THE UNIVERSITY OF HULL

Robo-CAMAL:
Anchoring in a Cognitive Robot

being a thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
in the University of Hull
by

James Gwatkin

December 20, 2009

Contents

1 Introduction **1**

1.1 Introduction 1

1.2 Objectives and Context 2

1.2.1 The Anchoring Problem 3

1.2.2 Situated and Embodied Cognition 3

1.2.3 Machine Learning 4

1.3 Robo-CAMAL Components 4

1.3.1 Reactive Component 5

1.3.2 Deliberative Component 6

1.4 Summary 11

2 Cognitive Architectures **13**

2.1 Introduction 13

2.2 Cognitive Science 13

2.3 Symbolic A.I. 15

2.4 Connectionism 19

2.5 Reactive Architecture 22

2.6 Hybrid Architecture 26

2.7 Summary 29

3 The Robo-CAMAL Framework **30**

3.1 Introduction 30

3.2 Mind as a Control System 30

3.3 CogAff Architecture 32
3.4 Belief Desire Intention Model 34
3.5 CRIBB and a-CRIBB 35
3.6 Blackboard Systems 41
3.7 Motivational Control Process 42
3.8 CAMAL Architecture 44
3.9 Summary 45

4 Robot Architecture Overview 46
4.1 Introduction 46
4.2 Architecture Platform 46
4.2.1 The Robot 46
4.2.2 The Environment 48
4.3 The Architecture 49
4.4 Objectives and Problems 53
4.5 Summary 53

5 Vision System 55
5.1 Introduction 55
5.2 Omnidirectional Vision 56
5.3 Basic Image Segmentation 57
5.4 Detecting Objects Within Robo-CAMAL 59
5.5 Node Placement Algorithm Limitations 63
5.6 Object Detection 65
5.7 Summary 70

6 Reactive Robo-CAMAL 72
6.1 Introduction 72
6.2 The Client/Server System 73
6.3 Sensor System 74
6.4 Micro-Behaviours 76

6.5 Arbitration Methods 78
6.5.1 Priority Method 78
6.5.2 Aggregate Method 78
6.5.3 Winner Method 79
6.5.4 Behaviour Suppression Method 80
6.6 Macro-Behaviours 80
6.7 Deliberative Interface 81
6.8 Reactive Robo-CAMAL Overview 84
6.9 Summary 86

7 The Domain Model 87
7.1 Introduction 87
7.2 The Domain Model 87
7.2.1 Environment Domain Model 88
7.2.2 Physical Form Domain Model 91
7.3 Situated and Embodied Cognition: A Domain Model Perspective 93
7.3.1 The Situated and Embodied Approach 93
7.3.2 The Frame Problem 94
7.3.3 The Symbol Grounding Problem 98
7.3.4 The Anchoring Problem 100
7.4 Summary 105

8 Associations 106
8.1 Introduction 106
8.2 Association Construct 106
8.3 Association Value 108
8.4 Association Creation 110
8.5 Machine Learning Through Associations 113
8.5.1 Machine Learning 113
8.5.2 Reinforcement Learning 114

8.5.3 Learning Using Associations 117
8.6 Adaptation Using Associations 119
8.7 Summary 124

9 Motivation Within Robo-CAMAL 125
9.1 Introduction 125
9.2 Motivation 125
9.3 Grounding Motivation 127
9.4 Implementing Motivation 129
9.5 Motivational Control States 131
9.6 Motivational Blackboard 134
9.6.1 The Blackboard 135
9.6.2 The Knowledge Sources 139
9.6.3 The Control Component 142
9.7 Summary 143

10 Experimental Results 144
10.1 Introduction 144
10.2 Control Experiments 145
10.2.1 The *hit(Object)* Control Experiment 145
10.2.2 The *avoid(Object)* Control Experiment 147
10.2.3 Association Generation Control Experiment . . 150
10.2.4 Goal Selection Control Experiment 152
10.3 Learning Experiments 154
10.3.1 First Learning Experiment 154
10.3.2 Second Learning Experiment 166
10.4 Adaptation Experiments 171
10.5 Conclusion 184

11 Analysis and Discussion 187
11.1 Introduction 187

11.2 Robo-CAMAL's Anchoring Mechanism 188
11.2.1 Anchoring Objects with Robo-CAMAL 188
11.2.2 Anchoring Events with Robo-CAMAL 191
11.3 Robo-CAMAL's Learning Mechanism 192
11.3.1 Reinforcement Learning 194
11.4 Adaptation with robo-CAMAL 197
11.4.1 The Domain Model and Adaptation 198
11.5 Grounding Motivation with Robo-CAMAL 200
11.6 Improving Robo-CAMAL 202
11.6.1 Improving the Domain Model 202
11.6.2 Meta-Cognitive Layer 208
11.7 Summary 211

12 Summary **214**
12.1 Introduction 214
12.2 Background Material 215
12.3 The Robo-CAMAL Architecture 216
12.4 Experiments and Results 219
12.5 Conclusion 221

A The Motivational Blackboard **236**
A.1 Introduction 236
A.2 The Blackboard 236
A.3 The Control Component 242
A.4 The Knowledge Sources 244

B The Domain Model **249**
B.1 Introduction 249
B.2 Domain Model Description 249

C Results **251**

C.1 Introduction 251

C.2 First Learning Experimental Results 252

C.2.1 *find(blackrobot)* 252

C.2.2 *find(redrobot)* 262

C.2.3 *find(blueball)* 271

C.3 Second Learning Experimental Results 280

C.3.1 Zero Objects, Static Environment 280

C.3.2 One Object, Static Environment 287

C.3.3 One Object, Dynamic Environment 294

C.3.4 Two Objects, Static Environment 302

C.3.5 Two Objects, Dynamic Environment 309

C.3.6 Three Objects, Static Environment 315

C.3.7 Three Objects, Dynamic Environment 322

C.3.8 Four Objects, Static Environment 329

C.3.9 Four Objects, Dynamic Environment 336

C.4 Adaptation Results 343

C.4.1 Adaptation Experiment 1 343

C.4.2 Adaptation Experiment 2 347

C.4.3 Adaptation Experiment 3 351

C.4.4 Adaptation Experiment 4 355

C.4.5 Adaptation Experiment 5 359

C.4.6 Adaptation Experiment 6 363

List of Figures

2.1 A neuron. 19
2.2 Threshold logic unit. 20
2.3 Light sensitive agents (Braitenberg, 1984). 23
2.4 Decomposition by function. 23
2.5 Decomposition by action. 24
2.6 Simple reactive architecture. 25
2.7 The BB1 agent architecture (Hayes-Roth, 1995). 28

3.1 CogAff architecture (Sloman, 2001). 33
3.2 Belief-Desire reasoning scheme (Bartsch and Wellman, 1989). 35
3.3 CRIBB architecture (Wahl and Spada, 2000). 36
3.4 Blackboard system (Corkill, 1991). 41
3.5 Motivational control states (Davis, 2001). 43
3.6 CAMAL architecture (Davis, 2004). 44

4.1 The Amigobot (Activemedia, 2000). 47
4.2 Amigobot with onmidirectional vision system added. . 48
4.3 The robot's environment. 48
4.4 The robo-CAMAL architecture. 49

