1 = NULL;
  component->Antecedent2 = NULL;
  component->Consequent  = NULL;
  component->Next        = NULL;

  return(component);
}


struct RuleComponent *CreateRuleComponent(struct FuzzySet1 *ant1,
                                          struct FuzzySet1 *ant2,
                                          struct FuzzySet1 *conseq)
{
  struct RuleComponent *component;

  component = CreateEmptyRuleComponent();
  component->Antecedent1 = ant1;
  component->Antecedent2 = ant2;
  component->Consequent  = conseq;
  component->Next        = NULL;

  return(component);

}



void AddComponents (struct RuleComponent *component1,
                    struct RuleComponent *component2)
{
 component2->Next = component1->Next;
 component1->Next = component2;
}


void FreeComponents(struct RuleComponent *component)
{
 if(component)
 {
  FreeComponents(component->Next);
  FreeComponent(component);
 }

}


void FreeRuleBase(struct RuleBase *rule_base)
{
 FreeComponents(rule_base->Left);
 FreeComponents(rule_base->Right);
 FreeComponents(rule_base->Back);
 free(rule_base);
```

```
}

struct RuleBase *CreateEmptyRuleBase()
{
  struct RuleBase   *rule_base;
  short int         percept;

  rule_base = (struct RuleBase *)malloc(sizeof(struct RuleBase));
  strcpy(rule_base->Name,"new");
  rule_base->Left = NULL;
  rule_base->Right = NULL;
  rule_base->Back = NULL;
  return(rule_base);
}


struct RuleBase *ReadRuleBaseFromFile(FILE *file)
{
   struct RuleBase          *rule_base;
   char                     side[5], s1[2], ant1[4], s2[2],
                            ant2[6], s3[4], cons[4];
   struct FuzzySet1         *a1, *a2, *c;
   struct RuleComponent     *component1, *component2;
   short int                rule, percept;

   rule_base = CreateEmptyRuleBase();
   strcpy(rule_base->Name,"FUZZY CONTROL RULES");
   a1 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
   a2 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
   c = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1 ));
   component1 = CreateEmptyRuleComponent();
   component2 = CreateEmptyRuleComponent();
   for(percept=LEFT_SIDE;percept<=BACK_SIDE;++percept)
   {
     fscanf(file,"%4s%1s%3s%1s%5s%3s%3s\n",side,s1,ant1,s2,ant2,s3,cons);
     a1 = ReadAntecedent1(ant1);
     a2 = ReadAntecedent2(ant2);
     c = ReadConsequent(cons);
     component1 = CreateRuleComponent(a1,a2,c);
     switch (percept)
     {
          case 0: rule_base->Left = component1;
            break;
          case 1: rule_base->Right = component1;
                  break;
          case 2: rule_base->Back = component1;
            break;
     }
     for(rule=2;rule<=NUM_SIDE_RULES;++rule)
     {
       fscanf(file,"%4s%1s%3s%1s%5s%3s%3s\n",side,s1,ant1,s2,ant2,s3,cons);
       a1 = ReadAntecedent1(ant1);
       a2 = ReadAntecedent2(ant2);
       c = ReadConsequent(cons);
       component2 = CreateRuleComponent(a1, a2, c);
       AddComponents(component1, component2);
       component1 = component1->Next;
```

```
    }
   }
  return(rule_base);
 }


struct FuzzySet1 *SelectLeftConsequent (struct RuleBase *rule_base,
                                        struct FuzzySet1 *a1,
                                        struct FuzzySet1 *a2,
                                        short int side)
{
 struct FuzzySet1      *found;
 struct RuleComponent  *search;

 search = (struct RuleComponent *)malloc(sizeof(struct RuleComponent));
 switch (side)
 {
        case 0: search = rule_base->Left;
                break;
        case 1: search = rule_base->Right;
                break;
        case 2: search = rule_base->Back;
                break;
 }
 found = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 found = NULL;
 while (search)
 {
   if (((a1->Center-search->Antecedent1->Center)==0.0)&&
      ((a2->Center-search->Antecedent2->Center)==0.0))
   {
        found = search->Consequent;
        search = NULL;
   }
   else
        search = search->Next;
 }
 return(found);
}


struct FuzzySet1 *SelectRightConsequent (struct RuleBase *rule_base,
                                         struct FuzzySet1 *a1,
                                         struct FuzzySet1 *a2,
                                         short int side)
{
 struct FuzzySet1      *found;
 struct RuleComponent  *search;
 short int              count;

 search = (struct RuleComponent *)malloc(sizeof(struct RuleComponent));
 switch (side)
 {
        case 0: search = rule_base->Left;
                break;
        case 1: search = rule_base->Right;
                break;
```

```
        case 2: search = rule_base->Back;
                break;
  }
  found = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  found = NULL;
  while (search)
  {
   for(count=1;count<=7;++count)
        search = search->Next;
   for(count=1;count<=7;++count)
   {
    if(((a1->Center-search->Antecedent1->Center)==0.0)&&
      ((a2->Center-search->Antecedent2->Center)==0.0))
    {
        found = search->Consequent;
        search = NULL;
        break;
    }
    else
     search = search->Next;
   }
  }
  return(found);
}


struct RuleComponent *FindLeftComponent(struct RuleBase *rule_base,
                                        struct FuzzySet1 *a1,
                                        struct FuzzySet1 *a2)
{
  struct RuleComponent *search, *found;
  struct FuzzySet1    *ant1,*ant2;
  float               center1, center2;

  ant1 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  ant2 = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  center1 = a1->Center;
  center2 = a2->Center;
  search = rule_base->Left;
  found = NULL;
  while(search)
  {
        ant1 = search->Antecedent1;
        ant2 = search->Antecedent2;
        if(((ant1->Center-center1)==0.0)&&((ant2->Center-center2)==0.0))
        {
         found = search;
         search = NULL;
        }
        else
         search = search->Next;
  }

  return(found);
}


void WriteRuleBaseToFile(struct RuleBase *rule_base,FILE *file)
```

```
{
 struct RuleComponent  *component;
 short int                    percept, rule;

 fprintf(file,"# File       : EXAMPLES|FL_EX2|FUZZYZ|flc2.rules\n");
 fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel\n");
 fprintf(file,"# Description : FUZZY CONTROL RULES for Khepera mobile robot\n");
 fprintf(file,"# Date       :      \n");
 fprintf(file," ------------------------------------------------------------\n\n");
 fprintf(file,"\t\t** %s **\n\n",rule_base->Name);
 percept = LEFT_SIDE;
 while(percept<=BACK_SIDE)
 {
  fprintf(file,"  PERCEPTION %d:",percept);
  switch(percept)
  {
        case 0: fprintf(file,"LEFT side IR sensors.\n\n");
         component = rule_base->Left;
                 break;
        case 1: fprintf(file,"RIGHT side IR sensors.\n\n");
                 component = rule_base->Right;
                 break;
        case 2: fprintf(file,"BACK side IR sensors.\n\n");
                 component = rule_base->Back;
                 break;
  }
  for(rule=1;rule<NUM_SIDE_RULES;++rule)
  {
     fprintf(file," Rule %3d : a1: %s\n",rule,
                      component->Antecedent1->Properties.Name);
     fprintf(file,"          a2: %s\n",
                      component->Antecedent2->Properties.Name);
     fprintf(file,"          --> %s\n\n",
                      component->Consequent->Properties.Name);
     component = component->Next;
  }
   percept++;
   fprintf(file,"\n\n\n");
 }

}


struct FuzzySet2 *CreateFiringStrengths (char Conjunction[MAX_TEXT],
                                         char m_function[MAX_TEXT],
                                         struct PartitionOfDistances *part_dists,
                                         struct PartitionOfSpeeds *part_speeds)
{

 struct FuzzySet2       *strengths;
 struct Value2          *strength;
 short int              dist_partition, speed_partition;
 short int                    univU[2], univV[2];
 struct PartitionMember   dist_member, speed_member;
 short int              dist=0, speed=0, d0,d1,s0,s1;
 struct Value1          *dist_value, *speed_value;
 float                        dist_mf, speed_mf,strength_value=0.0;
```

```
  univU[0] = part_dists->Interval[0];
  univU[1] = part_dists->Interval[1];
  univV[0] = part_speeds->Interval[0];
  univV[1] = part_speeds->Interval[1];
  strengths = CreateNullCartesianProduct11(univU,univV);
  strcpy(strengths->Properties.Name,"FIRING STRENGTHS");
  strcpy(strengths->Properties.MembershipFunctionType, m_function);
  strengths->CenterX = (1024/2);
  strengths->CenterY = 0.0;
  strengths->Properties.Cardinality = 1024*21;
  strength = (struct Value2 *)malloc(sizeof(struct Value2));
  strength = strengths->Values;

  for (dist_partition = 0; dist_partition<NUM_DIST_SETS; ++dist_partition)
  {
   dist_member = part_dists->Members[dist_partition];
   d0 = part_dists->Members[dist_partition].IntervalMember[0];
   d1 = part_dists->Members[dist_partition].IntervalMember[1];
   for(dist = d0;dist<= d1;++dist)
   {
    dist_value = FindValue1(dist_member.FuzzySet,dist);
    dist_mf = dist_value->MembershipValue;
     for (speed_partition = 0; speed_partition<NUM_SPEED_SETS; ++speed_partition)
     {
      speed_member = part_speeds->Members[speed_partition];
      s0 = speed_member.IntervalMember[0];
          s1 = speed_member.IntervalMember[1];
          for(speed = s0;speed <= s1;++speed)
     {
      speed_value = FindValue1(speed_member.FuzzySet,speed);
           speed_mf = speed_value->MembershipValue;
      if (strncmp(Conjunction,"Intersection",12)==0)
            TNormIntersection(dist_mf,speed_mf,&strength_value);
      else /* Algebraic product */
            TNormProduct(dist_mf,speed_mf,&strength_value);
          strength->CrispX = dist;
          strength->CrispY = speed;
      strength->MembershipValue = strength_value;
      strength = strength->Next;
     }
    }
   }
  }

  return(strengths);

}


float FindFiringStrength(short int distance, short int speed,
                         struct FuzzySet2 *strengths)
{
  float firing;
  struct Value2 *value;

  value = (struct Value2 *)malloc(sizeof(struct Value2));
  value = FindValue2(strengths,distance,speed);
  firing = value->MembershipValue;
```

```
  return(firing);
}


float DisjunctionOfFiringStrengths(struct FuzzySet2 *strengths, char Disjunction[MAX_TEXT])
{
 /* calculating the fuzzy interpretation of the
    "also" operator between the fuzzy control
    rules of each sub_side component */

 float     alpha, disjunction=0.0;
 short int  i, j;

 disjunction = FindFiringStrength(0,-10,strengths);
 for(i=-9;i<=10;++i)
  for(j=100;j<=1000;j+=100)
  {
    alpha = FindFiringStrength(j,i,strengths);
    if (strncmp(Disjunction,"Union",5)==0)
      TCoNormUnion(alpha,disjunction,&disjunction);
    else /*Algebraic Sum */
      TCoNormSum(alpha,disjunction,&disjunction);
  }
 return(disjunction);
}


void WriteFiringStregthsToFile(struct FuzzySet2 *strengths, FILE *file)
{
  struct Value2   *value;

  fprintf(file,"#FILE: FIRING_STRENGTHS.\n");
  fprintf(file,"# Author: Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date  :     \n");
  fprintf(file,"-----------------------------------------------------\n\n\n");
  fprintf(file," -Name: %s\n",strengths->Properties.Name);
  fprintf(file," -Membership Function Type: %s\n",
                    strengths->Properties.MembershipFunctionType);
  fprintf(file," -Cardinality: %d\n",strengths->Properties.Cardinality);
  fprintf(file," -XSupport: [%d,%d]\n",strengths->SupportX[0],strengths->SupportX[1]);
  fprintf(file," -YSupport: [%d,%d]\n",strengths->SupportY[0],strengths->SupportY[1]);
  fprintf(file," -XCenter: %f\n",strengths->CenterX);
  fprintf(file," -YCenter: %f\n",strengths->CenterY);
  fprintf(file," -Values:\n\n");
  value =  strengths->Values;
  while((value) && (value->CrispX<=strengths->SupportX[1]) &&
                   (value->CrispY<=strengths->SupportY[1]))
  {
    fprintf(file,"\t crisp distance = %3d  ,",value->CrispX);
    fprintf(file,"crisp speed = %2d -->",value->CrispY);
    fprintf(file,"firing strength = %f\n",value->MembershipValue);
    value = value->Next;
  }

}
```

```
struct FuzzySet1 *DiscriminatePerceptions(struct FuzzySet1 *f_speed,
                                          struct FuzzySet1 *percept1,
                                          struct FuzzySet1 *percept2,
                                          struct FuzzySet2 *strengths,
                                          char Disjunction[MAX_TEXT],
                                          short int *side)
{
  struct FuzzySet1          *fuzzy_set;
  short int                 speed, distance;
  float                     c1, c2, max_firing1, max_firing2, max;

  fuzzy_set = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  c1 = percept1->Center;
  c2 = f_speed->Center;
  distance = ( short int ) c1;
  speed = (short int ) c2;
  max_firing1 = FindFiringStrength(distance,speed,strengths);
  c1 = percept2->Center;
  distance = (short int) c1;
  max_firing2 =  FindFiringStrength(distance,speed,strengths);
  if (strncmp(Disjunction,"Union",5)==0)
     TCoNormUnion(max_firing1,max_firing2,&max);
  else /*Algebraic Sum */
     TCoNormSum(max_firing1,max_firing2,&max);
  if((max - max_firing1)==0.0)
  {
   fuzzy_set = percept1;
   *side = 1;
  }
  else
  {
   fuzzy_set = percept2;
   *side = 2;
  }
  return(fuzzy_set);
}


struct FuzzySet1 *SelectMotorVariation(short int motor,
                                       struct PerceptionStates *perceptions,
                                       short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                       struct FuzzySet2 *strengths,
                                       struct PartitionOfSpeeds *partitions,
                                       struct RuleBase *rule_base,
                                       char Disjunction[MAX_TEXT])
{
 /* final implemntation of "also"
   operator for each
   perception_side rule base */

  struct FuzzySet1       *speed, *temp, *selected, *variation;
  short int              num,side1,side2;
  struct PartitionMember   member;

  variation  = CreateEmptyFuzzySet1();
  speed      = CreateEmptyFuzzySet1();
  temp       = CreateEmptyFuzzySet1();
```

```
  selected   = CreateEmptyFuzzySet1();

  num = KheperaMotorFuzzyValues[motor];
  member = partitions->Members[num];
  speed = member.FuzzySet;
  temp = DiscriminatePerceptions(speed,perceptions->LeftState,
                                 perceptions->RightState,strengths,
                                 Disjunction,&side1);
  if ((side1-1)==0)
     side1 = LEFT_SIDE;
  else
     side1 = RIGHT_SIDE;
  selected = DiscriminatePerceptions(speed,temp,perceptions->BackState,strengths,Disjunction,&side2);
  if ((side2-1)==0)
        side2 = side1;
  else
        side2 = BACK_SIDE;

  switch (motor)
  {
   case RIGHT:
     variation = SelectRightConsequent(rule_base,selected,
                                       speed,side2);
     break;
   case LEFT:
        variation = SelectLeftConsequent(rule_base,selected,
                                         speed,side2);
        break;
  }
 return(variation);
}


struct SpeedVariation *FuzzyImplication(struct PerceptionStates *perceptions,
                                        short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                        struct RuleBase *rule_base,
                                        struct FuzzySet2 *strengths,
                                        struct PartitionOfSpeeds *partition,
                                        char Implication[MAX_TEXT])
{
 struct SpeedVariation *variations;

 variations = (struct SpeedVariation *)malloc(sizeof(struct SpeedVariation));
 if (strncmp(Implication,"Mamdami",7)==0)
 {
  variations->RightVariation = SelectMotorVariation(RIGHT,perceptions,
                                                    KheperaMotorFuzzyValues,strengths,
                                                   partition,rule_base,"Union");
  variations->LeftVariation = SelectMotorVariation(LEFT,perceptions,
                                                   KheperaMotorFuzzyValues,strengths,
                                                   partition,rule_base,"Union");

 }
 else /* Larse fuzzy implication */
 {
  variations->RightVariation = SelectMotorVariation(RIGHT,perceptions,
                                                    KheperaMotorFuzzyValues,strengths,
                                                   partition,rule_base,"Algebraic Sum");
```