5.1 Omnidirectional vision system. 56
5.2 Image from robo-CAMALs' vision system. 57

5.3 Extraction of an area of interest from robo-CAMAL image. 60
5.4 Edge image generated using Sobel operator on figure 5.3. 61
5.5 Node chain generation using 5.4. 62
5.6 Node chain generation using a proximity threshold. . . 62
5.7 Node chain generated using only gradient threshold. . . 62
5.8 Node chain generated using low, good and high edge threshold respectively. 63
5.9 Mug obscuring book on right hand side. 64
5.10 Effect of a patterned background on edge detection algorithm. 64
5.11 Effect of red colour profile mask on robo-CAMAL image. 66
5.12 Effect of black colour profile mask on robo-CAMAL image. 67
5.13 Effect of blue colour profile mask on robo-CAMAL image. 68
5.14 The Object-AOI as defined by 5.7. 69
5.15 The AOI of an image that is used to extract the Object-AOI. 69

6.1 The client/server structure for the amigobot. 73
6.2 Omnidirectional image processing. 75
6.3 Behaviour weighting functions. 79
6.4 Reactive robo-CAMAL. 85

7.1 Two fuzzy sets representing red and small. 101
7.2 Two methods to match fuzzy sets A and B. 102

8.1 Association value functions. 109
8.2 Association generation process. 112
8.3 Association rank value. 121
8.4 Goal importance value. 122

9.1 Hybrid agent architecture (Stoytchev and Arkin, 2004). 130
9.2 Proposed motivational control states (Davis, 2007). . . 132

10.1 The *hit(redrobot)* control experiment. 145
10.2 The *hit(blueball)* control experiment. 146
10.3 The *hit(blackrobot)* control experiment. 146
10.4 Mean collisions vs number of objects. 148
10.5 Mean collisions vs number of objects. 148
10.6 Mean collisions vs number of objects. 148
10.7 Mean collisions vs number of dynamic objects. 149
10.8 Mean collisions vs number of dynamic objects. 149
10.9 Mean collisions vs number of dynamic objects. 149
10.10Association generation process. 151
10.11*hit(object)* control experiment. 153
10.12Associations in the *hit(redrobot)* learning experiment. . 156
10.13Associations in the *find(blueball)* learning experiment. . 159
10.14Associations in the *track(redrobot)* learning experiment. 164
10.15Associations in the *find(blueball)* learning experiment. . 167
10.16Environment 1 static object, association belief *environment(sparse)*. 170
10.17Environment 2 static objects, association belief *environment(cluttered)*. 170
10.18Environment 3 dynamic objects, association belief *environment(dynamic)*. 171
10.19Periods when the belief *found(X)* was present. 172
10.20Periods when the belief *hit(X)* was present. 174
10.21Goal importance value. 176
10.22*hit(blackrobot)* association value. 177
10.23*hit(blueball)* association value. 177
10.24*hit(redrobot)* association value. 178

10.25Periods when the belief *found(X)* was present. 179
10.26Periods when the belief *hit(X)* was present. 180
10.27Goal importance value. 181
10.28*hit(blackrobot)* association value. 182
10.29*hit(blueball)* association value. 182
10.30*hit(redrobot)* association value. 183

11.1 Goal hierarchy. 196

List of Tables

2.1 Event calculus predicates (Shanahan, 1999). 27

5.1 The calculated colour profile for the three objects. . . . 65

6.1 Macro-Behaviours. 81
6.2 Deliberative message rule table. 83

7.1 Object profiles. 88

10.1 The possible environment combinations. 155
10.2 Total number of correct and incorrect associations found. 160
10.3 Total number of correct associations found in 10 minutes. 162
10.4 Find learning experiment results. 162
10.5 Hit learning experiment results. 163
10.6 Track learning experiment results. 163
10.7 *redrobot* learning experiment results. 165
10.8 *blackrobot* learning experiment results. 165
10.9 *blueball* learning experiment results. 165
10.10 Success count for each reactive architecture. 169
10.11 Success count for each arbitration method. 171
10.12 Correct and incorrect hit belief. 184

B.1 Domain model components. 250

Abstract

The CAMAL architecture (Computational Architectures for Motivation, Affect and Learning) provides an excellent framework within which to explore and investigate issues relevant to cognitive science and artificial intelligence. This thesis describes a small sub element of the CAMAL architecture that has been implemented on a mobile robot. The first area of investigation within this research relates to the anchoring problem. Can the robotic agent generate symbols based on responses within its perceptual systems and can it reason about its environment based on those symbols? Given that the agent can identify changes within its environment, can it then adapt its behaviour and alter its goals to mirror the change in its environment? The second area of interest involves agent learning. The agent has a domain model that details its goals, the actions it can perform and some of the possible environmental states it may encounter. The agent is not provided with the belief-goal-action combinations in order to achieve its goals. The agent is also unaware of the effect its actions have upon its environment. Can the agent experiment with its behaviour to generate its own belief-goal-action combinations that allow it to achieve its goals? A second related problem involves the case where the belief-goal-action combination is pre-programmed. This is when the agent is provided with several different methods with which to achieve a specific goal. Can the agent learn which combination is the best? This thesis will describe the sub-element of the CAMAL architecture that was developed for a robot (robo-CAMAL). It will also demonstrate how robo-CAMAL solves the anchoring problem, and learns how to act and adapt in its environment.

Chapter 1

Introduction

1.1 Introduction

Mobile robots provide an essential tool when investigating the interaction of cognitive architectures and the physical environment. Robots have been used to investigate many different aspects of artificial intelligence such as mapping and localization techniques (Stachniss and Burgard, 2005; Wolf and Sukhatme, 2005), robot perception and navigation (Fiala and Basu, 2004; Marques et al., 2002) and robot learning (Hougen et al., 1996). This thesis seeks to use a mobile robot to investigate a specific area of cognitive science known as the anchoring problem. The anchoring problem describes the problem of generating and maintaining links between symbols and perceptual data (see section 1.2.1).

This thesis will attempt to achieve two main goals. The first is to develop a robotic agent that can learn how to achieve its goals with no understanding of the effect of its actions. To do this the agent must be able to identify the focus of its goal, i.e. if the goal is to eat, it must recognise when an object is edible. The agent must also be able to recognise when the objective of its goal has been achieved.

The second goal is to develop a robotic agent that can use the learned actions from the first goal in order to adapt its behaviour

within a variable environment. If the agent can achieve these two goals it will have demonstrated several things. The first is that the agent can link, or anchor, perceptual data to objects representing the focus of its goals. The second is that the agent can anchor symbols representing events and actions to perceived changes within its environment.

To achieve these goals a hybrid reactive/deliberative architecture has been implemented upon a mobile robot (see section 1.3). The architecture consists of several different elements including a reactive component, consisting of many different reactive behaviours; a motivational blackboard; a belief-desire-intention (BDI) schema; a distributed model of affect; an association construct and a domain model. All these components implemented on a robotic platform combine to make robo-CAMAL.

The remainder of this chapter will consider some of the issues that need to be addressed in answering the questions under investigation. It will then briefly introduce the main components of the robo-CAMAL architecture. Finally it will provide an outline for the rest of this thesis.

1.2 Objectives and Context

This thesis presents a cognitive architecture that attempts to provide a positive answer to the two following questions. The first is can the agent developed here modify its goals and behaviour in response to changes in its environment?

The second question is can the agent learn which actions to use to achieve its goals? The agent has a set of beliefs it can hold, actions it can perform, and goals it wishes to achieve. To achieve its goal the agent needs to instigate the correct action based on its current belief. Given that the agent is given no explicit knowledge of the

correct belief-goal-action combination, can it determine the correct combination on its own? A related point is the case where the agent is provided with several possible belief-goal-action combinations that achieve a specific goal. Can the agent determine which is the best combination? For a robotic agent to provide a positive answer to these questions its architecture needs to address several key issues within cognitive science and robotics.

1.2.1 The Anchoring Problem

The symbol grounding problem concerns the difficulties of generating symbols using perceptual systems, and the meaning of those symbols (Harnad, 1990). The anchoring problem is a subset of the grounding problem. It investigates how links are generated and maintained between symbols used within an agent's cognitive architecture, and the data obtained via the agents perceptual system (Coradeschi and Saffiotti, 2003). In the past within robotics, this linking of symbols to perceptual data has been buried in the code of the agent's architecture. Recently there has been a push to formalise the problem in order to identify the difficulties in linking symbols to objects, and to either separate the anchoring process from the rest of the architecture, or to specify when and where anchoring occurs. This approach provides insight into the specific problems that surround the anchoring process. If an agent is to reason about symbolic representations of its environment, it must be able to perceive its environment. It must also be able to link those perceptions to the relevant symbols.

1.2.2 Situated and Embodied Cognition

Situated and embodied cognition refers to the role the environment plays in the development of cognitive processes within the agent (Pfeifer

and Scheier, 1999; Clark, 1998; Clancey, 1997). The cognitive processes of an agent that is situated (is present within its environment) are determined to a large extent by its environment. The cognitive processes of an agent that is embodied (has a physical body within its environment) are determined by its interactions with its environment. In other words, cognitive processes develop from real-time, goal-directed interactions between the agent and its environment (Thelen et al., 2001). From this viewpoint the agent can learn to achieve its goals by interacting with its environment. If this is the case then information about its environment and its physical body must be incorporated into the agent's architecture.

1.2.3 Machine Learning

One of the questions posed in section 1.1 is whether the agent can learn the correct behaviour required to achieve its goal. It is therefore clear that the agent needs some form of learning mechanism (Alpaydin, 2004). There are various different mechanisms possible (see section 8.5.1) but the one implemented here uses a reinforcement learning technique (Sutton and Barto, 1998). This is where an agent learns by interacting with, and receiving feedback from, its environment.

1.3 Robo-CAMAL Components

The cognitive architecture developed here is a combination of a reactive and a deliberative component, implemented on a mobile robot and a desktop computer. Perceptual data and control commands are passed between the robot and the computer via a radio modem and a USB cable (see section 4.2.1). This section will briefly introduce some of the main components of the architecture.

1.3.1 Reactive Component

A reactive robot is one where the perceptual input is directly connected to the motor output (Braitenberg, 1984; Brooks, 1991). There can be various different definitions of what constitutes a reactive component. For example the system may have no changeable internal state so that the current input determines the current output. In this case the output is always the same given the same input. The definition of a reactive system taken here is that the systems output is determined not only by its input, but also by its internal state. This is akin to a simple finite state machine. The system's output and new state is based on its input and its current state.

1.3.1.1 Reactive Behaviours

The reactive component consists of a number of several different reactive behaviours. These behaviours are modelled using software written on the desktop computer, and the robot circuit board, as opposed to being hard-wired into the robot. The lowest level consists of simple micro-behaviours that turn the robot left, or move it forward etc. These micro-behaviors are programed directly on the robot. The micro-behaviours are combined to generate task specific macro-behaviours e.g. hit a specific object, or avoid objects etc. The micro-behaviours are combined using one of four arbitration methods. The macro-behaviours can use one of three different sensor modalities. This provides twelve different methods of performing any specific task based behaviour. The specific behaviours that make up a task dependent behaviour group are determined prior to runtime. The specific behaviour grouping, combination method, and sensor mode, is chosen at runtime by the deliberative component. The deliberative component that directs the reactive level is run on a desktop computer.

1.3.2 Deliberative Component

Trying to define what is meant by the term deliberative component is difficult as there are many different possible deliberative systems. One such definition states that a deliberative component generates global solutions to complex tasks using planning (Russell and Norvig, 1995). A second definition may be that a deliberative system is one that can choose the most appropriate output of a non-deterministic finite state machine. This is where the input and current state lead to a number of possible outputs. In other words a system that uses heuristic techniques to choose the best output from a number of possible solutions (Newell and Simon, 1972; Newell, 1980). Another definition is that a deliberative component makes use of symbols about its current state and/or environment to reason about and determine its output (Genesereth and Nilsson, 1987). With so many different possible definitions and interpretations, a more concrete definition needs to be made to clarify the situation. Within this thesis the word deliberative will refer to any system whose output is not only determined by its input and current state, but also by its previous states and/or the current/previous states of other systems. In other words a deliberative system is one whose output is based upon an extended memory beyond that of its own current state. That is not to say that a deliberative system can not use symbolic reasoning or heuristic problem solving techniques, only that for this definition it is not absolutely necessary.

One argument may be that this definition does not go far enough, and should include planning, symbolic reasoning etc. This is a valid point. However, there are many ways to implement the various elements considered to be deliberative, i.e. planning, problem solving etc. As these elements can not be standardised from one architecture

to the next, it is difficult to identify elements that can be considered fundamental to a deliberative system. In addition, the symbol grounding problem and the use of symbols within cognitive architectures can be understood in a number of different ways. For example, there are a number of different ways in which symbolic representations can be implemented within a neural network (see section 2.4). This poses the same problem as before. Symbol use can not be standardised from one architecture to the next. These points aside, it is clear that the definition used here goes beyond a reactive system. It states that the system's output is more than just its input and current state. Although the definition may not go far enough in defining a deliberative system, it provides a solid starting point.

1.3.2.1 a-CRIBB

The CRIBB (Children's Reasoning about Intentions, Beliefs and Behaviour) model was developed to investigate reasoning in young children (Bartsch and Wellman, 1989). This schema was implemented as a computer model to simulate knowledge and the inference processes of a child solving problems (Wahl and Spada, 2000). However, the computer model did not incorporate basic emotions that are present in the original schema.

The a-CRIBB model (Lewis, 2004) was developed to investigate the use of affective computing within the CRIBB schema. a-CRIBB added several new elements to the original CRIBB computer model. The two most relevant to the robo-CAMAL architecture are the distributed model of affect and the use of associations. The affect model distributes affect values across the entire architecture rather than have a centralised emotion module (see section 1.3.2.4). Associations are belief-goal-action combinations. These combinations detail the correct action required to achieve a specific goal given a specific belief

(see section 1.3.2.3).