```
   variations->LeftVariation = SelectMotorVariation(LEFT,perceptions,
                                                    KheperaMotorFuzzyValues,strengths,
                                                    partition,rule_base,"Algebraic Sum");
  }
 return(variations);
}



float *ConjunctionAlphaVariation(float alpha, struct FuzzySet1 *variation,
                                 char Implication[MAX_TEXT])
{
 short int     U0,U;
 float         mf, *temp,*conjunction;
 struct Value1   *value;

 value = (struct Value1 *)malloc(sizeof(struct Value1));
 U0 = variation->Support[0];
 value = variation->Values;
 *conjunction = alpha;
 for(U=U0;U<=variation->Support[1];++U)
 {
  mf=value->MembershipValue;
  if(strncmp(Implication,"Mamdami",7)==0)
  {
   TNormIntersection(mf,alpha,temp);
   TNormIntersection(*temp,*conjunction,conjunction);
  }
  else /* Larse fuzzy implication */
  {
   TNormProduct(mf,alpha,temp);
   TNormProduct(*temp,*conjunction,conjunction);
  }
    value = value->Next;
 }

 return(conjunction);
}


float InferControlActionMembership(float alpha,struct FuzzySet1 *variation,
                                   char Implication[MAX_TEXT])

{
 float         inference,temp, mf;
 struct Value1  *value;

 value  = (struct Value1 *)malloc(sizeof(struct Value1));
 value = FindValue1(variation,variation->Support[0]);
 inference = alpha;
 while(value)
 {
  mf = value->MembershipValue;
  if(strncmp(Implication,"Mamdami",7)==0)
  {
     TNormIntersection(alpha,mf,&temp);
```

```
      TNormIntersection(temp,inference,&inference);

    }
    else /* Larse fuzzy implication */
    {
      TNormProduct(alpha,mf,&temp);
      TNormProduct(temp,inference,&inference);
    }
    value = value->Next;
  }

  return(inference);
}


float InferControlAction(float alpha,struct FuzzySet1 *variation,
                         char Implication[MAX_TEXT])
{
  float        inference,center;
  struct Value1  *value;
  short int      speed;

  value  = (struct Value1 *)malloc(sizeof(struct Value1));
  center = variation->Center;
  speed  = ( short int ) center;
  value = FindValue1(variation,speed);
  inference = value->MembershipValue;
  if(strncmp(Implication,"Mamdami",7)==0)
    TNormIntersection(alpha,inference,&inference);
  else /* Larse fuzzy implication */
    TNormProduct(alpha,inference,&inference);

  return(inference);
}
```

INTERFACE.C

```
#include "../SRC/include.h"
#include "../SRC/types.h"
#include "flc.h"
#include "fuzzysets.h"


struct Robot *robot;


/* PRIMARY FUZZY SETS */

/* Primary Fuzzy Distances */
```

```
struct FuzzySet1 *vpb, *pb, *pm, *ps, *ze, *ns, *nm, *nb, *vnb;

/* Primary Fuzzy Speeds */

struct FuzzySet1 *vf, *f, *pmod, *nulla, *nmod, *s, *vs;

/* Supports of primary Fuzzy Sets */

short int univ_distances[2] = {MIN_DIST,MAX_DIST};
short int    supportVPB[2] = {VPBmin,VPBmax};
short int     supportPB[2] = {PBmin,PBmax};
short int     supportPM[2] = {PMmin,PMmax};
short int     supportPS[2] = {PSmin,PSmax};
short int     supportZE[2] = {ZEmin,ZEmax};
short int     supportNS[2] = {NSmin,NSmax};
short int     supportNM[2] = {NMmin,NMmax};
short int     supportNB[2] = {NBmin,NBmax};
short int    supportVNB[2] = {VNBmin,VNBmax};
short int  univ_speeds[2] = {MIN_SPEED,MAX_SPEED};
short int     supportVF[2] = {VFmin,VFmax};
short int      supportF[2] = {Fmin,Fmax};
short int   supportPMod[2] = {PModmin,PModmax};
short int   supportNULL[2] = {NULLmin,NULLmax};
short int   supportNMod[2] = {NModmax,NModmin};
short int      supportS[2] = {Smax,Smin};
short int     supportVS[2] = {VSmax,VSmin};


void CreatePrimaryFuzzyDistances()
{
  vpb = CreateFuzzySet1("VPB",
                        "Triangular1",univ_distances,supportVPB,VPBct);
  pb  = CreateFuzzySet1("PB",
                        "Triangular1",univ_distances,supportPB,PBct);
  pm  = CreateFuzzySet1("PM",
                      "Triangular1",univ_distances,supportPM,PMct);
  ps  = CreateFuzzySet1("PS",
                        "Triangular1",univ_distances,supportPS,PSct);
  ze  = CreateFuzzySet1("ZE",
                        "Triangular1",univ_distances,supportZE,ZEct);
  ns  = CreateFuzzySet1("NS",
                        "Triangular1",univ_distances,supportNS,NSct);
  nm  = CreateFuzzySet1("NM",
                        "Triangular1",univ_distances,supportNM,NMct);
  nb  = CreateFuzzySet1("NB",
                        "Triangular1",univ_distances,supportNB,NBct);
  vnb = CreateFuzzySet1("VNB",
                        "Triangular1",univ_distances,supportVNB,VNBct);
}


void CreatePrimaryFuzzySpeeds()
{
  vf   = CreateFuzzySet1("VF",
                         "Triangular1",univ_speeds,supportVF,VFct);
  f    = CreateFuzzySet1("F",
```

```
                                "Triangular1",univ_speeds,supportF,Fct);
  pmod  = CreateFuzzySet1("PMod",
                                "Triangular1",univ_speeds,supportPMod,PModct);
  nulla = CreateFuzzySet1("NULL",
                                "Triangular1",univ_speeds,supportNULL,NULLct);
  nmod  =  CreateFuzzySet1("NMod",
                                 "Triangular1",univ_speeds,supportNMod,NModct);
  s     = CreateFuzzySet1("S",
                                "Triangular1",univ_speeds,supportS,Sct);
  vs    = CreateFuzzySet1("VS",
                                "Triangular1",univ_speeds,supportVS,VSct);
}


short int LimitPartitionMember(struct FuzzySet1 *fs1,
                               struct FuzzySet1 *fs2)
/* fs1 and fs2 may be ordered */
{
 struct Value1 *search1, *search2;
 short int element, intersection_1, intersection_2;

 search1 = (struct Value1 *)malloc(sizeof(struct Value1));
 search2 = (struct Value1 *)malloc(sizeof(struct Value1));
 intersection_1=fs2->Support[0];
 intersection_2=fs1->Support[1];
 element = intersection_1;

 if ((abs(intersection_2) - intersection_1) > 0)
 {
   search1 = FindValue1(fs1,intersection_1);
   search2 = FindValue1(fs2,intersection_1);
   while (search1 && search2)
   {
    if ((search2->MembershipValue - search1->MembershipValue) > 0.0 )
    {
       element = (search1->CrispX) - 1;
       search1 = NULL;
       search2 = NULL;
    }
    else
    {
     search1 = search1->Next;      /* next value of the support of fs1 */
     search2 = search2->Next;
    }
   }
 }
 return(element);
}


struct PartitionOfDistances *CreateEmptyPartitionOfDistances()
{
 struct PartitionOfDistances *partition;
 short int num;

 partition = (struct PartitionOfDistances *)
```

```
            malloc(sizeof(struct PartitionOfDistances));
  strcpy(partition->Name,"new");
  partition->Interval[0] = 0;
  partition->Interval[1] = 0;
  for(num = 0;num < NUM_DIST_SETS;++num)
  {
   partition->Members[num].IntervalMember[0] = 0;
   partition->Members[num].IntervalMember[1] = 0;
   partition->Members[num].FuzzySet = (struct FuzzySet1 *)
                                        malloc(sizeof(struct FuzzySet1));
  }
  return(partition);
}


void FreeDistanceMembers(struct PartitionOfDistances *partition)
{
  short int num;

  for(num = 0;num < NUM_DIST_SETS - 1;++num)
   FreeFuzzySet1(partition->Members[num].FuzzySet);
}


struct PartitionOfDistances *CreatePartitionOfDistances(short int interval[2])
{
  struct PartitionOfDistances *partition;

  partition = CreateEmptyPartitionOfDistances();
  strcpy(partition->Name,"FUZZY PARTITIONS OF DISTANCE VALUES");
  partition->Interval[0] = interval[0];
  partition->Interval[1] = interval[1];     /* to be modified with previous*/
                           /* connection of fuzzy distances */

  partition->Members[0].FuzzySet        = vpb;
  partition->Members[0].IntervalMember[0] = interval[0];
  partition->Members[0].IntervalMember[1] = LimitPartitionMember(vpb,pb);

  partition->Members[1].FuzzySet        = pb;
  partition->Members[1].IntervalMember[0] = partition->Members[0].IntervalMember[1]+1;
  partition->Members[1].IntervalMember[1] = LimitPartitionMember(pb,pm);

  partition->Members[2].FuzzySet     = pm;
  partition->Members[2].IntervalMember[0] = partition->Members[1].IntervalMember[1]+1;
  partition->Members[2].IntervalMember[1] = LimitPartitionMember(pm,ps);

  partition->Members[3].FuzzySet     = ps;
  partition->Members[3].IntervalMember[0] = partition->Members[2].IntervalMember[1]+1;
  partition->Members[3].IntervalMember[1] = LimitPartitionMember(ps,ze);

  partition->Members[4].FuzzySet     = ze;
  partition->Members[4].IntervalMember[0] = partition->Members[3].IntervalMember[1]+1;
  partition->Members[4].IntervalMember[1] = LimitPartitionMember(ze,ns);

  partition->Members[5].FuzzySet     = ns;
  partition->Members[5].IntervalMember[0] = partition->Members[4].IntervalMember[1]+1;
  partition->Members[5].IntervalMember[1] = LimitPartitionMember(ns,nm);
```

```
  partition->Members[6].FuzzySet      = nm;
  partition->Members[6].IntervalMember[0] = partition->Members[5].IntervalMember[1]+1;
  partition->Members[6].IntervalMember[1] = LimitPartitionMember(nm,nb);

  partition->Members[7].FuzzySet      = nb;
  partition->Members[7].IntervalMember[0] =  partition->Members[6].IntervalMember[1]+1;
  partition->Members[7].IntervalMember[1] = LimitPartitionMember(nb,vnb);

  partition->Members[8].FuzzySet      = vnb;
  partition->Members[8].IntervalMember[0] = partition->Members[7].IntervalMember[1]+1;
  partition->Members[8].IntervalMember[1] = partition->Interval[1];

  return(partition);

}


struct PartitionOfSpeeds *CreateEmptyPartitionOfSpeeds()
{
  struct PartitionOfSpeeds *partition;
  short int num;

  partition = (struct PartitionOfSpeeds *)malloc(sizeof(struct PartitionOfSpeeds));
  strcpy(partition->Name,"new");
  partition->Interval[0] = 0;
  partition->Interval[1] = 0;
  for(num = 0;num < NUM_SPEED_SETS;++num)
  {
    partition->Members[num].IntervalMember[0] = 0;
    partition->Members[num].IntervalMember[1] = 0;
    partition->Members[num].FuzzySet = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
  }
  return(partition);
}


struct PartitionOfSpeeds *CreatePartitionOfSpeeds(short int interval[2])
{
  struct PartitionOfSpeeds *partition;

  partition = CreateEmptyPartitionOfSpeeds();
  strcpy(partition->Name,"FUZZY PARTITIONS OF SPEED VALUES");
  partition->Interval[0] = interval[0];
  partition->Interval[1] = interval[1];

  partition->Members[0].FuzzySet        = vs;
  partition->Members[0].IntervalMember[0] = interval[0];
  partition->Members[0].IntervalMember[1] = LimitPartitionMember(vs,s);

  partition->Members[1].FuzzySet        = s;
  partition->Members[1].IntervalMember[0] = partition->Members[0].IntervalMember[1]+1;
  partition->Members[1].IntervalMember[1] = LimitPartitionMember(s,nmod);

  partition->Members[2].FuzzySet      = nmod;
  partition->Members[2].IntervalMember[0] = partition->Members[1].IntervalMember[1]+1;
```

```
  partition->Members[2].IntervalMember[1] = LimitPartitionMember(nmod,nulla);

  partition->Members[3].FuzzySet      = nulla;
  partition->Members[3].IntervalMember[0] = partition->Members[2].IntervalMember[1]+1;
  partition->Members[3].IntervalMember[1] = LimitPartitionMember(nulla,pmod);

  partition->Members[4].FuzzySet      = pmod;
  partition->Members[4].IntervalMember[0] = partition->Members[3].IntervalMember[1]+1;
  partition->Members[4].IntervalMember[1] = LimitPartitionMember(pmod,f);

  partition->Members[5].FuzzySet      = f;
  partition->Members[5].IntervalMember[0] = partition->Members[4].IntervalMember[1]+1;
  partition->Members[5].IntervalMember[1] = LimitPartitionMember(f,vf);


  partition->Members[6].FuzzySet      = vf;
  partition->Members[6].IntervalMember[0] = partition->Members[5].IntervalMember[1]+1;
  partition->Members[6].IntervalMember[1] = interval[1];

  return(partition);
}



void WritePartitionOfDistancesToFile(struct PartitionOfDistances *partition,
                                     FILE *file)
{
  short int num;
  fprintf(file,"#File      :FUZZY|distancespartition\n");
  fprintf(file,"#Description: %s\n",partition->Name);
  fprintf(file,"# Author    : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date      :      \n");
  fprintf(file,"-------------------------------------------------\n\n\n");

  fprintf(file,"  - Interval of crisp distances: [%d,%d]\n",partition->Interval[0],
                                                            partition->Interval[1]);
  fprintf(file,"  - Members:\n\n");
  for(num = 0; num < NUM_DIST_SETS; ++num)
  {
    fprintf(file," \t-Number of fuzzy distance member : %d\n",num);
    fprintf(file," \t-Interval of crisp distances: [%d,%d]\n",
                                          partition->Members[num].IntervalMember[0],
                                          partition->Members[num].IntervalMember[1]);
    fprintf(file," \t-Fuzzy distance set member: %s\n\n",
                                                  partition->Members[num].FuzzySet);
    fprintf(file," \t\t-XSupport: [%d,%d]\n",
                                     partition->Members[num].FuzzySet->Support[0],
                                     partition->Members[num].FuzzySet->Support[1]);
    fprintf(file," \t\t-Center:   %f\n",
                                     partition->Members[num].FuzzySet->Center);
    fprintf(file," \t\t-Membership Function Type:  %s\n\n\n\n",
            partition->Members[num].FuzzySet->Properties.MembershipFunctionType);

  }

}
```

```
void WritePartitionOfSpeedsToFile(struct PartitionOfSpeeds *partition, FILE *file)
{
  short int num;
  fprintf(file,"# File      : FUZZY|speedspartition\n");
  fprintf(file,"# Description: %s\n",partition->Name);
  fprintf(file,"# Author    : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Date      :     \n");
  fprintf(file,"-------------------------------------------------\n\n\n");


  fprintf(file," - Interval of crisp speed values: [%d , %d]\n",partition->Interval[0],
                                                                partition->Interval[1]);
  fprintf(file," - Members:\n\n");
  for(num = 0; num < NUM_SPEED_SETS; ++num)
  {
    fprintf(file," \t-Number of fuzzy speed member: %d\n",num);
    fprintf(file," \t-Interval of crisp speed values: [%d,%d]\n", partition->Members[num].IntervalMember[0],
                  partition->Members[num].IntervalMember[1]);
    fprintf(file," \t-Fuzzy speed set member: %s\n\n",partition->Members[num].FuzzySet);
    fprintf(file," \t\t-XSupport: [%d,%d] \n",partition->Members[num].FuzzySet->Support[0],
                  partition->Members[num].FuzzySet->Support[1]);
    fprintf(file," \t\t-Center:  %f\n",partition->Members[num].FuzzySet->Center);
    fprintf(file," \t\t-Membership Function Type:  %s\n\n\n\n",
                  partition->Members[num].FuzzySet->Properties.MembershipFunctionType);
  }
}


void FreeSpeedMembers(struct PartitionOfSpeeds *partition)
{
 short int num;

 for(num = 0;num < NUM_SPEED_SETS - 1;++num)
   FreeFuzzySet1(partition->Members[num].FuzzySet);
}


short int FindMaxPartitionMember(short int IR1, short int IR2,
                                 short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{
 short int max_partition_member, i;

 max_partition_member = KheperaSensorFuzzyValues[IR1];
 for(i = IR1; i <= IR2; ++i)
  if((KheperaSensorFuzzyValues[i] - max_partition_member ) > 0)
    max_partition_member = i;

 return(max_partition_member);
}




short int  FindDistancePartitionMember(short int crisp_distance,
                                       struct PartitionOfDistances *partition)
```

```
/* return fuzzyset member and
   the number of membership of this to the partition */
{
  struct PartitionMember member;
  short int found, num;

  num = 0;
  found = -1;
  member = partition->Members[num];
  while((found == -1) && ((NUM_DIST_SETS - num) > 0 ))
  {
    if(((crisp_distance - member.IntervalMember[0]) >= 0) &&
          ((crisp_distance - member.IntervalMember[1]) <= 0))

          found = num;

    else
    {
      num++;
      member = partition->Members[num];
    }
  }
  return(found);

}



short int FindSpeedPartitionMember(short int crisp_speed,
                                   struct PartitionOfSpeeds *partition)
{
  struct PartitionMember member;
  short int num, found;

  num = 0;
  found = -1;
  member = partition->Members[num];
  while((found == -1) && ((NUM_SPEED_SETS - num) > 0 ))
  {
    if(((crisp_speed - member.IntervalMember[0]) >= 0) &&
          ((crisp_speed - member.IntervalMember[1]) <= 0))

          found = num;

    else
    {
      num++;
      member = partition->Members[num];
    }
  }
  return(found);
}


short int  SensorValueFuzzification(struct Robot *robot,
                                    short int sensor,
                                    struct PartitionOfDistances *partition)
{
```

```
  short int dist_member;

  dist_member =FindDistancePartitionMember(robot->IRSensor[sensor].DistanceValue,partition);
  return(dist_member);
}


short int MotorValueFuzzification(struct Robot *robot,
                                  short int motor,
                                  struct PartitionOfSpeeds *partition)
{
 short int speed_member;

 speed_member = FindSpeedPartitionMember(robot->Motor[motor].Value,partition);

 return(speed_member);
}


void CompleteDistanceFuzzification(struct Robot *robot,
                                   struct PartitionOfDistances *partition,
                                   short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
/* returns a group of member of partition of distance */
{
 short int num_sensor;

 for(num_sensor = 0; num_sensor < NUM_IRSENSORS; ++num_sensor)
 {
  KheperaSensorFuzzyValues[num_sensor] = SensorValueFuzzification(robot,
                                                                  num_sensor,
                                                                  partition);
 }
}


void CompleteSpeedFuzzification(struct Robot *robot,
                                struct PartitionOfSpeeds *partition,
                                short int KheperaMotorFuzzyValues[NUM_MOTORS-1])
/* returns a group of member of partition of speeds */
{
 short int num_motor;

 for(num_motor = 0; num_motor < NUM_MOTORS; ++num_motor)
 {
  KheperaMotorFuzzyValues[num_motor] = MotorValueFuzzification(robot,
                                                               num_motor,
                                                               partition);

 }
}


void WriteCurrentFuzzyDistancesToFile(FILE *file,
                                      short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1],
                                      struct PartitionOfDistances *partition,
                                      short int step)
{
```

```
  short int sensor,num;
  struct FuzzySet1 *member;
  fprintf(file,"# File       : FUZZY\fuzzy_distancessensors\n");
  fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Description : Current FUZZY DISTANCES of Khepera\n");
  fprintf(file,"# Date       :      \n");
  fprintf(file," ------------------------------------------------\n");
  fprintf(file,"  *STEP: %d\n",step);
  fprintf(file," ------------------------------------------------\n\n\n");
  fprintf(file,"  *Khepera Infra Red Sensors:\n\n");
  for(sensor = 0; sensor < NUM_IRSENSORS; ++sensor)
  {
    num = KheperaSensorFuzzyValues[sensor];
    if (sensor==0 || sensor==1 || sensor==2)
     fprintf(file,"  IR Sensor %d LEFT\n",sensor);
    else if (sensor==3 || sensor==4 || sensor==5)
     fprintf(file,"  IR Sensor %d RIGHT\n",sensor);
    else /* (sensor==6 || sensor==7) */
     fprintf(file,"  IR Sensor %d BACK\n",sensor);
    fprintf(file," \t-Number of fuzzy distance member: %d\n",num);
    fprintf(file," \t-Interval of crisp sensor values: [%d,%d]\n",
                                          partition->Members[num].IntervalMember[0],
                                          partition->Members[num].IntervalMember[1]);
    member = partition->Members[num].FuzzySet;
    fprintf(file," \t-Fuzzy distance set member: %s\n\n",
                                                  member->Properties.Name);
    fprintf(file," \t\t-XSupport: [%d,%d]",
                                    member->Support[0],
                                    member->Support[1]);
    fprintf(file," \t-Center:   %f\n\n",
                                    member->Center);

  }

}

void WriteCurrentFuzzySpeedsToFile(FILE *file,
                                   short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                   struct PartitionOfSpeeds *partition,
                                   short int step)
{
 short int motor,num;
  struct FuzzySet1 *member;

  fprintf(file,"# File       : FUZZY\fuzzy_speedsmotors\n");
  fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel.\n");
  fprintf(file,"# Description : Current FUZZY SPEEDS of Khepera\n");
  fprintf(file,"# Date       :      \n");
  fprintf(file," ------------------------------------------------\n");
  fprintf(file,"  *STEP: %d\n",step);
  fprintf(file," ------------------------------------------------\n\n\n");
  fprintf(file,"  *Khepera motors:\n\n");
  for(motor = 0; motor < NUM_MOTORS; ++motor)
  {
    num = KheperaMotorFuzzyValues[motor];
    if (motor == 0)
     fprintf(file,"  Motor %d: RIGHT\n",motor);
    else /*(motor = 1)*/
```

```
      fprintf(file,"  Motor %d: LEFT\n",motor);
    fprintf(file," \t-Number of fuzzy speed member: %d\n",num);
    fprintf(file," \t-Interval of crisp motor values: [%d,%d]\n",
                                                partition->Members[num].IntervalMember[0],
                                                partition->Members[num].IntervalMember[1]);
    member = partition->Members[num].FuzzySet;
    fprintf(file," \t-Fuzzy speed set member: %s\n\n",
                                                  member->Properties.Name);
    fprintf(file," \t\t-XSupport: [%d,%d] ",
                                            member->Support[0],
                                            member->Support[1]);
    fprintf(file," \t-Center:   %f\n\n",member->Center);
  }
}


short int CAM (struct FuzzySet1 *variation, float inference)
{

 float        ponderation;
 short int    crisp_value,
              var0, var1, var;


 var0 = DecFindCrispValue1(variation,inference,
                           variation->Center, variation->Support[1]);

 var1 = IncFindCrispValue1(variation,inference,
                           variation->Support[0],variation->Center);

 crisp_value = ( short int )(var0 + var1)/2;
 crisp_value = ( short int ) crisp_value/10;

 return(crisp_value);
}
```

USER.H

```
#ifndef USER_H
#define USER_H

#define WORLD_SCALE (double) 2
#define WORLD_OFFSET_X 35
#define WORLD_OFFSET_Y 22

#include "../CONTRIB/fuzzysets.h"
#include "../CONTRIB/flc.h"
#include "../SRC/include.h"
#include "../SRC/types.h"

#include <X11/Xatom.h>
#include <X11/Xos.h>
#include <X11/keysym.h>
#include <X11/cursorfont.h>
```

```
#include <X11/Xutil.h>

#define FUNCTION_TEXT    25
#define TEXT_SIZE        64
#define INFO_NEXT        20

extern struct FuzzySet1 *vf, *f, *pmod, *nulla, *nmod, *s, *vs;
extern struct FuzzySet1 *vpb, *pb, *pm, *ps, *ze, *ns, *nm, *nb, *vnb;
extern struct FuzzySet1 *nop, *olp, *omp, *ohp, *collision;
extern struct FuzzySet1 *his, *mis, *lis, *nms, *lds, *mds, *hds;
extern struct Context  *context;
extern struct Robot    *r;
extern struct Button   *b;
extern void CreatePrimaryFuzzyDistances();
extern void CreatePrimaryFuzzySpeeds();
extern void CreateFuzzySetsOfPerception();
extern struct PartitionOfDistances *CreatePartitionsOfDistances(short int i[2]);
extern struct PartitionOfSpeeds *CreatePartitionOfSpeeds(short int i[2]);

extern void WritePartitionOfDistancesToFile(struct PartitionOfDistances *partition,
                                            FILE *file);
extern void WritePartitionOfSpeedsToFile(struct PartitionOfSpeeds *partition,
                                         FILE *file);
extern short intFindMaxPartitionMember(short int IR1, short int IR2,
                                       short int fd[NUM_IRSENSORS-1]);
extern void CompleteDistanceFuzzification(struct Robot *r,
                                          struct PartitionOfDistances *partition,
                                          short int fd[NUM_IRSENSORS-1]);
extern void CompleteSpeedFuzzification(struct Robot *r,
                                       struct PartitionOfSpeeds *partition,
                                       short int fs[NUM_MOTORS-1]);
extern void WriteCurrentFuzzyDistancesToFile(FILE *ffile,
                                          short int fd[NUM_IRSENSORS-1],
                                          struct PartitionOfDistances *partition,
                                          short int step);
extern void WriteCurrentFuzzySpeedsToFile(FILE *ffile,
                                          short int fs[NUM_MOTORS-1],
                                          struct PartitionOfSpeeds *partition,
                                          short int step);
extern struct PerceptionStates *CreateFuzzyPerceptionStates(short int
                              fd[NUM_IRSENSORS-1]);
extern void WriteFuzzyPerceptionStatesToFile(struct PerceptionStates *percetions,
                                          struct Robot *robot,
                                          short int step,
                                          FILE *file);
extern void CreateFuzzySetsOfSpeedVariation();
extern struct RuleBase *CreateEmptyRuleBase();
extern void FreeRuleBase(struct RuleBase *rule_base);
extern struct RuleBase *ReadRuleBaseFromFile(FILE *file);
extern void WriteRuleBaseToFile(struct RuleBase *rule_base,FILE *file);
extern struct FuzzySet2 *CreateFiringStrengths (char Conjunction[MAX_TEXT],
                                                char m_function[MAX_TEXT],
                                                struct PartitionOfDistances *part_dists,
                                                struct PartitionOfSpeeds *part_speeds);
extern void WriteFiringStregthsToFile(struct FuzzySet2 *strengths, FILE *file);
extern struct SpeedVariation *FuzzyImplication(struct PerceptionStates *perceptions,
                                                short int KheperaMotorFuzzyValues[NUM_MOTORS-1],
                                                struct RuleBase *rule_base,
```

```
                                                    struct FuzzySet2 *strengths,
                                                    struct PartitionOfSpeeds *partition,
                                                    char Implication[MAX_TEXT]);
extern float DisjunctionOfFiringStrengths(struct FuzzySet2 *strengths,
                                                  char Disjunction[MAX_TEXT]);
extern short int CAM (struct FuzzySet1 *variation, float inference);
extern void CorrectMotorValues(struct Robot *robot);
extern struct SpeedVariation *CreateNullSpeedVariation();
extern struct SpeedVariation FreeSpeedVariations(struct SpeedVariation *variations);
extern float InferControlAction(float alpha,struct FuzzySet1 *variation,
                                char Implication[MAX_TEXT]);
extern float InferControlActionMembership(float alpha,struct FuzzySet1 *variation, char
Implication[MAX_TEXT]);



#define DrawPlateUp(x,y,w,h)        DrawPlate(x+X_O,y+Y_O,w,h,GREY,PLATE_DOWN);
#define DrawPlateDown(x,y,w,h)      DrawPlate(x+X_O,y+Y_O,w,h,LIGHT_GREY,PLATE_UP);
#define DrawPlateColor(x,y,w,h,c)   DrawPlate(x+X_O,y+Y_O,w,h,c,PLATE_UP);
#define DrawFuzzyDistances(r,pd,fd)    DrawInputFuzzySet(40,125,180,70,r,pd,fd);
#define DrawFuzzySpeeds(r,sp,fs)       DrawOutputFuzzySet(40,235,180,55,r,sp,fs);
#define DrawFuzzyPerceptions(r,per)    DrawStatePerception(245,125,220,70,r,per);
#define DrawFuzzySpeedVariation(r,var) DrawSpeedVariation(245,235,220,55,r,var);
#define DrawStepTable(r)               DrawTable(40,90,r);



extern void DrawPlate(int x, int y, int w, int h, u_char c, u_char state);
extern void DrawInputFuzzySet(int x, int y, int w, int h, struct Robot *r,
                              struct PartitionOfDistances *partition,
                              short int fd[NUM_IRSENSORS-1]);
extern void DrawOutputFuzzySet(int x, int y, int w, int h, struct Robot *r,
                               struct PartitionOfSpeeds *partition,
                               short int fs[NUM_MOTORS-1]);
extern void DrawStatePerception(int x, int y, int w, int h, struct Robot *robot,
                                struct PerceptionStates *perceptions);
extern void DrawSpeedVariation(int x, int y, int w, int h, struct Robot *robot,
                               struct SpeedVariation *variation);
extern void DrawTable(int x, int y, struct Robot *r);
extern void DrawUserInformation();
extern void DrawUserInformation2();
extern void PlotFuzzyProximities();
extern void PlotFuzzySpeeds();
extern void DrawFuzzification();
extern void DrawDataBase();
extern void DrawRuleBase();
extern void DrawDecisionMakingLogic();
extern void InitStepControllerFile(FILE *file);
extern void WriteStepController(FILE *file, short int pas);
extern short int FindMaxProximity(short int IR1, short int IR2, struct Robot *robot);
extern void WaitCursor();
extern void PointerCursor();

#endif
```

USER1.C

```
#include "../SRC/include.h"
#include "../CONTRIB/flc.h"
#include "user_info.h"
#include "user.h"

extern struct FuzzySet1 *his, *mis, *lis, *nms, *lds, *mds, *hds;

long int                pas=0;
short int               khepera_crisp_sensor_values[2] = {0,1023},
                        fuzzy_distances[NUM_IRSENSORS-1],
                        fuzzy_speeds[NUM_MOTORS-1],
                        khepera_crisp_motor_values[2]  = {-10,10};
char                    fuzzy_reasoning[MAX_TEXT] = "Mamdami";
struct PartitionOfDistances   *dist_partition;
struct PartitionOfSpeeds      *speed_partition;
struct PerceptionStates       *fuzzy_states;
struct SpeedVariation         *fuzzy_variation;
struct RuleBase               *rulebase;
struct FuzzySet2              *firing_strengths;
float                          rule_contribs;
float                          DML_right,DML_left;
short int                     crisp_variation[2];


FILE *gnuplot_file,
     *path_file,
     *leftspeed_file,
     *rightspeed_file,
     *leftsensors_file,
     *rightsensors_file,
     *backsensors_file,
     *step_controller;



void UserInit(struct Robot *robot)
{
 gnuplot_file = popen("gnuplot","w");
 if (gnuplot_file == NULL)
 {
        fprintf(stderr,"Warning: gnuplot not available");
 }
 ShowUserInfo(1,1);
 CreatePrimaryFuzzyDistances();
 CreatePrimaryFuzzySpeeds();
 dist_partition  = CreatePartitionOfDistances(khepera_crisp_sensor_values);
 speed_partition = CreatePartitionOfSpeeds(khepera_crisp_motor_values);
 CompleteDistanceFuzzification(robot,dist_partition,fuzzy_distances);
 CompleteSpeedFuzzification(robot,speed_partition,fuzzy_speeds);
 CreateFuzzySetsOfPerception();
 fuzzy_states = CreateFuzzyPerceptionStates(fuzzy_distances);
```

```
 rulebase = CreateEmptyRuleBase();
 CreateFuzzySetsOfSpeedVariation();
 firing_strengths = CreateFiringStrengths( "Union2", "Triangular2",dist_partition,speed_partition);
 rule_contribs = DisjunctionOfFiringStrengths(firing_strengths,"alg union");
 fuzzy_variation = CreateNullSpeedVariation();

}


void UserClose(struct Robot *robot)
{
 if (gnuplot_file) pclose(gnuplot_file);
 FreeRuleBase(rulebase);
 FreeSpeedVariations(fuzzy_variation);
}


void NewRobot(struct Robot *robot)
{
 pas = 0;
 ShowUserInfo(2,1);
 FreeRuleBase(rulebase);
 FreeSpeedVariations(fuzzy_variation);
 DML_right                = 0.0;
 DML_left                 = 0.0;
 crisp_variation[RIGHT] = 0;
 crisp_variation[LEFT]  = 0;
}


void LoadRobot(struct Robot *robot,FILE *file)
{
 ShowUserInfo(2,1);

}


void SaveRobot(struct Robot *robot,FILE *file)
{
 ShowUserInfo(2,1);
}


void RunRobotStart(struct Robot *robot)
{
 path_file       = fopen("STATS/path.dat","w"); /* modify to "a"*/
 leftspeed_file   = fopen("STATS/leftspeed.dat","w");
 rightspeed_file  = fopen("STATS/rightspeed.dat","w");
 leftsensors_file  = fopen("STATS/leftsensors.dat","w");
 rightsensors_file = fopen("STATS/rightsensors.dat","w");
 backsensors_file  = fopen("STATS/backsensors.dat","w");
 step_controller  = fopen("EXAMPLES/FBM_FLC/step.controller","w");
 InitStepControllerFile(step_controller);
 if (GetUserInfo()!=3 || GetUserInfoPage()!=1)
        ShowUserInfo(3,1);
 fuzzy_variation = CreateNullSpeedVariation();
 }
```