1.3.2.2 The Belief-Desire-Intention Model

The belief-desire-intention (BDI) model (Georgeff et al., 1999) is a schema that calculates the actions of an agent based on its beliefs and its desires. A belief is a statement about the confidence of a proposition. The confidence the agent can have in a belief can vary. In the BDI model beliefs are based on input from the agent's perceptual system, and its previously held beliefs. The agent's desires are a set of goals which the agent wishes to achieve. The agent's current desires are based on its internal state, possibly its emotional state, and its previously held desires. Coupling the agent's beliefs and its desires generate a set of intentions or plans to achieve its goals. For example the agent has a goal to hit a ball. Its perceptual system generates the belief that there is a ball to the right. The agent can implement a set of plans to turn the agent right and move forward.

1.3.2.3 Associations

Associations are a construct that consist of a belief, a desire, an intention, and an association value (Davis, 2007). The associations provide an indication of the success of a specific set of plans given the agents current beliefs and desires. This allows the agent to quickly determine the most appropriate set of plans based on its beliefs and desires. The associations work in the following way. From a large list of associations the agent extracts only those that have a belief desire combination that correspond to the agent's current belief and desire set. Of the remaining associations the one with the highest association value is chosen. This represents the corresponding set of plans that are the most likely to achieve the agent's goal. The association value is then modified depending on the outcome of the agent's actions. If

it fails to achieve its goal the value is reduced. If the plans succeed in achieving the goal then the value is increased.

1.3.2.4 Distributed Model of Affect

Affective computing refers to the use of computers to explore emotion within cognitive architectures (Picard, 1998). There are many different models of affect, such as the use of discrete basic emotions or modal emotion. Rather than use a centralised model of affect, roboCAMAL uses a distributed model of affect. This means that various elements within the architecture have an associated magnitude that can fluctuate. For example, each belief has a confidence value which reflects the reliability of that belief. Furthermore each goal has an importance value that determines the level of relevance of that goal to the agent at that time. Also, as described in section 1.3.2.3, the association value indicates the likelihood of success of a specific plan given a specific belief desire combination. All these values fluctuate and are often highly dependent on other systems within the architecture.

1.3.2.5 Motivational Blackboard

The use of motivation is pervasive throughout the architecture (Davis, 2002) (see chapter 9). The most important aspect here is the use of a motivational blackboard. It is this blackboard that provides the backbone of the deliberative component. A blackboard system (Corkill, 1991) uses three components. The first is the blackboard, which is a global structure and holds all the relevant information such as the agent's beliefs, goals etc. This structure is accessible to the whole deliberative agent. The second component consists of various knowledge sources which access the blackboard. These extract the relevant information, manipulate it in some way, then post the result back to the blackboard. This could be a belief or goal update mechanism

for example. The final element is a control component, in this instance a motivational construct. The motivational construct consists of two elements. A motivator which contains several parameters including the agent's current goal and the association chosen to achieve that goal, and a method for evaluating that motivator (see chapter 9). This evaluation allows the various knowledge sources, such as the belief revision mechanism, access to the blackboard. This enables the motivator's success or failure to be determined. At this point the motivational construct allows the current motivator to be modified depending on the success of the goal.

1.3.2.6 The Domain Model

Robo-CAMAL operates within a controlled environment consisting of various objects. Its actions are also confined by what its physical body can perceive and do. It is therefore vital that information about the agent's environment and its physical body be incorporated into its architecture. This encoding is achieved with the use of a domain model. The domain model first defines the type of objects to be found within the agent's environment. There is also an abstract belief schema that details the structure and constituents of all possible beliefs the agent can have about its environment. The domain model defines the possible beliefs that are most relevant to the agent. It also defines the relationships between the stated beliefs. These belief definitions and relationships incorporate the situated nature of the agent into the architecture.

The domain model then goes on to define the goals the agent can have. These goals are dictated by the possible objects and beliefs defined by the model. They are also dictated by the possible actions the robot can perform. It also provides a list of all the possible actions the agent can undertake. In relation to robo-CAMAL, this refers to

the macro-behaviours (see section 1.3.1.1). These two elements incorporate some of the embodied nature of the agent into the architecture (see chapter 7).

Finally the domain model provides the object profiles, i.e. the information required to recognise an object. This incorporates both the situated and embodied nature of the agent into the architecture. It is situated in that it provides some of the physical attributes of the environments objects. It is embodied in that it provides information on how the agents sensors should process the perceptual data.

The use of the domain model provides two key advantages. The first is that the model allows the situated and embodied nature of the agent to be separated from the deliberative component. This means that, as is the case with robo-CAMAL, the deliberative component can be generic and non domain/environment specific. The second advantage is that the model makes it easy to pinpoint when and where within the architecture the anchoring of symbols occurs.

1.4 Summary

This chapter introduced the main aims of this thesis and briefly presented some of the areas within cognitive science that are relevant to these aims. It then went on to describe some of the key elements of robo-CAMAL, the architecture designed to investigate the proposed aims of the thesis. The remainder of this thesis is divided as follows. Chapter 2 gives a general discussion on some of the different cognitive architectures that can be used to investigate the issues surrounding cognitive science. Chapter 3 will look at the previous research undertaken that directly contributes towards the robo-CAMAL architecture. Chapter 4 will give a general overview on how the robo-CAMAL architecture will run. Chapter 5 will give an overview of the

agent's vision system. Chapter 6 will describe the reactive component of Robo-CAMAL, and how it passes information to and from the deliberative component. Chapter 7 then goes on to describe the domain model in some detail. It describes what is contained within the model, how this relates to both the anchoring problem and the situated and embodied nature of robo-CAMAL. Chapter 8 describes the association construct and how this is used to allow the agent to learn about and adapt its behaviour. Chapter 9 will discuss motivation within robo-CAMAL. Chapter 10 will describe the various experiments that have been carried out and provide the results of these experiments. Chapter 11 will discuss the results of the various experiments undertaken, and show how they relate to the aims of the thesis. It will also consider some of the flaws in the model and provide some suggestions on how to improve it. Finally chapter 12 will summarise the thesis and provide the final conclusions.

Chapter 2

Cognitive Architectures

2.1 Introduction

Cognitive architecture refers to the design and organization of the mind (Sloman, 1999). This chapter will first look briefly at cognitive science and the possible designs for mind. It will then describe some of the various cognitive architectures that have been suggested and investigated over the years. This will include a description of symbolic artificial intelligence (symbolic A.I.) with reference to the physical symbol system. Connectionism is then described using neural networks as an example. The chapter will then go on to describe reactive or behavioural systems. These are systems that link a robot's perceptual system directly to its actuators. This enables autonomous robots to operate in dynamic physical environments. Finally, hybrid architectures, which are a combination of several different architecture types, are explained.

2.2 Cognitive Science

Cognitive science is the study of cognitive processes and how they are integrated to form a mind. The term cognitive in this context is used for any kind of mental operation (Lakoff and Johnson, 1999). Cogni-

tive processes include systems such as memory, perception, learning etc. What then is the purpose of a mind? One important approach states that mind is used to decide the next action of an agent (Franklin, 1995; Newell, 1990; Sloman, 1993).