```
void RunRobotStop(struct Robot *robot)
{
 fclose(path_file);
 fclose(leftspeed_file);
 fclose(rightspeed_file);
 fclose(leftsensors_file);
 fclose(rightsensors_file);
 fclose(backsensors_file);
 fclose(step_controller);
 ShowUserInfo(3,1);

}


boolean StepRobot(struct Robot *robot)
{
 pas++;
 DrawStep(pas);
 DrawPositionRobot(robot);
 DML_right = 0.0;
 DML_left = 0.0;

 fprintf(path_file,"%lg, %lg\n",robot->X,1000.0-robot->Y);
 fprintf(leftspeed_file,"%ld, %ld\n",pas,robot->Motor[LEFT].Value);
 fprintf(rightspeed_file,"%ld, %ld\n",pas,robot->Motor[RIGHT].Value);
 CompleteDistanceFuzzification(robot,dist_partition,fuzzy_distances);
 CompleteSpeedFuzzification(robot,speed_partition,fuzzy_speeds);
 DrawFuzzySpeeds(robot,speed_partition,fuzzy_speeds);
 DrawFuzzyDistances(robot,dist_partition,fuzzy_distances);
 fuzzy_states = CreateFuzzyPerceptionStates(fuzzy_distances);
 DrawFuzzyPerceptions(robot,fuzzy_states);
  fuzzy_variation = FuzzyImplication(fuzzy_states,fuzzy_speeds,
                                     rulebase,firing_strengths,
                                     speed_partition,fuzzy_reasoning);
  DML_right=InferControlActionMembership(rule_contribs,
                                         fuzzy_variation->RightVariation,fuzzy_reasoning);
  DML_left=InferControlActionMembership(rule_contribs,
                                        fuzzy_variation->LeftVariation,fuzzy_reasoning);
   crisp_variation[RIGHT] = CAM(fuzzy_variation->RightVariation,DML_right);
   crisp_variation[LEFT] = CAM(fuzzy_variation->LeftVariation,DML_left);
   robot->Motor[RIGHT].Value += crisp_variation[RIGHT];
   robot->Motor[LEFT].Value += crisp_variation[LEFT];
   CorrectMotorValues(robot);
   DrawFuzzySpeedVariation(robot,fuzzy_variation);
   fprintf(leftsensors_file,"%ld, %lg\n",pas,1000.0-10);
   fprintf(rightsensors_file,"%ld, %lg\n",pas,1000.0-10);
   fprintf(backsensors_file,"%ld, %lg\n",pas,1000.0-10);
   fprintf(step_controller,"%6d %6d\n",crisp_variation[RIGHT],
           crisp_variation[LEFT]); /* for control action surface */
 return(TRUE);   /* continue the run of the robot */
}


/*****************************************************************/
```

```
void FastStepRobot(struct Robot *robot)  {}


void ResetRobot(struct Robot *robot)
{

 FILE *test1, *test2, *test3, *test4, *test5, *test6;

 test1 = fopen("STATS/path.dat","r");
 test2 = fopen("STATS/leftspeed.dat","r");
 test3 = fopen("STATS/rightspeed.dat","r");
 test4 = fopen("STATS/leftsensors.dat","r");
 test5 = fopen("STATS/rightsensors.dat","r");
 test6 = fopen("STATS/backsensors.dat","r");

 if (test1)
 {
        fclose(test1);
        system("rm STATS/path.dat");
 }
 else if (test2)
 {
        fclose(test2);
        system("rm STATS/leftspeed.dat");
 }
 else if (test3)
 {
        fclose(test3);
        system("rm STATS/rightspeed.dat");
 }
 else if (test4)
 {
        fclose(test4);
        system("rm STATS/leftsensors.dat");
 }
 else if (test5)
 {
        fclose(test5);
        system("rm STATS/rightsensors.dat");
 }
 else if (test6)
 {
        fclose(test6);
        system("rm STATS/backsensors.dat");
 }
 pas = 0;
 ShowUserInfo(3,1);
}


void UserCommand(struct Robot *robot,char *text)
{
 FILE *file, *ffile, *rules;
 char file_name[TEXT_SIZE], name[TEXT_SIZE], extension[TEXT_SIZE];
 short int num;
 struct FuzzySet1 *member;
```

```
if (strcmp(text, "plot path")==0)
{
        fprintf(gnuplot_file,"load 'GNUPLOT/path.gnu'\n");
        fflush(gnuplot_file);
        WriteComment("path plotted");
}
else if (strcmp(text, "plot left speed values")==0)
{
        fprintf(gnuplot_file,"load 'GNUPLOT/leftspeed.gnu'\n");
        fflush(gnuplot_file);
        WriteComment("Left Speed plotted");
}
else if (strcmp(text, "plot right speed values")==0)
{
        fprintf(gnuplot_file,"load 'GNUPLOT/rightspeed.gnu'\n");
        fflush(gnuplot_file);
        WriteComment("Right Speed plotted");
}
else if (strcmp(text, "plot left sensor values")==0)
{
        fprintf(gnuplot_file,"load 'GNUPLOT/leftsensors.gnu'\n");
        fflush(gnuplot_file);
        WriteComment(" Maximum Left IR Sensor Values plotted");
}
else if (strcmp(text, "plot right sensor values")==0)
{
        fprintf(gnuplot_file,"load 'GNUPLOT/rightsensors.gnu'\n");
        fflush(gnuplot_file);
        WriteComment("Maximum Right IR Sensor Values plotted");
}
else if (strcmp(text, "plot back sensor values")==0)
{
        fprintf(gnuplot_file, "load 'GNUPLOT/backsensors.gnu'\n");
        fflush(gnuplot_file);
        WriteComment("Maximum Back IR Sensor Values plotted");
}
else if (strcmp(text, "write primary fuzzy distances to file")==0)
{
 if (GetUserInfo()!=4 || GetUserInfoPage()!=1)  ShowUserInfo(4,1);

 WriteComment("saving.....FUZZY DISTANCES");
 member = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 for(num  = 0; num < NUM_DIST_SETS; ++num)
 {
   member = dist_partition->Members[num].FuzzySet;
   strcpy(name,member->Properties.Name);
   strcpy(extension,".fuzzyset");
   strcat(name,extension);
   strcpy(file_name,"FUZZY/");
   strcat(file_name,name);
   ffile=fopen(file_name,"w");
   WriteFuzzySet1ToFile(member,ffile);
   fclose(ffile);
 }
 WriteComment("FUZZY DISTANCES.....saved");

}
else if (strcmp(text, "write primary fuzzy speeds to file")==0)
```

```
{
 if (GetUserInfo()!=4 || GetUserInfoPage()!=2) ShowUserInfo(4,2);
 WriteComment("saving.....FUZZY SPEEDS");
 member = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
 for(num  = 0; num < NUM_SPEED_SETS; ++num)
 {
   member = speed_partition->Members[num].FuzzySet;
   strcpy(name,member->Properties.Name);
   strcpy(extension,".fuzzyset");
   strcat(name,extension);
   strcpy(file_name,"FUZZY/");
   strcat(file_name,name);
   ffile=fopen(file_name,"w");
   WriteFuzzySet1ToFile(member,ffile);
   fclose(ffile);
 }
 WriteComment("FUZZY SPEEDS.....saved");

}
else if (strcmp(text, "write partition of distances to file")==0)
{
 if (GetUserInfo()!=4 || GetUserInfoPage()!=1)  ShowUserInfo(4,1);
 WriteComment("saving ...partition of distances");
 strcpy(name,"distances.partition");
 strcpy(file_name,"FUZZY/");
 strcat(file_name,name);
 ffile=fopen(file_name,"w");
 WritePartitionOfDistancesToFile(dist_partition,ffile);
 fclose(ffile);
 WriteComment("partition of distances....saved");
}
else if (strcmp(text, "write partition of speeds to file")==0)
{
 if (GetUserInfo()!=4 || GetUserInfoPage()!=2)  ShowUserInfo(4,2);
 WriteComment("saving ....partition of speeds");
 strcpy(name,"speeds.partition");
 strcpy(file_name,"FUZZY/");
 strcat(file_name,name);
 ffile=fopen(file_name,"w");
 WritePartitionOfSpeedsToFile(speed_partition,ffile);
 fclose(ffile);
 WriteComment("partition of speeds ....saved");
}
else if (strcmp(text, "write current fuzzy distances to file")==0)
{
 if (GetUserInfo()!=4 || GetUserInfoPage()!=1)  ShowUserInfo(4,1);
 WriteComment("saving ....Khepera current FUZZY DISTANCES");
 strcpy(name,"fuzzy_distances.sensors");
 strcpy(file_name,"FUZZY/");
 strcat(file_name,name);
 ffile=fopen(file_name,"w");
 WriteCurrentFuzzyDistancesToFile(ffile,fuzzy_distances,dist_partition,pas);
 fclose(ffile);
 WriteComment("current Khepera FUZZY DISTANCES ....saved");

}
else if (strcmp(text, "write current fuzzy speeds to file")==0)
{
```

```
  if (GetUserInfo()!=4 || GetUserInfoPage()!=2)  ShowUserInfo(4,2);
  WriteComment("saving ....Khepera current FUZZY SPEEDS");
  strcpy(name,"fuzzy_speeds.motors");
  strcpy(file_name,"FUZZY/");
  strcat(file_name,name);
  ffile=fopen(file_name,"w");
  WriteCurrentFuzzySpeedsToFile(ffile,fuzzy_speeds,speed_partition,pas);
  fclose(ffile);
  WriteComment("current Khepera FUZZY SPEEDS ....saved") ;

 }
 else if (strcmp(text, "write fuzzy perceptions to file")==0)
 {
  if (GetUserInfo()!=4 || GetUserInfoPage()!=1)  ShowUserInfo(4,1);
  WriteComment("saving ....Khepera current FUZZY PERCEPTIONS");
  strcpy(name,"fuzzy.perceptions");
  strcpy(file_name,"FUZZY/");
  strcat(file_name,name);
  ffile=fopen(file_name,"w");
  WriteFuzzyPerceptionStatesToFile(fuzzy_states,robot,pas,ffile);
  fclose(ffile);
  WriteComment("current Khepera FUZZY PERCEPTIONS ....saved") ;
 }
 else if (strcmp(text, "write fuzzy sets of speed variations")==0)
 {
    if (GetUserInfo()!=2 || GetUserInfoPage()!=1) ShowUserInfo(2,1);
   WriteComment("saving....FUZZY VARIATIONS OF SPEEDS");
   member = (struct FuzzySet1 *)malloc(sizeof(struct FuzzySet1));
   member = his;
   strcpy(name,member->Properties.Name);
   strcpy(extension,".fuzzyset");
   strcat(name,extension);
   strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
   strcat(file_name,name);
   ffile=fopen(file_name,"w");
   WriteFuzzySet1ToFile(member,ffile);
   fclose(ffile);
   member = mis;
   strcpy(name,member->Properties.Name);
   strcpy(extension,".fuzzyset");
   strcat(name,extension);
   strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
   strcat(file_name,name);
   ffile=fopen(file_name,"w");
   WriteFuzzySet1ToFile(member,ffile);
   fclose(ffile);
   member = lis;
   strcpy(name,member->Properties.Name);
   strcpy(extension,".fuzzyset");
   strcat(name,extension);
   strcpy(file_name,"FUZZY/");
   strcat(file_name,name);
   ffile=fopen(file_name,"w");
   WriteFuzzySet1ToFile(member,ffile);
   fclose(ffile);
   member = nms;
   strcpy(name,member->Properties.Name);
   strcpy(extension,".fuzzyset");
```

```
  strcat(name,extension);
  strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
  strcat(file_name,name);
  ffile=fopen(file_name,"w");
  WriteFuzzySet1ToFile(member,ffile);
  fclose(ffile);
  member = lds;
  strcpy(name,member->Properties.Name);
  strcpy(extension,".fuzzyset");
  strcat(name,extension);
  strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
  strcat(file_name,name);
  ffile=fopen(file_name,"w");
  WriteFuzzySet1ToFile(member,ffile);
  fclose(ffile);                                  /* TO BE MODIFIED */
  member = mds;                                         /* GETTING SETS  */
  strcpy(name,member->Properties.Name);                 /* FROM RULES    */
  strcpy(extension,".fuzzyset");
  strcat(name,extension);
  strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
  strcat(file_name,name);
  ffile=fopen(file_name,"w");
  WriteFuzzySet1ToFile(member,ffile);
  fclose(ffile);
  member = hds;
  strcpy(name,member->Properties.Name);
  strcpy(extension,".fuzzyset");
  strcat(name,extension);
  strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
  strcat(file_name,name);
  ffile=fopen(file_name,"w");
  WriteFuzzySet1ToFile(member,ffile);
  fclose(ffile);
  WriteComment("FUZZY VARIATIONS OF SPEEDS.....saved");
}

/* Setting the control policy for the behavioural performance */
/* This command allows the analysis of different control policies */
/* Note: different RBs can be installed while Khepera is moving in env */



else if (strcmp(text, "AFLC")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("reading....fuzzy control rules");
 rules = fopen("ROBOT/rules1.robot","r");
 rulebase = ReadRuleBaseFromFile(rules);
 fclose(rules);
 WriteComment(".....done");
}
else if (strcmp(text, "WFFLC1")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("reading....fuzzy control rules");
 rules = fopen("ROBOT/wall_follow1.robot","r");
 rulebase = ReadRuleBaseFromFile(rules);
```

```
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC2a")==0)
 {
  if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
  WriteComment("reading....fuzzy control rules");
  rules = fopen("ROBOT/wall_follow2a.robot","r");
  rulebase = ReadRuleBaseFromFile(rules);
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC2b")==0)
 {
  if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
  WriteComment("reading....fuzzy control rules");
  rules = fopen("ROBOT/wall_follow2b.robot","r");
  rulebase = ReadRuleBaseFromFile(rules);
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC2")==0)
 {
  if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
  WriteComment("reading....fuzzy control rules");
  rules = fopen("ROBOT/wall_follow2.robot","r");
  rulebase = ReadRuleBaseFromFile(rules);
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC3")==0)
 {
  if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
  WriteComment("reading....fuzzy control rules");
  rules = fopen("ROBOT/wall_follow3.robot","r");
  rulebase = ReadRuleBaseFromFile(rules);
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC4")==0)
 {
  if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
  WriteComment("reading....fuzzy control rules");
  rules = fopen("ROBOT/wall_follow4.robot","r");
  rulebase = ReadRuleBaseFromFile(rules);
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC5")==0)
 {
  if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
  WriteComment("reading....fuzzy control rules");
  rules = fopen("ROBOT/wall_follow5.robot","r");
  rulebase = ReadRuleBaseFromFile(rules);
  fclose(rules);
  WriteComment(".....done");
 }
 else if (strcmp(text, "WFFLC6")==0)
```

```
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("reading....fuzzy control rules");
 rules = fopen("ROBOT/wall_follow6.robot","r");
 rulebase = ReadRuleBaseFromFile(rules);
 fclose(rules);
 WriteComment(".....done");
}
else if (strcmp(text, "WFFLC6a")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("reading....fuzzy control rules");
 rules = fopen("ROBOT/wall_follow6a.robot","r");
 rulebase = ReadRuleBaseFromFile(rules);
 fclose(rules);
 WriteComment(".....done");
}
else if (strcmp(text, "WFFLC7")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("reading....fuzzy control rules");
 rules = fopen("ROBOT/wall_follow7.robot","r");
 rulebase = ReadRuleBaseFromFile(rules);
 fclose(rules);
 WriteComment(".....done");
}
else if (strcmp(text, "WFFLC8")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("reading....fuzzy control rules");
 rules = fopen("ROBOT/wall_follow8.robot","r");
 rulebase = ReadRuleBaseFromFile(rules);
 fclose(rules);
 WriteComment(".....done");
}



/* end of commands for setting the fuzzy control rules */

else if (strcmp(text, "write rule base")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("saving ...fuzzy control rules");
 strcpy(name,"flc2.rules");
 strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
 strcat(file_name,name);
 ffile=fopen(file_name,"w");
 WriteRuleBaseToFile(rulebase,ffile);
 fclose(ffile);
 WriteComment("fuzzy control rules....saved");
}
else if (strcmp(text, "write firing strengths")==0)
{
 if (GetUserInfo()!=2 || GetUserInfoPage()!=1)  ShowUserInfo(2,1);
 WriteComment("saving ...firing strengths");
 strcpy(name,"firing.strengths");
```

```
   strcpy(file_name,"EXAMPLES/FL_EX2/FUZZY/");
   strcat(file_name,name);
   ffile=fopen(file_name,"w");
   WriteFiringStregthsToFile(firing_strengths,ffile);
   fclose(ffile);
   WriteComment("firing strengths....saved");
  }
  else if (strcmp(text, "apply Mamdami")==0)
  {
   strcpy(fuzzy_reasoning,"Mamdami");
   WriteComment("Mamdami fuzzy reasoning technique...  by A.M. Gonzalez");
  }
  else if(strcmp(text,"apply Larsen")==0)
  {
   strcpy(fuzzy_reasoning,"Larsen");
   WriteComment("Larsen fuzzy reasoning technique...  by A.M. Gonzalez");
  }
  else
        WriteComment("unknown command");  /* no commands */
     .
}


void DrawUserInfo(struct Robot *robot,u_char info,u_char page)
{
 char text[256];

 switch(info)
 {
    case 1:
     switch(page)
     {
          case 1: DrawUserInformation(robot);
          break;
         case 2: DrawUserInformation2(robot);
     }
     break;
    case 2:
     switch(page)
     {
      case 1: DrawController();
              break;
      case 2: DrawFuzzification();
              break;
      case 3: DrawDataBase();
             break;
      case 4: DrawRuleBase();
              break;
      case 5: DrawDecisionMakingLogic();
              break;
     }
     break;
    case 3:
        switch(page)
        {
         case 1: DrawStepTable(robot);
                 break;
        }
```

```
            break;
      case 4: /* primary fuzzy sets */
            switch(page)
            {
              case 1: PlotFuzzyProximities();
                      break;
              case 2: PlotFuzzySpeeds();
                      break;
            }
            break;
    }
}
```

USER2.C

```
#include "../SRC/include.h"
#include "../CONTRIB/flc.h"
#include "../CONTRIB/fuzzysets.h"
#include "user.h"

extern long int pas;
extern Display *display;
extern struct PartitionOfDistances  *dist_partition;
extern struct PartitionOfSpeeds     *speed_partition;
extern struct PerceptionStates      *fuzzy_states;
extern struct SpeedVariation        *fuzzy_variation;
extern short int                    fuzzy_distances[NUM_IRSENSORS-1];
extern short int                    fuzzy_speeds[NUM_MOTORS-1];
extern float                        DML_right,DML_left;
extern short int                    crisp_variation[2];
extern float                        rule_contribs;
extern char                         fuzzy_reasoning[MAX_TEXT];


void DrawPositionRobot(struct Robot *robot)
{
  char text[10];
  short int x,y;

  x = (robot->X);
  y = (robot->Y);
  sprintf(text,"position = < %3d,%3d >",x,y);
  DrawPlateUp(250,50,208,35);
  Color(BLACK);
  DrawText(270,70,text);
}


void DrawWindowPosition(int x, int y, int w, int h, struct Robot *robot)
{
  DrawPlateUp(x,y,w,h);
  DrawPositionRobot(robot);
}
```

```
void DrawStep(int p)
{
 char text[256];

 sprintf(text, "step = %d", p);
 DrawPlateUp(55,50,148,35);
 Color(BLACK);
 DrawText(75,70,text);

}


void DrawWindowStep(int x, int y, int w, int h, int p)
{
 DrawPlateUp(x,y,w,h);
 DrawStep(p);
}


void DrawInputFuzzySet(int x, int y, int w, int h, struct Robot *robot,
                       struct PartitionOfDistances *partition,
                       short int KheperaSensorFuzzyValues[NUM_IRSENSORS-1])
{

 char fuzzy_set_name[MAX_TEXT];
 short int MaxLeft, MaxRight, MaxBack;
 struct FuzzySet1 *member;


 DrawPlateUp(x,y,w,h);
 Color(BLUE);
 DrawText(x,y-5,"Fuzzy distances: ");
 Color(BLACK);
 DrawText(x+10,y+20,"Left Sensors  :");
 DrawText(x+10,y+40,"Right Sensors :");
 DrawText(x+10,y+60,"Back Sensors  :");
 Color(BLUE);
 MaxLeft = FindMaxPartitionMember(0,2,KheperaSensorFuzzyValues);
 member = partition->Members[MaxLeft].FuzzySet;
 strcpy(fuzzy_set_name,member->Properties.Name);
 DrawText(x+140,y+20,fuzzy_set_name);
 MaxRight = FindMaxPartitionMember(3,5,KheperaSensorFuzzyValues);
 member = partition->Members[MaxRight].FuzzySet;
 strcpy(fuzzy_set_name, member->Properties.Name);
 DrawText(x+140,y+40,fuzzy_set_name);
 MaxBack = FindMaxPartitionMember(6,7,KheperaSensorFuzzyValues);
 member = partition->Members[MaxBack].FuzzySet;
 strcpy(fuzzy_set_name,member->Properties.Name);
 DrawText(x+140,y+60,fuzzy_set_name);
 Color(BLACK);
 DrawText(x+180,y+20,">>>");
 DrawText(x+180,y+40,">>>");
 DrawText(x+180,y+60,">>>");

}
```

```
void PlotFuzzyProximities()
{
 DrawPlateDown(20,20,460,295);
 WriteComment("Fuzzy Sets");
 Color(BLUE);
 DrawText(40,40,"Fuzzy Sets of Distances:");
 Color(BLACK);
 DrawLine(83,70,83,275);
 DrawLine(50,275,450,275);
 DrawText(75,110,"1");
 DrawText(50,288,"-100");
 DrawText(75,271,"0");
 DrawText(88,288,"0");
 DrawText(106,288,"100");
 DrawText(139,288,"200");
 DrawText(171,288,"300");
 DrawText(205,288,"400");
 DrawText(237,288,"500");
 DrawText(270,288,"600");
 DrawText(306,288,"700");
 DrawText(339,288,"800");
 DrawText(372,288,"900");
 DrawText(405,288,"1000");
 DrawText(441,288,"1100");
 Color(BLUE);
 DrawLine(50,275,83,100);
 DrawLine(83,100,119,275);
 DrawLine(107,275,152,100);
 DrawLine(152,100,193,275);
 DrawLine(176,275,201,100);
 DrawLine(201,100,226,275);
 DrawLine(218,275,234,100);
 DrawLine(234,100,250,275);
 DrawLine(234,275,250,100);
 DrawLine(250,100,266,275);
 DrawLine(250,275,266,100);
 DrawLine(266,100,283,275);
 DrawLine(274,275,299,100);
 DrawLine(299,100,327,275);
 DrawLine(307,275,352,100);
 DrawLine(352,100,393,275);
 DrawLine(385,275,418,100);
 DrawLine(418,100,450,275);
 DrawText(72,97,"VPB");
 DrawText(145,97,"PB");
 DrawText(195,97,"PM");
 DrawText(225,97,"PS");
 DrawText(244,97,"ZE");
 DrawText(261,97,"NS");
 DrawText(293,97,"NM");
 DrawText(345,97,"NB");
 DrawText(405,97,"VNB");
}

void DrawOutputFuzzySet(int x, int y, int w, int h, struct Robot *robot,
                        struct PartitionOfSpeeds *partition,
```

```
                        short int KheperaMotorFuzzyValues[NUM_MOTORS-1])
{
  struct FuzzySet1 *member;
  short int partition_member;
  char fuzzy_set_name[MAX_TEXT];

  DrawPlateUp(x,y,w,h);
  Color(RED);
  DrawText(x,y-5,"Fuzzy speeds:");
  Color(BLACK);
  DrawText(x+10,y+20,"Left Motor  :");
  DrawText(x+10,y+40,"Right Motor  :");
  Color(RED);
  partition_member = KheperaMotorFuzzyValues[0];
  member = partition->Members[partition_member].FuzzySet;
  strcpy(fuzzy_set_name,member->Properties.Name);
  DrawText(x+135,y+20,fuzzy_set_name);
  partition_member = KheperaMotorFuzzyValues[1];
  member = partition->Members[partition_member].FuzzySet;
  strcpy(fuzzy_set_name,member->Properties.Name);
  DrawText(x+135,y+40,fuzzy_set_name);
  Color(BLACK);
  DrawText(x+180,y+20,">>>");
  DrawText(x+180,y+40,">>>");

}



void PlotFuzzySpeeds()
{
  DrawPlateDown(20,20,460,295);
  WriteComment("Fuzzy Sets");
  Color(RED);
  DrawText(40,40,"Fuzzy Sets of Speeds:");
  Color(BLACK);
  DrawLine(250,70,250,275);
  DrawLine(30,275,470,275);
  DrawText(242,105,"1");
  DrawText(242,271,"0");
  DrawText(22,288,"-11");
  DrawText(42,288,"-10");
  DrawText(67,288,"-9");
  DrawText(87,288,"-8");
  DrawText(107,288,"-7");
  DrawText(127,288,"-6");
  DrawText(147,288,"-5");
  DrawText(167,288,"-4");
  DrawText(187,288,"-3");
  DrawText(207,288,"-2");
  DrawText(227,288,"-1");
  DrawText(247,288,"0");
  DrawText(262,288,"1");
  DrawText(282,288,"2");
  DrawText(302,288,"3");
  DrawText(322,288,"4");
```

```
  DrawText(342,288,"5");
  DrawText(362,288,"6");
  DrawText(382,288,"7");
  DrawText(402,288,"8");
  DrawText(422,288,"9");
  DrawText(442,288,"10");
  DrawText(462,288,"11");
  Color(RED);
  DrawLine(30,275,50,100);
  DrawLine(50,100,70,275);
  DrawLine(50,275,110,100);
  DrawLine(110,100,170,275);
  DrawLine(150,275,190,100);
  DrawLine(190,100,230,275);
  DrawLine(210,275,250,100);
  DrawLine(250,100,290,275);
  DrawLine(270,275,310,100);
  DrawLine(310,100,350,275);
  DrawLine(330,275,390,100);
  DrawLine(390,100,450,275);
  DrawLine(430,275,450,100);
  DrawLine(450,100,470,275);
  DrawText(42,92,"VS");
  DrawText(107,92,"S");
  DrawText(182,92,"NM");
  DrawText(237,92,"NULL");
  DrawText(305,92,"PM");
  DrawText(390,92,"F");
  DrawText(445,92,"VF");
}


void DrawSpeedVariation(int x, int y, int w, int h, struct Robot *robot,
                        struct SpeedVariation *variation)
{

  char fuzzy_set_name[MAX_TEXT];


  DrawPlateUp(x,y,w,h);
  Color(CORAL);
  DrawText(x,y-5,"Fuzzy variation of speed:");
  strcpy(fuzzy_set_name,variation->LeftVariation->Properties.Name);
  DrawText(x+10,y+20,fuzzy_set_name);
  strcpy(fuzzy_set_name,variation->RightVariation->Properties.Name);
  DrawText(x+10,y+40,fuzzy_set_name);
}


void DrawStatePerception(int x, int y, int w, int h, struct Robot *robot,
                         struct PerceptionStates *perceptions)
{

  char fuzzy_set_name[MAX_TEXT];

  DrawPlateUp(x,y,w,h);
  Color(SEA_GREEN);
```

```
  DrawText(x,y-5,"Fuzzy perception state:");
  Color(SEA_GREEN);
  strcpy(fuzzy_set_name,perceptions->LeftState->Properties.Name);
  DrawText(x+10,y+20,fuzzy_set_name);
  strcpy(fuzzy_set_name,perceptions->RightState->Properties.Name);
  DrawText(x+10,y+40,fuzzy_set_name);
  strcpy(fuzzy_set_name,perceptions->BackState->Properties.Name);
  DrawText(x+10,y+60,fuzzy_set_name);
}


void DrawTable(int x, int y, struct Robot *robot)
{
        char text[MAX_TEXT];
        char text2[MAX_TEXT];

        DrawPlateDown(25,20,460,300);
        Color(RED);
        strcpy(text2,"Fuzzy Reasoning Technique: ");
        strcat(text2,fuzzy_reasoning);
        strcat(text2,"'s Fuzzy Implication");
        DrawText(20,19,text2);
        DrawWindowStep(55,50,148,35,pas);
        DrawWindowPosition(250,50,208,35,robot);
        DrawInputFuzzySet(40,125,180,70,robot,dist_partition,
                          fuzzy_distances);
        DrawStatePerception(245,125,220,70,robot,fuzzy_states);
        DrawOutputFuzzySet(40,235,180,55,robot,speed_partition,
                           fuzzy_speeds);
    DrawSpeedVariation(245,235,220,55,robot,fuzzy_variation);
        Color(BLACK);
        DrawText(25,335,"Avoidance FLC version 1.0 by A.M. Gonzalez");
}


void DrawController()
{
        char text[100];
        char text2[MAX_TEXT];
        struct FuzzySet1 *set;

        WriteComment("FUZZY LOGIC CONTROL     by A.M. Gonzalez");
        Color(RED);
        DrawText(20,20,"Basic Structure of a FLC:");
        DrawPlateDown(20,25,460,285);
        DrawPlateColor(190,50,120,50,LIME_GREEN);
        Color(BLACK);
        DrawLine(100,75,190,75);
        DrawLine(310,75,400,75);
        DrawLine(100,75,100,105);
        DrawLine(400,75,400,105);
        DrawText(215,70,"Knowledge");
        DrawText(230,85,"Base");
        DrawLine(250,100,250,150);
        DrawPlateColor(30,105,140,50,SEA_GREEN);
        Color(BLACK);
        DrawText(45,135,"Fuzzification");
        DrawLine(120,155,120,190);
```

```
        DrawLine(120,190,190,190);
        Color(RED);
        DrawLine(80,155,80,260);
        DrawLine(80,260,130,260);
        DrawText(60,280,"OUTPUT");
        DrawPlateColor(330,105,140,50,SEA_GREEN);
        Color(BLACK);
        DrawText(345,135,"Defuzzification");
        DrawLine(360,155,360,190);
        DrawLine(360,190,310,190);
        Color(BLUE);
        DrawLine(420,155,420,260);
        DrawLine(420,260,180,260);
        DrawText(395,280,"INPUT");
        DrawPlateColor(190,130,120,90,AQUAMARINE);
        Color(BLACK);
        DrawText(220,160,"Decision");
        DrawText(225,180,"Making");
        DrawText(230,200,"Logic");
        DrawPlateColor(120,240,260,50,CORAL);
        Color(BLACK);
        DrawText(155,260,"Controlled System Process");
        DrawText(150,280,"Simulation STEP of Khepera ");
        DrawText(10,330,"For more information, press '+' button.");
}

void DrawFuzzification()
{
}

void DrawDataBase()
{
}

void DrawRuleBase()
{
}

void DrawDecisionMakingLogic()
{
}

void DrawUserInformation()
{
        Color(RED);
        DrawText(160,20,"FUZZY LOGIC: Example 2   by A.M. Gonzalez");
        Color(BLACK);
        DrawText(10,60,"This example shows the implementation of a Fuzzy Logic");
        DrawText(10,80,"Avoidance Controller for the Kephera mobile robot.");
        DrawText(10,120," Posible commands are:");
        DrawText(100,150,"- plot path");
        DrawText(100,170,"- plot left sensor values");
        DrawText(100,190,"- plot right sensor values");
```

```
        DrawText(100,210,"- plot back sensor values");
        DrawText(100,230,"- plot left speed values");
        DrawText(100,250,"- plot right speed values");
        DrawText(100,270,"- write primary fuzzy distances to file");
        DrawText(100,290,"- write primary fuzzy speeds to file");
        DrawText(10,330,"For more information, press '+' button.");
}

void DrawUserInformation2()
{
        Color(RED);
        DrawText(160,20,"FUZZY LOGIC: Example 2");
        Color(BLACK);
        DrawText(10,60,"some more available commands are:");
        DrawText(100,100,"- write partition of distances to file");
        DrawText(100,120,"- write partition of speeds to file ");
        DrawText(100,140,"- write current fuzzy distances to file");
        DrawText(100,160,"- write current fuzzy speeds to file");
        DrawText(100,180,"- write fuzzy perceptions to file");
        DrawText(100,200,"- write fuzzy sets of speed variations");
        DrawText(100,220,"- read fuzzy control rules from file");
        DrawText(100,240,"- write fuzzy control rules to file");
    DrawText(100,260,"- write firing strengths to fiel");
    DrawText(10,310,"To see the structure of the Fuzzy Logic Controller,");
    DrawText(10,330,"press 'info' button.");
}

void InitStepControllerFile(FILE *file)
{
 short int p;

 fprintf(file,"# File       : EXAMPLES|FL_EX2|FUZZYZ|flc2.rules\n");
 fprintf(file,"# Author     : Ana Maria Gonzalez de Miguel\n");
 fprintf(file,"# Description : FUZZY LOGIC CONTROLLER variables\n");
 fprintf(file,"# Date       :      \n");
 fprintf(file," ------------------------------------------------------------\n\n");
 fprintf(file,"*%s:\n",dist_partition->Name);
 for(p=0;p<NUM_DIST_SETS;++p)
  fprintf(file,"p %d: %s\n",dist_partition->Members[p].FuzzySet->Properties.Name);
 fprintf(file,"\n");
 fprintf(file,"*%s:\n",speed_partition->Name);
 for(p=0;p<NUM_SPEED_SETS;++p)
 fprintf(file,"p %d: %s\n",speed_partition->Members[p].FuzzySet->Properties.Name);
 fprintf(file,"\n\n");
}

void WriteStepController(FILE *file, short int pas)
{
 fprintf(file," ------------------------------------------------------------\n\n");
 fprintf(file,"*STEP: %d\n\n",pas);
 fprintf(file," - Fuzzy Distances\n");
 fprintf(file,"\t Left Side IR Sensors : %s\n",fuzzy_distances[0]);
 fprintf(file,"\t Right Side IR Sensors: %s\n",fuzzy_distances[1]);
 fprintf(file,"\t Back Side IR Sensors : %s\n\n",fuzzy_distances[2]);
 fprintf(file," - Fuzzy Speeds\n");
```