It has been argued that there are twelve issues for cognitive science to address (Norman, 1980). These are belief systems; consciousness; development; emotion; interaction; language; perception; learning; memory; performance; skill and thought. By defining a blueprint for how various cognitive mechanisms that address these issues are integrated, it should be possible to describe how mind works.

What is the correct way to integrate such systems? Is there some common principle with which a mind is built? The Unified Theory of Cognition (Newell, 1990) argues the need for a set of general assumptions for cognitive models. This includes how cognitive mechanisms should be integrated. Any unified theory of cognition must explain how agents react to their environment, exhibit goal directed behaviour, generate goals, how they represent knowledge, and how they learn. Another important aspect is that any theory needs to be able to explain observed behaviour. This places constraints on any theory. If it can not explain experimental results or specific phenomena, then it may be incorrect. By using such constraints and assumptions in developing a cognitive architecture, it should be possible to create an artificial mind. This artificial mind could then be used to control a simulated agent or a physical robot.

There are various views on the form which a mind can take. One prevalent view is that mind can be decomposed into task specific modules. For example, mind can be made of distinct processing units. One unit may handle memory, another may process visual data. This is often referred to as the modularity of mind view (Fodor, 1983). An alternative view suggests domain-general processing. This is where

mental activity is distributed across the entire mind, and can not be decomposed into independent units (Uttal, 2003).

As robo-CAMAL claims no similarity with biological intelligence, the modular view point is taken due to the ease with which it can be implemented.

2.3 Symbolic A.I.

A symbol is a token that represents something with an independent existence. This makes them an incredibly powerful tool for conveying information and knowledge. Symbols are most commonly used in language where their form is generally arbitrary. They can also be iconic where their form is often visually related to the object they represent. Much of the work done in regard to symbolic A.I. is based on the physical symbol system hypothesis. The hypothesis states that a physical symbol system has the necessary components for general intelligence (Newell and Simon, 1976).

A physical symbol system is a set of symbols that relate to each other in some way. This symbol structure must be realised in a physical system such as a computer. A good example of a physical symbol system is a production system (Post, 1934). A production system consists of a collection of rules, a set of conditions or facts that reside in a working memory, and a series of actions that produce new facts based on the stored facts (Simon, 1999). A production system takes the form

IF (conditions) **THEN** (actions)

If the conditions or facts in the working memory are true, then the actions are carried out or new facts are generated. However, a production system alone is not intelligent.

To demonstrate general intelligence the production system must be used to achieve goals. For symbolic A.I., goals are achieved by solving problems using production rules. The production rules operate within problem space, which is a representation of the problem including initial, final, and possible intermediate states. The strategy used to search the problem space is highly dependent on the problem, the goal, and the capability of the symbol processing system. For example a chess player will spend the majority of their time considering the consequences of a particular move. This search is highly selective, only considering a few possible moves (Newell and Simon, 1972). This is known as a depth first search, as the problem space is considered a long way into any possible move. As unpromising moves are disregarded almost immediately this type of search is heuristic. That is, the problem space is greatly reduced in width. This concept takes the view that intelligent agents use problem space to achieve goals. It is summed up by the problem space hypothesis that states

> *"the fundamental organizational unit of all human goal-oriented symbolic activity is the problem space"* (Newell, 1980).

It is clear that the use of physical symbol systems to investigate problem space is beneficial. It is also clear from the previous chess example that humans utilise symbol systems to search problem space. If much of human knowledge is symbolic then encoding a symbolic architecture should be comparatively simple. This is because symbol systems are far easier to relate to human intelligence. This provides a major advantage for symbol systems as they are easier for humans to understand, and therefore implement.

One system that uses a symbolic architecture to solve problems is SOAR (Rosenbloom et al., 1993). SOAR uses production rules to search through the problem space of a particular problem to achieve

a specific goal state. Each move through problem space consists of an elaboration phase, where any information relevant to the problem in the current state is brought to the working memory. Then a decision procedure is then executed. This considers information from the previous elaboration phase, and assigns preferences to the possible actions that can be taken. If an impasse is reached where two possible actions are equally preferable, or if there is not enough information to make a decision, then a sub-goal is created.

Various methods are used to acquire more knowledge that enable SOAR to reach a state that satisfies the sub-goal. One such method is a means end analysis. This calculates the difference between each available action and the eventual goal state. The action chosen is the action that shows the least difference from the goal state. Once SOAR achieves the sub-goal it uses a method called chunking in order to learn about what to do if the impasse ever arises again. This is done by creating a production rule that describes the situation of the state leading up to the impasse. It also creates an action that uses a single step to achieve the state required to satisfy the sub-goal.

Another system that uses a symbolic architecture is Shakey the robot (Nilsson, 1984). Shakey uses a camera to observe its environment. It also uses an off board computer to process the information and control the robot's motion. The robot uses the camera and vision processing routines to build a representation of its environment. This is generated and manipulated using predicate calculus statements. The robot was designed to perform simple tasks, provided by user input, in an office environment.

The system has various low level action predicates that are used to provide information about the robot's current state and location. For example, **at** and **theta** give the position and bearing of the robot. There are also predicates to control the robot's actuators, such as

turn(deg), which turns the robot by the said number of degrees.

The intermediate level actions are responsible for seeing that the pre-conditions for the designated action are satisfied. This is done by using the low level actions. One such intermediate action is **gotoadjroom** which uses various low level actions to pass through an open doorway.

An example of how this action is used is the case where the agent has a goal to push a block. If the block is not present in the robot's current room this intermediate action may be called. This is done in an attempt to satisfy the pre-condition of finding a block. Intermediate actions that affect the environment such as **push** are also responsible for changing the robot's current world model.

To achieve a goal the system uses a planning system called the Stanford Research Institute Problem Solver (**STRIPS**). **STRIPS** attempts to find a sequence of intermediate level actions that change the environment such that the input goal is true. To do this **STRIPS** needs to know about the effects of those actions on the environment, i.e. a model of each action.

The systems mentioned above work in environments that have been specifically designed for them. They either perceive the environment and create an internal model, or have a model of the environment pre-programmed. This is so that a planner can operate. The planner tries to achieve the goals it has been given. For Shakey this is often to get the robot from point **A** to point **B**. Despite using simple constructed environments Shakey often operates slowly. Most of the computational time is taken up generating and updating the internal environment model.

Like the previous two systems described, robo-CAMAL will make use of symbolic processing. This occurs at the deliberative level within the architecture. However, the use of a reactive system will allow the

agent to operate without a complicated internal model of its environment. This means that very little computational time is used in updating that model.