```
  fprintf(file,"\t Left Motor          : %s\n",fuzzy_speeds[RIGHT]);
  fprintf(file,"\t Right Side Motor     : %s\n\n",fuzzy_speeds[LEFT]);
  fprintf(file," - Fuzzy Perception States\n");
  fprintf(file,"\t Left Side Percep.    : %s\n",fuzzy_states->LeftState->Properties.Name);
  fprintf(file,"\t Right Side Percept. : %s\n",fuzzy_states->RightState->Properties.Name);
  fprintf(file,"\t Back Side Percept.   : %s\n\n",fuzzy_states->BackState->Properties.Name);
  fprintf(file," - Fuzzy Variation of Speed\n");
  fprintf(file,"\t Left Variation.      : %s\n",fuzzy_variation->LeftVariation->Properties.Name);
  fprintf(file,"\t Right Variation      : %s\n\n",fuzzy_variation->RightVariation->Properties.Name);
  fprintf(file,"- Decision Making Logic\n");
  fprintf(file,"\t DML Left : %f\n",DML_left);
  fprintf(file,"\t DML Right: %f\n\n",DML_right);
  fprintf(file,"- Defuzzification\n");
  fprintf(file,"\t Left Defuzzification : %d\n",crisp_variation[1]);
  fprintf(file,"\t Right Defuzzification: %d\n",crisp_variation[0]);
}
```

USER_INF.C

```
#ifndef USER_INFO_H

#define NUMBER_OF_INFO  4
#define PAGES_INFO_1    2
#define TITLE_INFO_1    "User Information"
#define PAGES_INFO_2    5
#define TITLE_INFO_2    "Fuzzy Logic Controller"
#define PAGES_INFO_3    1
#define TITLE_INFO_3    "Fuzzy I/O of Controller Process"
#define PAGES_INFO_4    4
#define TITLE_INFO_4    "Fuzzy Sets"