2.4 Connectionism

The connectionist approach takes the idea that intelligence arises from the interactions of large numbers of highly interconnected simple processing units (Rumelhart and McClelland, 1986; Lloyd, 1989). This idea models cognitive architecture on the brain. In this case the simple processing unit is the equivalent of a neuron. All neurons have an input component (dendrite), a trigger component (cell body), and an output component (axon), (figure 2.1). The dendrites receive signals

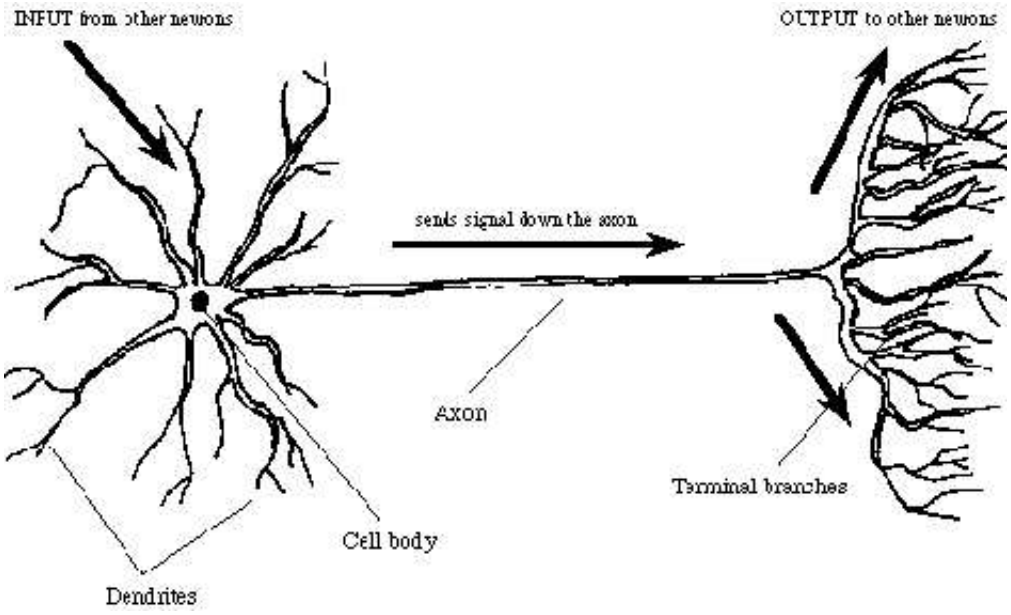


Figure 2.1: A neuron.

from other neurons and pass this signal as a voltage to the cell body. The cell body combines the input voltage with its current voltage. When this voltage exceeds a threshold, the cell body fires an output voltage down the axon to other neurons. This can be described in a more formal way with a threshold logic unit (figure 2.2). The inputs

x_i can take positive (excitatory) values or negative (inhibitory) values. The weights of the system w_i (again the values can be positive or negative) determine the strength of the signal passed to the cell body. The cell fires an output signal, y, once the input value reaches a threshold value h. The value y is then passed to the next threshold logic unit. This unit can be used to produce logic gates. For example the cell body has two inputs x_1 and x_2, each are binary. The associ-

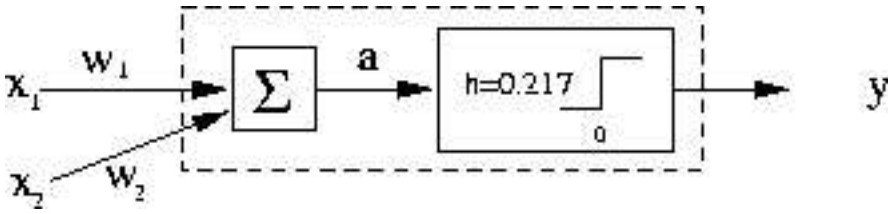


Figure 2.2: Threshold logic unit.

ated weight values w_1 and w_2 are both 1. If the threshold is also 1, the unit fires if either x_1 or x_2 are 1. However, if the threshold is 2, then the unit fires only if both x_1 and x_2 are 1. This defines an **OR** and **AND** gate. It is clear that if a large number of threshold logic units are interconnected they manipulate input values in a logical way.

There are some important advantages to the connectionist approach. The use of many simple processing units in parallel means there is no central processing unit. Each unit receives an input and decides on its output alone. This means that the system does not need to waste time collecting all the relevent information to make a decision. Another advantage is that a connectionist model is robust. The failure of a small number of units results in a minor reduction in performance, not a complete break down of the system.

There are however challenges in developing a connectionist architecture. One such problem is the question of how objects are represented. There are several styles of representation for the connectionist approach. One such method is known as local representation. In this

type of representation individual units represent specific properties or features of the environment. When the connection strength from one such unit to another is strongly positive, if the first representation is true, so is the second (Ramsey et al., 1990). A second method is distributed representation. For this type of representation it is the pattern of activity over many units, not the meaning of any individual unit, which is important (Rumelhart and McClelland, 1986). An example of this form of representation is the perceptual symbol system (Barsalou, 1999). This method states that a perceptual symbol is a record of the neural activation that occurs during the perception of an event. A third method of representation is featural representation. This approach is an amalgamation of the two previous approaches. Similar to a distributed representation, the pattern of activation over many units is significant. However, each unit represents a feature of the distributed representation. This means that individual units can be thought of as representing a small part of the overall object or concept (Lloyd, 1989). These are just some of the possible ways in which objects can be represented. It is a major technical challenge to develop a connectionist system that can use symbolic representations. It is therefore a major challenge to develop connectionist systems that can reason about high level problems that are trivial to a symbolic A.I. system.

A connectionist approach was not adopted in the development of robo-CAMAL. Though connectionist systems are considered to be good at learning (Franklin, 1995), the challenges in developing one that can reason symbolically go beyond this thesis.

2.5 Reactive Architecture

In general terms, intelligent agents operate in a perception-action cycle (Holyoak, 1999). Mind is therefore often split into three separate components, sensory systems, central processing systems and motor systems. Information from the agent's sensors is processed by the sensory systems. This information is then passed to the central processing system. The central processing system uses this information to decide the next action. This instruction is passed to the motor system which then carries out the required action.

The central processing system is traditionally decomposed into functional modules (Stillings et al., 1995). These modules can include elements such as memory, learning etc. This approach means that generating an object avoidance behaviour, for example, requires the implementation of a sensor processing module, a planner or decision making module, as well as a module to control the agents motion to avoid the detected object. This is because the whole architecture needs to be implemented before any behaviour can be observed.

However, a simple coupling of sensor to a motor can produce the same results. For example, consider two photosensitive plates each connected to a motor (figure 2.3). The charge on the photosensitive plate inhibits the speed of the motor. The more photons hit the plate, the slower the motor operates. This simple coupling of sensor to motor in this way produces an agent that avoids, light sources (Braitenberg, 1984).

The classical approach to A.I. decomposes intelligence into functional information processing modules (Brooks, 1991) e.g. planning, learning, memory etc. Each individual module does not produce any behaviour. Only when all the various modules are combined will an agent produce any behaviour (figure 2.4). By contrast the reactive ap-

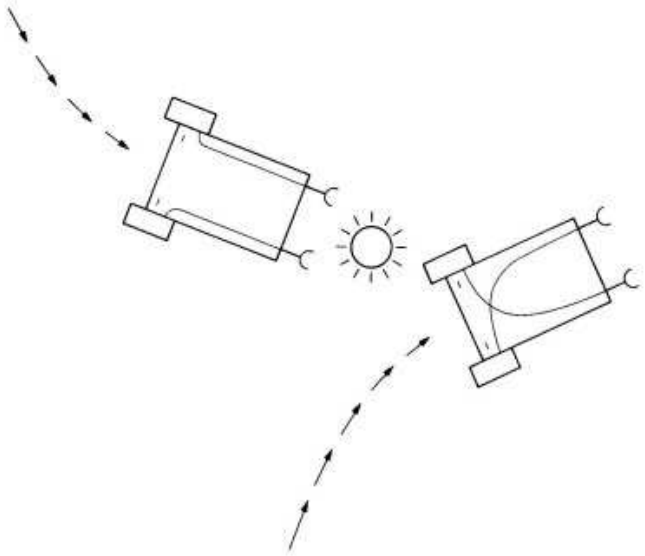


Figure 2.3: Light sensitive agents (Braitenberg, 1984).

proach decomposes the agent into behaviour producing modules (figure 2.5). Each module produces a distinct behaviour independent of all the other modules. In this case the agents behaviour is determined by the combined effect of the individual modules acting independently.

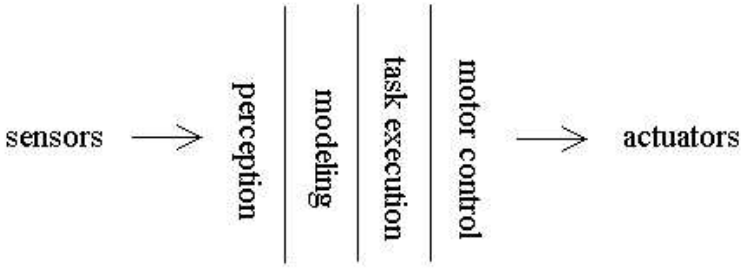


Figure 2.4: Decomposition by function.

The reactive approach is more robust than the classical approach. Consider the flow of information through the two different architectures in figures 2.4 and 2.5. In the classical approach each module is placed in series with information flowing from one module to the next. If one module breaks down or is removed, the flow of information stops. This means that the agent stops working. The modules

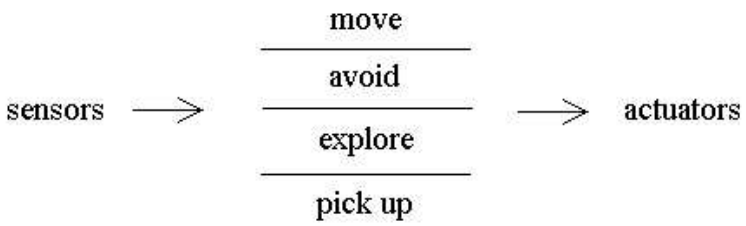


Figure 2.5: Decomposition by action.

in a reactive system are placed in parallel, with information from the sensors passed to all. If a module breaks or is removed, information is still passed to the others, and they can still operate (Brooks, 1986).

An important aspect to the reactive approach is that there are no internal representations (Brooks, 1991). In a reactive agent each module is in essence a finite state machine (FSM). The inputs to the FSM come from the agent's sensors. Depending on the FSM's input and current state, the system can either remain unchanged, or alter its current state. The output of the FSM is connected to the agent's actuators and is determined by the FSM's current state.

Another aspect to reactive architectures is that the behaviour producing FSMs have the ability to inhibit other FSMs. Consider an agent architecture that has two simple actions, move forward and avoid object. If there are no objects present the agent will perform the move forward action. If however the agent's sensors detect an object in front of it, the information passed to the avoid object action causes it to inhibit the move forward action. The avoid object action then changes the agent's trajectory. Once the object is no longer in front of the agent the move forward action continues (figure 2.6).

Behaviours such as object avoidance, that are difficult to achieve using symbolic methods, become trivial if the reactive approach is used. The fact that the agent does not need to create an internal model

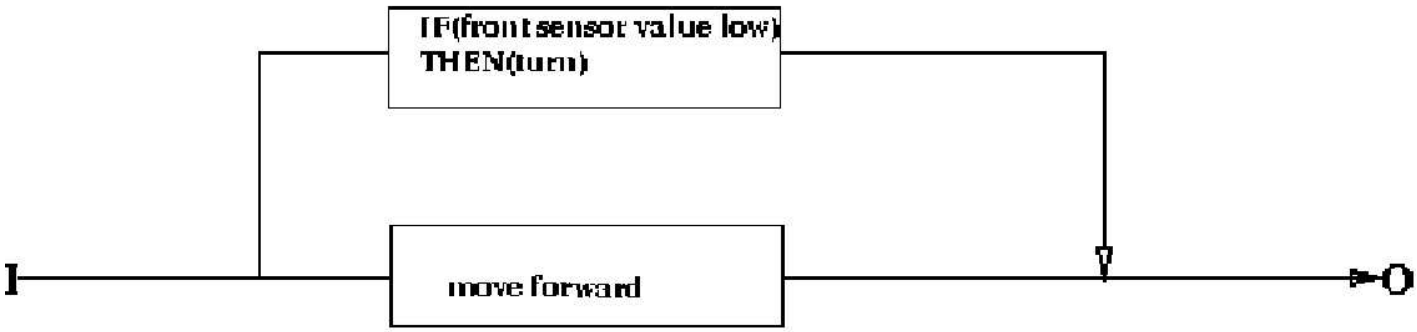


Figure 2.6: Simple reactive architecture.

of its environment removes many of the problems attached to symbolic A.I. For example, there is no need to anchor symbols to sensor data. The removal of the need for an internal model of the environment also saves time, and allows a reactive system to react quickly in a complex dynamic environment. Though reactive architectures provide an excellent way of generating simple behaviours for agents, those behaviours often need to be defined in advanced.

Some reactive agents also have the ability to learn (Mahadevan and Connell, 1991). However, such agents require pre-defined goals (Mahadevan and Connell, 1991). This means that a reactive agent's goals must be defined by the system's designer. Goals can not emerge from any reasoning ability the agent has. As the agent's behaviours are fixed it will respond in the same way to a specific stimulus. This means the agent is unable to learn without direct intervention from the agent's designer to provide it with new goals.

Reactive agents have often been considered capable of simple behaviours only. This however is not true as there are reactive systems capable of planning (Langer et al., 1994; Stenz, 1995). One such example uses Trulla, an algorithm that determines a path for a reactive agent to follow (Murphy et al., 1999). Trulla has an inbuilt a-priori

map. Its goal is a location on that map. Trulla calculates every possible path from every possible location to that goal on the map. This calculation produces a vector field that provides the optimal direction for the agent at every point within the map. The agent then fires a move to goal behaviour and follows the vector field. If an unexpected object is encountered, the avoid object behaviour fires. At the same time the move to goal behaviour continues to follow the vector field along its new path. However if a major discrepancy between the a-priori map and the agent's sensors occurs, Trulla recalculates the vector field with the new map information.

The use of a reactive system is important to robo-CAMAL. It enables robo-CAMAL to operate in a dynamic environment without a complex internal model of that environment. It also filters out unnecessary information about the agent's environment that has the potential to slow the deliberative components processing speed.