#include "../SRC/user_info.c"   /* assigns the values for the info pages */

#endif
```

Bibliography

Agre, P.-E. (1995), *Computational Research on Interaction and Agency*, Artificial Intelligence, Vol. 72, pp. 1-52, Elsevier.

Aitken, A.-M. (1994), *An Architecture for Learning to Behave*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 315-324, MIT Press/Bradford Books.

Albus, J.-S. (1996), *The Engineering of Mind*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 23, 34, MIT Press/Bradford Books.

Arbib, M.-A., Schweighofer, N. & Thach, W.-T. (1994), *Modeling the Role of Cerebellum in Prism Adaptation*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 36-44, MIT Press/Bradford Books.

Arbib, M.-A. & Liaw, J.-S. (1995), *Semsorimotor Transformations in the World of Frogs and Robots*, AI, N. 72 (1-2), pp. 53-79.

Arkin, R.-C. & Ali, K.-S. (1994), *Integration of Reactive and Telerobotic Control in Multi-Agent Robotic Systems*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 473-478, MIT Press/Bradford Books.

Azvine, B., Azarmi, N. & Tsuji, K.-C. (1996), *Soft-Computing -- A Tool for Building Intelligent Systems*, In: BT Technological Journal, Vol. 14, No. 4, pp. 37-45, Chapman & Hall.

Ball, N. (1994), *Organising Animat's Behavioural Repertoires Using Kohonen's Feature Maps*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 128-137, MIT Press/Bradford Books.

Barto, A.-G. (1990), *Some Learning Tasks from a Control Perspective*, Technical Report COINS TR 90-122, University of Massachusetts.

Barto, A.-G., Bradtke, S.-J. & Singh, S.-P. (1995), *Learning to Act Real-Time Dynamic Programming*, AI, N. 72 (1-2)m pp. 81-138.

Beer, R.-D., Chiel, H. & Sterling, L. (1990), *A Biological Perspective on Autonomous Agent Design*, In: Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, edited by Maes, P., MIT Press / Bradford Books.

Beer, R.-D. (1990), *Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology*, Academic Press, Cambridge Ma.

Beer, R.-D. (1995), *A Dynamical System Perspective on Agent-Environment Interaction*, AI, N. 72 (1-2), pp. 173-215.

Beer, R.-D. (1996), *Toward the Evolution of Dynamical Neural Networks for Minimally Cognitive Behavior*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 421-429, MIT Press/Bradford Books.

Belanger, J.-H. & Willis, M.-A. (1996), *Centrally-Generated and Reflexive Control Strategies in the Adaptive Behaviour of Real and Simulated Animals*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 155-162, MIT Press/Bradford Books.

Bellman, E. & Zadeh, L.-A. (1970), *Decision Making in a Fuzzy Environment*, Management Sciences, Vol. 17, No. 4.

Bersini, H. (1994), *Reinforcement Learning for Homeostatic Endogenous Variables*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 325-333, MIT Press/Bradford Books.

Bezdek, J. (1992), *IEEE Transactions on Neural Networks*, Special Issue on Fuzzy Logic and Neural Networks, Vol. 3. IEEE Neural Networks Council.

Blythe, P.-W., Miller, G.-F. & Todd, P.-M. (1996), *Human Simulation of Adaptive Behaviour: Interactive Studies on Pursuit, Evasion, Courtship, Fighting, and Play*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 13-22, MIT Press/Bradford Books.

Blumberg, B. (1994), *Action-Selection in Hamsterdam: Lessons from Ethology*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 108-117, MIT Press/Bradford Books.

Blumberg, B.-M., Todd, P.-M. & Maes, P. (1996), *No Bad Dogs: Ethological Lessons for Learning in Hamsterdam*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 295-304, MIT Press/Bradford Books.

Bohner, P. (1995), *A Multi-Agent Approach with Distributed Fuzzy Logic Control*, Computers in Industry, Vol. 26, pp. 219-227, Elsevier.

Bond, A. & Gasser, L. (1988), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, California.

Bonner, J.-T. (1988), *The Evolution of Complexity by Means of Natural Selection*, Princeton University Press, Princeton, N.-J.

Booker, L. (1988), *Classifier Systems That Learn Internal World Models*, Machine Learning Journal, Vol. 1, No. 2,3.

Bose, N.-K. & Liang, P. (1996), *Neural Network Fundamentals with Graph, Algorithms, and Applications*, Mc. Graw-Hill, USA.

Braitenberg, V. (1984), *Vehicles, Experiments in Synthetic Psychology*, MIT Press, Cambridge, Massachusetts.

Brannback, M. & Malaske, P. (1995), *Cognitive Mapping Approach Analysing Societal Decision-Making*, World Features, Vol. 44, pp. 231-245.

Brooks, R.-A. (1985), *Visual Map Making for a Mobile Robot*, In: Proceedings IEEE Conference on Robotics and Automation.

Brooks, R.-A. (1986), *A Robust Layered Control System For a Mobile Robot*, IEEE Journal of Robotics and Automation, RA-2, pp. 14-23.

Brooks, R.-A. (1990a), *The Behavior Language: User's Guide*, Technical Report AIM-1127, MIT Artificial Intelligence Laboratory.

Brooks, R.-A. (1990b), *Elephant's Don't Play Chess*, In: Designing Autonomous Agents, edited by Maes, P., The MIT Press, pp. 3-15.

Brooks, R.-A. (1990c), *Lunar Based Construction Robots*, In: IEEE International Workshop on Intelligent Robots and Systems, Tokyo, pp. 389-392.

Brooks, R.-A. (1991a), *Intelligence Without Representation*, Artificial Intelligence, Vol. 47, pp.139-160.

Brooks, R.-A. (1991b), *Intelligence Without Reason*, In: Proceedings of IJCAI-91, Sidney, Australia.

Brooks, R.-A. (1991c), *How to Build Complete Creatures Rather than Isolated Cognitive Simulators*, In: Architectures for Intelligence, edited by VanLehn, K., Laurence Erlbaum Associates, pp. 225-239, Hillsdale, NJ.

Brooks, R.-A. (1991d), *Integrated Systems Based on Behaviours*, SIGART Bulletin, Vol. 2, pp. 46-50.

Brooks, R.-A. (1991e), *Challenges for Complete Creature Architectures*, In: From Animals to Animats, Proceedings of the First International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A. & Wilson, S.-W., MIT Press / Bradford Books.

Brooks, R.-A. (1991f), *Artificial Life and Real Robots*, In: Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, edited by Varela, F. & Bourgine, P., The MIT Press / Bradford Books.

Brooks, R.-A. (1991g), *New Approaches to Robotics*, Science 253, pp. 1227-1232.

Brooks, R.-A. (1993), *Real Robots, Real Learning Problems*, In: Robot Learning, Kluwer Academic Press, pp. 193-213.

Brooks, R.-A. (1994), *Coherent Behavior from Many Adaptive Processes*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 22-29, MIT Press/Bradford Books.

Bryson, J. (1996), *The Design of Learning for an Artifact*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Porgamme, pp. 12-15.

Chapman, D. (1992), *Vision, Instruction and Action*, MIT Press.

Clancey, W. (1995), *A Boy Scout, Toto, and a Bird: How Situated Cognition is Different from Situated Robotics*, In: Artificial Life Route to Artificial Intelligence; Building Embodied, Situated Agents, edited by Steels L. & Brooks, R.-A., LEA, pp. 227-235.

Cliff, D. (1991), *Computational Neuroethology: A Provisional Manifesto*, Cognitive Science Research Paper 162, School of Cognitive and Computing Science, University of Sussex;

Cliff, D., Husbands, P. & Harvey, I. (1992), *Issues in Evolutionary Robotics*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 364-373, MIT Press/Bradford Books.

Cliff, D., Harvey, I. & Husbands, P. (1993a), *Explorations in Evolutionary Robotics*, Adaptive Behaviour, Vol. 2, No. 1, pp. 71-108.

Cliff, D., Husbands, P. & Harvey, I. (1993b), *Evolving Visually Guided Robots*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 374-383.

Cliff, D. & Miller, G.-F. (1996), *Co-evolution of Pursuit and Evasion II: Simulation Methods and Results*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 506-515, MIT Press/Bradford Books.

Colombetti, M & Dorigo, M (1992), *Learning to Control an Autonomous Robot by Distributed Genetic Algorithms*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 305-312, MIT Press/Bradford Books.

Connell, J. (1990), *Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature*, Academic Press.

Corbacho, F.-J. & Arbib, M.-A. (1996), *Learning to Detour & Schema-Based Learning*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., MIT Press/Bradford Books.

Craiger, P. & Coovert, M.-D. (1994), *Modeling Dynamic Social and Psyhological Processes with Fuzzy Cognitive Maps*, IEEE International World Congress on Computational Intelligence, Orlando, FL, pp. 1873-1877.

Cruse, H., Muller-Wilm, U., & Dean, J. (1992), *Artificial Neural Nets for Controlling a 6-Legged Walking System*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 52-60, MIT Press/Bradford Books.

Cruse, H., Bartling, C., Dean, J., Kindermann, T., Schmitz, J., Schumm, M. & Wagner, H. (1996), *Coordination in a Six-Legged Walking System: Simple Solutions to Complex Problems by Exploitation of Physical Properties*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 84-93, MIT Press/Bradford Books.

Darley, V. (1994), *Emergent Phenomena and Complexity*, In: Artificial Life 4, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, edited by Brooks, R.-A. & Maes, P., The MIT Press.

Dellaert, F. & Beer, R.-D. (1996), *A Developmental Model for the Evolution of Complete Autonomous Agents*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 393-401, MIT Press/Bradford Books.

Deugo, D. & Oppacher, F. (1992), *An Evolutionary Approach to Cognition*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 400-409, MIT Press/Bradford Books.

Dickerson, J.-A. & Kosko, B. (1994), *Virtual Worlds as Fuzzy Cognitive Maps*, Presence, Vol. 3, N. 2, Spring, pp. 173-189, MIT.

Dickinson, J. & Dyer, F. (1996), *How Insects Learn about the Sun's Course: Alternative Modeling Approaches*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 193-203, MIT Press/Bradfor Books.

Digney, B.-L. & Gupta, M. (1994), *A Distributed Adaptive Control System for a Quadruped Mobile Robot*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., MIT Press/Bradford Books.

Digney, B.-L. (1996), *Emergent Hierarchical Control Structures: Learning Reactive / Hierarchical Relationships in Reinforcement Environments*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 363-372, MIT Press/Bradford Books.

Donald, B.-R. (1995), *On Information Invariants in Robots*, AI, N. 72 (1-2), pp. 217-304.

Donnart, J.-Y. & Meyer, J.-A. (1994), *A Hierarchical Classifier System Implementing a Motivationally Autonomous Animat*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 144-153, MIT Press/Bradford Books.

Donnart, J.-Y. & Meyer, J.-A. (1996), *Spatial Exploration, Map Learning, and Self-Positioning with MonaLysa*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 204-213, MIT Press/Bradford Books.

Dorigo, M. & Bersini, H. (1994), *A Comparison of Q-Learning and Classifier Systems*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 248-255, MIT Press/Bradford Books.

Drescher, G.-L. (1991), *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*, MIT Press.

Dubois, D. & Prade. H. (1983), *Fuzzy Sets and Systems: Theory and Applications*, Mathematics in Sciences and Engineering, Vol. 144, pp. 220-226, Academic Press.

Duchon, A.-P. (1996), *Maze Navigation Using Optical Flow*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 224-232, MIT Press/Bradford Books.

Edelman, G. (1987), *Neural Darwinism: The Theory of Neuronal Group Selection*, Basic Books, New York.

Ferrell, C. (1993), *Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators*, Technical Report AI-TR-1443, MIT Artificial Intelligence Laboratory.

Ferrell, C. (1996), *Orientation Behaviour Using Registered Topographic Maps*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 94-103, MIT Press/Bradford Books.

Floreano, D. & Mondada, F. (1994), *Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 421-430, MIT Press/Bradford Books.

Floreano, D. & Mondana, F. (1996), *Evolution of Plastic Neurocontrollers for Situated Agents*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 402-410, MIT Press/Bradford Books.

Foner, L.-N. & Maes, P. (1994), *Paying Attention to What is Important: Using Focus of Attention to Improve Unsupervised Learning*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 256-265, MIT Press/Bradford Books.

Foulloy, L. (1993), *Qualitative Control and Fuzzy Control: Towards a Writing Methodology*, AICOM, Vol. 6, No. 3, pp. 147-154.

Gallagher, J.-C. & Beer, R.-D. (1992), *A Qualitative Dynamical Analysis of Evolved Locomotion Controllers*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 71-80, MIT Press/Bradford Books.

Garforth, J., Brammer, K. & Meehan, A. (1997), *A Flexible and Extensible Architecture for Autonomous Robotics*, In: International Workshop on Advanced Robotics and Intelligent Machines, Research Seminars in Robotics and Potential for Exploration, Univeristy of Salford, edited by Gray, J. & Caldwell, D.

Goldberg, D.-E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Ma.

Gonzalez, A. M., Vacher, P., Collingwood, P. & Osborn, R. (1997), *An Obstacle Avoidance Fuzzy Logic Controller*, International Workshop on Advanced Robotics and Intelligent Machines, Research Seminars in Robotics and Potential for Exploration, Univeristy of Salford, edited by Gray, J. & Caldwell, D.

Goode, P. & Chow, M. (1994), *A Hybrid Fuzzy Neural System Used to Extract Heuristic Knowledge from a Fault Detection Problem*, In: Proceedings of the Third IEEE Conference on Fuzzy Systems, Vol. 3, pp. 1731-1736.

Goonatilake, S. & Khebbal S. (1995), *Intelligent Hybrid Systems*, John Wiley & Sons, pp. 107-120, 209-220.

Gorrini, V. & Bersini, H. (1994), *Recurrent Fuzzy Systems*, In: Proceedings of the Third IEEE Conference on Fuzzy Systems, Vo. 1, pp. 193-198.

Gould, J.-L. (1982), *Ethology: The Mechanisms and Evolution of Behaviours*, W. W. Norton & Co. New York.

Green, P.-R. (1994), *How to Watch Your Step: Biological Evidence and An Initial Model*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 55-63, MIT Press/Bradford Books.

Gregory, R.-L. (1966), *Eye and Brain: The Psychology of Seeing*, World University Library, London.

Grossberg, S. (1978), *A Theory of Human Memory: Self-organisation and Performance of Sensory-Motor Codes, Maps, and Plans*, In: Progress in Theoretical Biology, Vol. 5, edited by Rosen, R. & Snell, F., Academic Press, New York.

Grossberg, S. (1980), *How Does a Brain Build a Cognitive Code?*, Psychology Review, Vol. 1, 1.

Grossberg, S. (1981), *Adaptive Resonance in Development, Perception, and Cognition*, In: Mathematical Psychology and Psychopshysiology, edited by Grossberg, S., American Math Society, RI.

Grossberg, S. (1982), *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*, Reidel Press, Boston.

Grossberg, S. (1986), *The Adaptive Brain. I & II*, North Holland, Amsterdam.

Hallam, J. (1995), *Autonomous Robots, A Question of Design?*, In: Artificial Life Route to Artificial Intelligence; Building Embodied, Situated Agents, edited by Steels L. & Brooks, R.-A., LEA, pp. 217-225.

Hallam, J., Ijspeert, A.-J. & Willshaw, D. (1997), *Artificial Lampreys: Comparing Naturally and Artificially Evolved Swimming Controllers*, In: Proceedings of the 4th European Conference on Artificial Life (ECAL97), edited by Husbands, P. & Harvey, I.

Hammond, K.-J., Converse, T.-M. & Grass, J.-W. (1995), *The Estabilization of Environments*, AI, N. 72 (1-2), pp. 305-327

Harvey, Y., Husbands, P. & Cliff, D. (1994), *Seeing the Light: Artificial Evolution, Real Vision*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 392-401, MIT Press/Bradford Books.

Hendler, J. (1988), *Integrating Marker-Passing and Problem Solving, A Spreading Activation Approach to Improved Choice in Planning*, LEA.

Hercock, R.-G. & Barnes, D.-P. (1996), *An Evolved Fuzzy Reactive Control System for Co-operating Autonomous Robots*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 599-607, MIT Press/Bradford Books.

Holland, J.-H. (1985), *Properties of the Bucket Brigade Algorithm*, In: Proceedings International Conference Genetic Algorithms and Their Applications, Pittsburgh, PA, pp. 1-7.

Holland, O. & Melhuish, C. (1996), *Some Adaptive Movements of Animats with Single Symmetrical Sensors*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on the Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Pollack, J. & Wilson, S.-W., MIT Press,

Hopfield, J.-J. (1982), *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, In: Proc. Natl. Acad. Sci. USA, Vol. 79, pp. 2554.

Horswill, I. (1992), *A Simple, Cheap and Robust Visual Navigation System*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 129-136, MIT Press/Bradford Books.

Horswill, I. & Brooks, R.-A. (1988), *Situated Vision in a Dynamic World: Chasing Objects*, AAAI-88, St. Paul, MN, August, pp. 796-800.

Horswill, I. (1995), *Analysis of Adaptation and Environment*, AI, N. 72 (1-2), pp. 1-30.

Horswill, Y. (1996), *Variable Binding and Predicate Representation in a Behavior-Based Architecture*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 163-172, MIT Press/Bradford Books.

Humphrys, M. (1996), *Action Selection Methods Using Reinforcement Learning*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 135-144, MIT Press/Bradford Books.

Iba, H., Garis, H. & Higuchi, T. (1992), *Evolutionary Learning of Predatory Behaviors Based on Structured Classifiers*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp.356-363, MIT Press/Bradford Books.

Juille, H. & Pollack, J.-B. (1996), *Dynamics of Co-evolutionary Learning*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 526-535, MIT Press/Bradford Books.

Kaelbling, L. (1987), *An Architecture for Intelligent Reactive Systems*, Reasoning about Actions_and Plans: Proc., California.

Kaelbling, L. (1988), *Goals and Parallel Program Specifications*, AAAI-88, St. Paul. MN, August, pp. 60-65.

Kaelbling, L. (1995), *The Importance of Being Adaptable*, In: Artificial Life Route to Artificial Intelligence; Building Embodied, Situated Agents, edited by Steels L. & Brooks, R.-A., LEA, pp. 265-275.

Kandel, (1987), *Fuzzy Expert Systems*, Addison Wesley.

Keller, J.-M., Krishnapuram, R. and Rhee, F. (1992), *Evidence Aggregation Networks for Fuzzy Logic Inference*, In: IEEE Transactions on Neural Networks, Vol. 3, No. 5, pp. 761-769.

Kohonen, T. (1982), *Self-Organised Formation of Topologically Correct Feature Maps*, Biological Cybernetics, No. 43, pp. 59-69.

Kortenkamp, D. & Chown, E. (1992), *A Directional Spreading Activation Network for Mobile Robot Navigation*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Kosko, B. (1986a), *Fuzzy Cognitive Maps*, Int. J. Man-Machine Studies, Vol. 24, pp. 65-75.

Kosko, B. (1986b), *Differential Hebbian Learning*, In: Am. Inst. Physics Conf. Proc.: Neural Networks for Computing, edited by Denker, J.-S., Snowbird, Utah. pp. 277.

Kosko, B. (1986c), *Fuzzy Knowledge Combination*, Int. J. Intelligent Systems, Vol. 1.

Kosko, B. (1987), *Fuzzy Associative Memory Systems*, In: Fuzzy Expert Systems, edited by Kandel, Reading MA, Addison Wesley, pp. 135-162.

Kosko, B. (1988), *Hidden Patterns in Combined and Adaptive Knowledge Networks*, International Journal of Approximate Reasoning, Vol. 2, pp. 337-393.

Kosko, B. (1992), *Neural Networks and Fuzzy Systems, A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall.

Kosko, B. (1993), *Fuzzy Thinking*, The New Science of Fuzzy Logic, Hyperion, New York, pp. 224-237,

Kosko, B. (1994), *Virtual Worlds as Fuzzy Cognitive Maps*, Presence, Vol. 3, No. 2, pp. 173-189, Massachusetts.

Koza, J.-R. (1991), *Evolving Emergent Wall-Following Robotic Behaviour Using the Genetic Programming Paradigm*, In: Towards a Practice of Autonomous Systems, Proceeding of the First European Conference on Artificial Life, edited by Varela, F.-J. & Bourgine, MIT Press / Bradford Books, pp. 110-119.

Koza, J.-R. (1992), *Genetic Programming*, MIT Press, Cambridge, MA.

Kube, C.-R. & Zhang, H. (1992), *Collective Robotic Intelligence*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Langston, C. (1990), *Proceedings of Artificial Life II*, Addison-Wesley.

Laszlo, E., Masuli, Y., Artigiani, R. & Csanyi, V. (1993), *The Evolution of Cognitive Maps, New Paradigms for the Twenty-First Century*, The World Features, General Evolution Studies, Vol. 5, Gordon & Breach Science Publishers, Amsterdam.

Lee, C. (1990), *Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part 1*, In: IEEE Transactions on Man, Systems, and Cybernetics, Vol. 20, No. 2, pp. 404-417.

Lee, C. (1990), *Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part 2*, In: IEEE Transactions on Man, Systems, and Cybernetics, Vol. 20, No. 2, pp. 419-435.

Levenick, J.-R. (1991), *NAPS: A Connectionist Implementation of Cognitive Maps*, Connection Science, Vol. 3, No. 2.

Lynch, N.-A. (1993), *Simulation Techniques for Providing Properties of Real-Time Systems*, Technical Report MIT-LCS-TM-949, MIT.

Lyons, D.-M. & Arbib, M.-A. (1989), *A Formal Model of Computation for Sensory-Based Robotics*, In: IEEE Trans. on Robotics and Automation, 5, pp. 280-293.

Maes, P. (1989a), *The Dynamics of Action Selection*, In: IJCAI-89, Dtroit, MI. pp. 991-997.

Maes, P. (1989b), *How to Do the Right Thing*, Connection Science, Vol. 1, No. 3, pp. 291-323

Maes, P. (1990a), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, In: Robotics and Autonomous Systems, Vol. 6, No. 1, North-Holland, also in MIT Press / Bradford Books.

Maes, P. (1990b), *Situated Agents Can Have Goals*, In: Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, edited by Maes, P., MIT Press / Bradford Books.

Maes, P. & Brooks, R.-A. (1990c), *Learning to Coordinate Behaviours*, In: Proceedings of AAAI-90: The American Conference on Artificial Intelligence, Boston, MA, pp. 796-802.

Maes, P. (1991), *Adaptive Action Selection*, In: Thirteenth Annual Conference of the Cognitive Science, Lawrence Erlbaum Associates.

Maes, P. (1992a), *Learning Behaviour Networks from Experience*, In: Towards a Practice of Autonomous Systems, Proceedings in the First European Conference of Artificial Life, edited by Vareal, F.-J. & Bourgine, P., MIT Press/Bradford Books.

Maes, P. (1992b), *Behaviour-Based Artificial Intelligence*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 2-10, MIT Press/Bradford Books.

Maes, P. (1994a), *Modelling Adaptive Autonomous Agents*, Artificial Life Journal, C. Langton, Vol. 1, No. 1&2, pp. 135-162, MIT Press.

Maes, P. (1994b), *Agents that Reduce Work and Information Overload*, Communications of the ACM, Vol. 37, No. 7, pp. 31-40, 146, ACM Press.

Maes, P. (1995), *Intelligent Software Agents*, Scientific American, Vol. 273, No. 3, pp. 84-86.

Mahadevan, S. & Connell, J. (1991), *Automatic Programming of Behaviour-Based Robots Using Reinforcement Learning*, In: Proceedings of the Ninth National Conference on Artificial Intelligence, MIT Press.

Malaka, R., Schmitz, S. & Getz, W.-M. (1996), *A Self-Organising Model of the Antennal Lobes*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on the Simulation of Adaptive Behaviour, edited by Maes, P.

Mamdani, E.-H. (1976), *Advances in Linguistic Synthesis of Fuzzy Logic Controllers*, Int J. Man Mchine Studies, Vol. 8, No. 6.

Mariano, M. & Morasso, P.-G. (1994), *ICANN94: Proceedings of the Fourth International Conference on Artificial Networks*, Vol.1, Springer-Verlang, London.

Mataric, M.-J. (1990a), *A Distributed Model for a Mobile Robot Environment-Learning and Navigation*, Technical Report AI-TR 1228, MIT Artificial Intelligence Laboratory.

Mataric, M.-J. (1990b), *Environment Learning Using a Distributed Representation*, In: Proceedings of 1990 IEEE International Conference on Robotics and Automation, pp. 402-406.

Mataric, M.-J. (1991a), *A Comparative Analysis of Reinforcement Learning Methods*, Technical Report AIM-1322, MIT Artificial Intelligence Lab.

Mataric, M.-J. (1991b), *Behavioural Synergy Without Explicit Representation*, In: Special Issue of SIGART on Integrated Cognitive Architectures, Vol. 2, No. 4.

Mataric, M.-J. (1992a), *Designing Emergent Behaviours: From Local Interactions to Collective Intelligence*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Mataric, M.-J. (1992b), *Behaviour-Based Systems: Key Properties and Implications*, In: IEEE International Conference on Robotics and Automation Workshop on Architectures for Intelligent Control Systems, Nice, France, pp. 46-54.

Mataric, M.-J. (1994a), *Interaction and Intelligent Behaviour*, MIT EECS PhD Thesis, MIT Lab Tech Report AITR-1495.

Mataric, M.-J. (1994b), *Learning to Behave Socially*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 453-462, MIT Press/Bradford Books.

Mataric, M.-J. (1995a), *Integration of Representation Into Goal-Driven Behaviour-Based Robots ?*, In: Artificial Life Route to Artificial Intelligence; Building Embodied, Situated Agents, edited by Steels L. & Brooks, R.-A., LEA, pp. 165-183,

Mataric, M.-J. (1995b), *Designing and Understanding Adaptive Group Behaviour, Adaptive Behaviour*, Vol. 4, No. 1, pp. 51-80.

McCullock, W.-S. & Pitts, W. (1943), *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bull. Math. Biophys., Vol. 5, pp. 115.

McEachern, R.-H. (1993), *Human and Machine Intelligence, An Evolutionary View*, R&E Publishers.

McFarland, D. (1985), *Animal Behaviour*, Benjamin Cummings.

McFarland, D. (1987), *The Oxford Companion to Animal Behaviour*, In: Oxford University Press.

McFarland, D. (1991), *What It Means for Robot Behaviour to be Adaptive*, In: From Animals to Animats; Proceedings of the 1st International Conference of Adaptive Behaviour, edited by Meyer, J.-A. & Wilson, S.-W., MIT Press/Bradford Books.

McFarland, D. (1992), *Animals As Cost-Based Robots*, In: International Studies in the Philosophy of Science, Vol. 6, No. 2, pp. 133-153.

McFarland, D. & Bosser, T. (1993), *Intelligent Behaviour in Animals and Robots*, The MIT Press, Cambridge, Massachusetts.

McFarland, D. (1994), *Towards Robot Cooperation*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 440-444, MIT Press/Bradford Books.

McFarland, D. (1995), *Autonomy and Self-Sufficiency in Robots*, In: Artificial Life Route to Artificial Intelligence; Building Embodied, Situated Agents, edited by Steels L. & Brooks, R.-A., LEA, pp. 187-213.

McNeill, F. M & Thro, E. (1994), *Fuzzy Logic a Practical Approach*, Acamedic Press.

Meyer, J.-A. & Guillot, A. (1994), *From SAB90 to SAB94: Four Years of Animat Research*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 2-11, MIT Press/Bradford Books.

Michel, O. (1996), *Khepera Simulator version 1.0 - User Manual*, University of Nice-Sophia Antipolis, Valbonne, France, (http://wwwi3s.unice.fr/~om/khep-sim.html).

Millan, J.-R. (1994), *Learning Efficient Reactive Behavioural Sequences from Basic Reflexes in a Goal-Directed Autonomous Robot*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 266-274, MIT Press/Bradford Books.

Miller, G.-F. & Cliff, D. (1994), *Protean Behavior in Dynamic Games: Arguments for the Co-Evolution of Pursuit-Evasion Tactics*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 411-420, MIT Press/Bradford Books.

Minsky, M. (1986), *The Society of Mind*, Simon & Schuster, New York.

Moran, F., Moreno, A., Merelo, J. & Chacon, P. (1995), *Advances in Artificial Life*, Proceedings of the Third European Conference on Artificial Life, Lecture Notes in Artificial Intelligence, Subseries of Lecture Notes in Computer Science, Springer-Verlag, Germany.

Morasso, P. & Sanguineti, V. (1994), *Self-Organizing Topographic Maps and Motor Planning*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 214-220, MIT Press/Bradford Books.

Munos, R. & Patinel, J. (1994), *Reinforcement Learning with Dynamic Covering of State-Action Space: Partitioning Q-Learning*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 354-363, MIT Press/Bradford Books.

Narazaki, H. & Ralescu, A.-L. (1992), *A Connectionist Approach for Rule-Based Inference Using an Improved Relaxation Method*, In: IEEE Transactions on Neural Networks, Vol. 3, No. 5, pp. 741-751.

Nepomnyashchikh, V.-A. & Gremyatchikh, V.-A. (1996), *The Experimental Study and Computer Simulation of Fish Behavior in the Uniform Environment*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on

Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 173-179, MIT Press/Bradford Books.

Neven, H., Steinghage, A., Giese, M. & Bruckhoff, C. (1996), *A Dynamics for Vision-Based Autonomous Mobile Robots*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on the Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Pollack, J. & Wilson, S.-W., MIT Press/Bradford Books.

Newell, A. (1982), *The Knowledge Level*, Artificial Intelligence, Vol. 18, pp.87-127.

Noble, J. & Cliff, D. (1996), *On Simulating the Evolution of Communication*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 608-617, MIT Press/Bradford Books.

Nwana, H.-S., Azarmi, N. & Smith, R. (1996a), *The Rise of Machine Intelligence*, In: BT Technological Journal, Vol. 14, No. 4, pp. 9-13, Chapman & Hall.

Nwana, H.-S. & Ndumu, D.-T. (1996b), *An Introduction to Agent Technology*, In: BT Technological Journal, Vol. 14, No. 4, pp. 55-67, Chapman & Hall.

Nwana, H.-S. & Wooldridge, M. (1996c), *Software Agent Techonologies*, In: BT Technological Journal, Vol. 14, No. 4, pp. 68-78, Chapman & Hall.

Pappis, C. & Mamdami, E. (1977), *A Fuzzy Logic Controller for a Traffic Junction*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-7, No. 10, pp. 707-717.

Parker, L.-E. (1992), *Adaptive Action Selection for Cooperative Agent Teams*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Peng, J. & Williams, R.-J. (1992), *Efficient Learning and Planning within the Dyna Framework*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 281-290, MIT Press/Bradford Books.

Perkins, S. & Hayes, G. (1996), *Robot Shaping-Principles and Architectures*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Porgamme, pp. 36-45.

Pfeifer, R. & Verschure, P. (1992), *Designing Efficiently Navigating Non-Goal-Directed Robots*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 31-39, MIT Press/Bradford Books.

Pfeifer, R. & Verschure, P. (1995), *The Challenge of Autonomous Agents: Pitfalls and How to Avoid Them*, In: Artificial Life Route to Artificial Intelligence; Building Embodied, Situated Agents, edited by Steels L. & Brooks, R.-A., LEA, pp. 237-263.

Pfeifer, R. (1996a), *Building 'Fungus Eaters': Desing Principles of Autonomous Agents*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 3-12, MIT Press/Bradford Books.

Pfeifer, R. (1996b), *Symbols, Patterns, and Behaviour: Towards a New Understanding of Intelligence*, In: Proceedings Japanese Conference on Artificial Intelligence, pp. 1-15.

Pfeifer, R. (1997), *Teaching Powerful Ideas with Autonomous Mobile Robots*, In: Journal of Computer Science Education, Vol. 7, No. 2.

Piaget, J. (1962), *Play, Dreams and Imitation in Children*, W. W. Norton & Co., New York.

Pipe, A.-G., Fogarty, T.-C. & Winfield, A. (1994), *A Hybrid Architecture for Learning Continuous Environmental Models in Maze Problems*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 198-205, MIT Press/Bradford Books.

Pipe, A.-G. & Winfield, A. (1996a), *An Autonomous System for Extracting Fuzzy Behavioral Rules in Mobile Robotics*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 233-243, MIT Press/Bradford Books.

Pipe, A.-G., Fogarty, T.-C. & Winfield, A. (1996b), *Abstracting Fuzzy Behavioural Rules from Geometric World Models in Mobile Robotics*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Porgamme, pp. 56-65.

Prescott, T.-J. (1993), *Explorations in Reinforcement and Model-Based Learning*, PhD Thesis, University of Sheffield.

Prescott, T.-J. (1994), *Spatial Learning and Representation in Animats*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 164-173, MIT Press/Bradford Books.

Prescott, T.-J. & Redgrave, P. (1996a), *Layered Control Architectures in Natural and Artificial Systems*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Programme, pp. 46-55.

Prescott, T.-J. (1996b), *Robot Spatial Learning: Insights From Animal and Human Behaviour*, IEEE Workshop on Self Learning Robots.

Prescott, T.-J. (1997), *The Early Evolution of Spatial Behaviour: Robot Models of Trace Fossils*, In: Proceedings AISB Workshop on Spatial Reasoning in Mobile Robots and Animals, Manchester, Tech. Rep. Series, Dept. of Computer Science, Man. Univ. ISSN 1361-6161. No. UMCS 97-4-1.

Resnick, M. (1992), *Beyond the Centralized Mindset; Exploration in Massively Parallel Microworlds*, PHD Thesis, MIT.

Reynolds, C.-W. (1994), *Evolution of Corridor Following Behaviour in a Noisy World*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp 402-410, MIT Press/Bradford Books.

Ring, M. (1992), *Two Methods for Hierarchy Learning in Reinforcement Environments*, In: From Animals to Animats 2; Proceedings of the 2rd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Roitblat, H.-L. (1982), *The Meaning of Representation in Animal Memory, Behavioural and Brain Science*, Vol. 5, pp 353-406.

Roitblat, H.-L. (1994), *Mechanism and Process in Animal Behaviour: Models of Animals and Animals as Models*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp 12-21, MIT Press/Bradford Books.

Rosenblant, J. & Payton, D. (1989), *A Fine Grained Alternative to the Subsumption Architecture for a Mobile Robot* Control, In: Proceedings of the International Loint Conference on Neural Networks, IJCNN, Washington DC.

Rowe, D.-G. (1998), *March of the Biobots*, New Scientist, http://www.newscientist.com.

Rutkowska, J.-C. (1994), *Emergent Functionality in Human Infants*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 179-188, MIT Press/Bradford Books.

Saffiotti, A. (1997), *The Uses of Fuzzy Logic in Autonomous Robot Navigation*, In: Soft Computing, Vol.1, Num. 4, pp. 180, 197, Springer.

Saunders, G.-M., Kolen, J.-F. & Pollack, J.-B. (1994), *The Importance of Leaky Levels for Behaviour-Based AI*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 275-281, MIT Press/Bradford Books.

Saunders, G.-M. & Pollack, J.-B. (1996), *The Evolution of Communication Schemes over Continuos Channels*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 580-589, MIT Press/Bradford Books.

Scutt, T. (1994), *The Five Neuron Trick: Using Classical Conditioning to Learn How to Seek the Light*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 364-370, MIT Press/Bradford Books.

Schelcher, C. & Lambrinos, D. (1996), *Categorization in a Real-World Agent Using Haptic Exploration and Active Perception*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 65-74, MIT Press/Bradford Books.

Schmajuk, N.-A. & Blair, H.-T. (1992), *Dynamics of Spatial Navigation: An Adaptive Neural Network*, In: From Animals to Animats 2; Proceedings of the 2rd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Schmajuk, N.-A. (1994), *Behavioural Dynamics of Escape and Avoidance: A Neural Network Approach*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 118-127, MIT Press/Bradford Books.

Scheider, M. & Mataric, M.-J. (1996), *A Study of Territoriality: The Role of Critical Mass in Adaptive Task Division*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 553-561, MIT Press/Bradford Books.

Sholl, M.-J. (1987), *Cognitive Maps as Orienting Schemata*, Journal of Experimental Psychology: Learning, Memory and Cognition, Vol. 13, No. 4, pp. 615-628.

Smart, W.-D. & Hallam, J. (1994), *Location Recognition in Rats and Robots*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 174-178, MIT Press/Bradford Books.

Smith, R. & Mamdani, E.-H. (1996), *Intelligent Software Technologies*, In: BT Technological Journal, Vol. 14, No. 4, pp. 22-31, Chapman & Hall.

Smithers, T. (1992), *Taking Eliminative Materialism Seriously: A Methodology for Autonomous Systems Research*, In: Towards a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life, edited by Varela, F.-J. & Bourgine, P., MIT Press/Bradford Books, Cambridge Ma, pp. 31-40.

Smithers, T. (1994), *On Why Better Robots Make it Harder*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 64-72, MIT Press/Bradford Books.

Smithers, T. (1995), *Are Autonomous Agents Information Processing Systems?*, In: Artificial Life Route to AI; Building Situated Emboided Agents, edited by Steels, L. & Brooks, R.-A., LEA, pp. 138-155.

Spector, L. & Stoffel, K. (1996), *Automatic Generation of Adaptive Programs*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 476-483, MIT Press/Bradford Books.

Spier, E. & McFarland, D. (1996), *A Finner-Grained Motivational Model of Behaviour Sequencing*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 225-263, MIT Press/Bradford Books.

Steels, L. (1989), *Co-operating Between Distributed Agents Through Self-Organisation*, In: Workshop in Multi-Agent Co-operation, North-Holland, Cambridge, UK.

Steels, L. (1991), *Towards a Theory of Emergent Functionality*, In: From Animals to Animats; Proceedings of the 1st International Conference of Adaptive Behaviour, edited by Meyer, J.-A. & Wilson, S.-W., MIT Press/Bradford Books.

Steels, L. (1993a), *Building Agents with Autonomous Behaviour Systems*, In: The Artificial Life Route to AI, edited by Steels, L. & Brooks, R.

Steels, L. & Vertommen (1993b), *Emergent Behaviour. A Case Study for Wall Following*, VUB AI Lab Memo, Brussels.

Steels, L. (1994a), *The Artificial Life Roots of Artificial Intelligence*, Artificial Life, Vol. 1, No. 1, pp-75-100, MIT Press, Cambridge.

Steels, L. (1994b), *A Case Study in the Behaviour-Oriented Design of Autonomous Agents*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 445-452, MIT Press/Bradford Books.

Steels, L. (1995), *Building Agents Out Of Autonomous Behaviour Systems*, In: Artificial Life Route to AI; Building Situated Emboided Agents, edited by Steels, L. & Brooks, R.-A., LEA, pp. 83-119.

Steels, L. (1996), *Emergent Adaptive Lexicons*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 562-567, MIT Press/Bradford Books.

Taber, R. (1991), *Knowledge Processing with Fuzzy Congnitive Maps*, Expert Systems With Applications, Vol. 2, pp. 83-87.

Takagi, H., Suzuki, N. Koda, T. & Kojima, Y. (1992), *Neural Networks Design on Approximate Reasoning Architecture and Their Applications*, In: IEEE Transactions on Neural Networks, Vol. 3, No. 5, pp. 752-760.

Tani, J. (1996a), *Model-Based Learning for Mobile Robot Navigation from Dynamical System Perspective*, In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. 26, No. 3, special issue on Learning Autonomous Robotics.

Tani, J. (1996b), *Does Dynamics Solve the Symbol Grounding Problem of Robots? An Experiment in Navigation Learning*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Programme, pp. 66-74.

Thorton, C. (1996), *Brave Robots Use Representation*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Programme, pp. 94-103.

Titmuss, R., Crabtree, I.-B & Winter, C.-S. (1996), *Agents, Mobility and Multimedia Information*, In: BT Technological Journal, Vol. 14, No. 4, pp. 141-148, Chapman & Hall.

Toates, F. & Jensen, P. (1991), *Ethological and Psychological Models of Motivation Towards a Synthesis*, In: From Animals to Animats; Proceedings of the 1st International Conference of Adaptive Behaviour, edited by Meyer, J.-A. & Wilson, S.-W., MIT Press/Bradford Books.

Toates, F. (1994), *What is Cognitive and What is Not Cognitive?*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 102-107, MIT Press/Bradford Books.

Todd, P.-M. & Wilson, S.-W. (1992), *Environment Structure and Adaptive Behaviour From the Group Up*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 11-20, MIT Press/Bradford Books.

Todd, P.-M., Wilson, S.-W., Somayaji, A.-B. & Yanco, H.-A. (1994), *The Blind Breeding the Blind: Adaptive Behaviour Without Looking*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 228-237, MIT Press/Bradford Books.

Tolman, E. (1984), *Cognitive Maps in Rats and Men*, Psychological Review, Vol. 55, pp. 189-208.

Tsuji, S. & Li, S. (1992), *Memorizing and Representing Route Scenes*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., pp. 225-232, MIT Press/Bradford Books.

Tyrrell, T. (1992), *The Use of Hierarchies for Action Selection*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Tyrrell, T. (1993), *Computational Mechanisms for Action Selection*, PhD Thesis, University of Edinburgh.

Tyrrell, T. (1994), *An Evaluation of Maes' Bottom-Up Mechanism for Behaviour Selection*, Journal of Adaptive Behaviour, MIT Press, Massachusetts.

Varela, F.-J. & Bourgine (1992), *Toward a Practice of Autonomous Systems*, Proceedings of the First Conference on Artificial Life, MIT Press / Bradford Books, Cambridge Ma.

Verschure, P. & Pfeifer, R. (1992), *Categorization, Representations and, The Dynamics of System-Environment Interaction: A Case Study in Autonomous Systems*, From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Wang, P.-Z. & Loe, K.-F. (1993), *Between Mind and Computer, Fuzzy Science and Engineering*, Advances in Fuzzy Systems-Applications and Theory, Vol. 1, pp. 35-105, 130-177.

Watt, S.-N.-K (1996), *Artificial Societies and Psychological Agents*, In: BT Technological Journal, Vol. 14, No. 4, pp. 89-97, Chapman & Hall.

Webb, B. (1994), *Robotic Experiments in Cricket Phonotaxis*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 45-54, MIT Press/Bradford Books.

Webb, B. & Hallam, J. (1996), *How to Attract Femaels: Further Robotic Experiments in Cricket Phonotaxis*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 75-83, MIT Press/Bradford Books.

Weiss, G. (1992), *Action Selection and Learning in Multi-Agent Environments*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Werger, B.-B. & Matarlc, M.-J. (1996), *Robotic Food Chains: Externalization of State and Program for Minimal-Agent Foraging*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 625-634, MIT Press/Bradford Books.

Werner, G.-M. & Dyer, M.-G. (1992), *Evolution of Herding Behaviour in Artificial Animals*, In: From Animals to Animats 2; Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour, edited by Meyer, J.-A., Roitblat, H.-L. & Wilson, S.-W., MIT Press/Bradford Books.

Werner, G.-M. (1994), *Using Secondary Order Neural Connections for Motivation of Behavioural Choices*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 154-162, MIT Press/Bradford Books.

Wiener, N. (1948), *Cybernetics or Control and Communication in the Animal and the Machine*, MIT Press, USA.

Wilde, P. (1996), *Nonlinear Dynamics of Two Types of Networks with Intelligent Nodes*, In: BT Technological Journal, Vol. 14, No. 4, pp. 98-104, Chapman & Hall.

Williamson, M.-M. (1996), *Postural Primitives: Interactive Behaviour for a Humanoid Robot Arm*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 124-134, MIT Press/Bradford Books.

Wilson, S.-W. (1985), *Knowledge Growth in an Artificial Animal*, In: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, edited by Greffenstette, Lawrence Eribum Associates.

Wilson, S.-W. (1991), *The Animat Path to AI*, In: From Animals to Animats; Proceedings of the 1st International Conference of Adaptive Behaviour, edited by Meyer, J.-A. & Wilson, S.-W., MIT Press/Bradford Books.

Wilson, S.-W. (1996), *Explore / Exploit Strategies in Autonomy*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Willson, S.-W., pp. 325-332, MIT Press/Bradford Books.

Yamauchi, B. & Beer, R. (1994), *Integrating Reactive, Sequential, and Learning Behavior Using Dynamical Neural Networks*, In: From Animals to Animats 3; Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour, edited by Cliff, D., Husbands, P., Meyer, J.-A. & Wilson, S.-W., pp. 382-391, MIT Press/Bradford Books.

Zadeh, L.-A. (1965), *Fuzzy Sets*, Information and Control, Vol. 8, pp. 338-353.

Zadeh, L.-A. (1972a), *A Rationale For Fuzzy Control*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 123-126, World Scientific, 1996.

Zadeh, L.-A. (1972b), *On Fuzzy Algorithms*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 127-147, World Scientific, 1996.

Zadeh, L.-A. (1972c), *Fuzzy Languages and Their Relation to Human and Machine Intelligence*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 148-179, World Scientific, 1996.

Zadeh, L.-A. (1972d), *On Fuzzy Mapping and Control*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 180-184, World Scientific, 1996.

Zadeh, L.-A. (1973a), *A System Theoretic View of Behaviour-Modification*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, pp. 185-194, World Scientific, 1996.

Zadeh, L.-A. (1973b), *The Concept of a Linguistic Variable and Its Application to Approximate Reasoning*, Memorandum M-411, Electronics Research Laboratory, University of California, Berkeley.

Zadeh, L.-A. (1974), *On The Analysis of Large-Scale Systems*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 196-209, World Scientific, 1996.

Zadeh, L.-A. (1975), *Calculus of Fuzzy Restrictions*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 210-237, World Scientific, 1996.

Zadeh, L.-A. (1983), *The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems*, Fuzzy Sets and Systems, Vol. 11, pp. 199.

Zadeh, L.-A. (1989), *Knowledge Representation in Fuzzy Logic*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 764-774, World Scientific, 1996.

Zadeh L.-A. & Kacprzyk, J. (1992), *Fuzzy Logic for the Management of Uncertainty*, Inc, Wisly & Sons, pp. 281-347.

Zadeh, L.-A. (1994), *Soft Computing and Fuzzy Logic*, In: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh, Advances in Fuzzy Systems - Applications and Theory, Vol. 6, edited by Klir, G.-J. & Yuan, B, pp. 796, World Scientific, 1996.

Zadeh, L.-A. (1996), *The Roles of Fuzzy Logic and Soft Computing in the Conception and Deployment of Intelligent Systems*, In: BT Technological Journal, Vol. 14, No. 4, pp. 32-36, Chapman & Hall.

Ziemke, T. (1996a), *Towards Adaptive Behaviour System Integration Using Connectionist Infinite State Automata*, In: From Animals to Animats 4; Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour, edited by Maes, P., Mataric, M.-J., Meyer, J.-A., Pollack, J. & Wilson, S.-W., pp. 145-154, MIT Press/Bradford Books.

Ziemke, T. (1996b), *Towards Autonomous Robot Control Using Connectionist Infinite State Automata*, Learning in Robots and Animals Working Notes, University of Sussex, AISB96 Workshop and Tutorial Porgamme, pp. 121-130.