2.6 Hybrid Architecture

Hybrid architectures are architectures that combine two or more different types of cognitive architecture. For example, a connectionist-deliberative system or a reactive-deliberative system (Arkin, 1990; Pfeifer, 1995). Hybrid architectures seek to avoid the disadvantages of their component architectures, whilst retaining all their benefits. A common hybrid architecture is the reactive-deliberative architecture. This architecture uses a reactive component to interact with its environment. This filters out unnecessary information. Relevant information is then passed to a deliberative component. Constraining the information from the environment allows the deliberative component to reason about its environment in a more efficient manner.

One important example of a hybrid system involves the use of event

calculus. In this example the robot has a set of reactive behaviors, and uses event calculus to reason about its environment in a deliberative manner (Shanahan and Witkowski, 2001). Event calculus is a formalism for representing events and their effects (Kowalski and Sergot, 1986). The event calculus contains three types of elements, events or actions, fluents, and time points. A fluent is a value that has the capacity to change over time. These elements can be reasoned about using various statements, some of which can be seen in table 10.1. A goal within this framework is a predicate containing a fluent

Predicate	**Meaning**
Initiates (a,b,t)	Fluent b starts to hold after action a at time t
Terminates (a,b,t)	Fluent b ceases to hold after action a at time t
Initially (b)	Fluent b holds from time 0
$t_i < t_j$	Time point t_i is before time point t_j
Happens (a,t)	Action a occurs at time t
HoldsAt (b,t)	Fluent b holds at time t
Clipped (t_i,b,t_j)	Fluent b is terminated between times t_i and t_j

Table 2.1: Event calculus predicates (Shanahan, 1999).

with a specific value. For example, if an agent's goal is to hold a fork, the goal is stated as *HoldsAt* (is_holding_fork(true),t). A planner can then search the problem space to determine the correct sequence of events to achieve the goal predicates (Shanahan, 2000). The planner can then calculate the correct sequence of reactive behaviours to achieve the desired goal.

The Guardian intensive-care monitoring system (Larsson and Hayes-Roth, 1998) monitors patients in intensive-care, and takes appropriate action when necessary. It is also based on a hybrid architecture. The BB1 architecture (Hayes-Roth, 1995) shown in figure 2.7 uses a reactive component that has reflexes and perception-action responses. The deliberative component uses a blackboard system in order to reason about and act on its environment (see section 3.6).

The blackboard system uses a set of data structures in the global

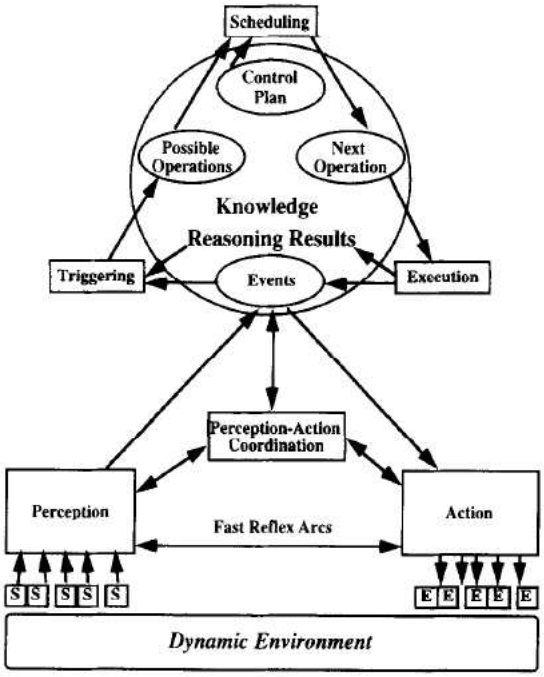


Figure 2.7: The BB1 agent architecture (Hayes-Roth, 1995).

memory (inside the circle in figure 2.7) and a three step execution cycle. The triggering step triggers all the possible actions in relation to the current events. A scheduling step rates all possible actions against the current control plan. It then chooses the action with the highest rating to be executed. The execution step executes the next action and updates the current events.

The control plan is central to rating the possible actions. It is a data structure containing various component plans. Each step in a plan specifies

- a set of intended actions in terms of their desirable attributes, e.g. resources they consume, what kind of knowledge they apply to etc.
- a condition under which the plan should be activated.
- a condition under which the plan should be deactivated.
- a rating function that evaluates how closely the outcome of the

possible action compares to the desired outcome.

Robo-CAMAL is also a reactive-deliberative hybrid system. This approach has been taken in order to exploit the advantages of both approaches.

2.7 Summary

This chapter described various cognitive architectures. It started by briefly discussing cognitive science. It looked at some of the possible views that can be adopted on the design of mind. The rest of the chapter then looked at some of the possible architectures that can be used to develop an agent. This included symbolic A.I., connectionism, reactive and hybrid architectures. It also gave some examples of the various types of architectures.

Chapter 3

The Robo-CAMAL Framework

3.1 Introduction

This chapter will review the previous work that has directly influenced the design of Robo-CAMAL. This work takes the reductionist's view point (Fodor, 1983; Minsky, 1986) and will introduce the view of mind as a control system for an agent. It will then look at the CogAff architecture that adopts this view. Next the belief, desire, intention (BDI) model is introduced, and the CRIBB and a-CRIBB architectures that make use of this model are then described. Following this, blackboard systems are then introduced, and use of motivation from the view of mind as a control system is then discussed. Finally all these elements are brought together in the CAMAL architecture.

3.2 Mind as a Control System

Mind can be viewed as a collection of cognitive processes. These processes are integrated in a way that enables an agent to decide its next action (see section 2.2). One approach that takes this view is the use of mind as a control system (Sloman, 1993). This takes the view that mind is a collection of many different control processes passing data between them asynchronously. This is an alternative to the view

of mind as a single computation that can be reduced to a single state at any given time (Sloman, 1993). Control processes can be a number of different things such as beliefs, desires, reflexes, motivators etc.

In essence control processes are general behaviours within the agent. Each control process can be modified or updated at any time by any other control process. Control processes do not need to exhibit external behaviour in that they can also be used to control the agent's internal state. One important element is how the control processes interact with each other.

There are a number of different possible interactions that must be considered. These are (Sloman, 1990, 1993)

- How a control process modifies and is modified by other control processes.
- The type and amount of information passed between the control processes.
- Whether the information transferred between control processes is direct or filtered.
- If control processes generate behaviour directly, or if they modify other control states in order to achieve a goal.

This approach is fully consistent with the concept that the organisation of the mind is modular as discussed in section 2.2.

The advantage of this approach to mind is that each of the control processes can be isolated and investigated. Each control process can be tested on quantifiable qualities such as speed and accuracy. This makes assessment of any artificial mind developed far simpler than if notions such as consciousness and reasoning ability are used to describe mind.

One important aspect of this approach to mind is that the number of possible designs for each control process is potentially infinite. By placing constraints on the design space of a control process, different designs for that control process can be compared. This can be extended to the whole agent in a specific environmental niche (Sloman, 1993; Davis, 2001). By designing two complete agents of different designs to compete in a constrained environmental niche space, the two agents can be compared in a quantifiable way. This is analogous to two different species inhabiting the same environment. This could be a predator-prey relationship, or each agent competing for the same resources.

The use of design and niche space constrains the design of the control processes. This allows more emphasis to be placed on quantifiable aspects, such as speed or efficiency, when assessing the agents performance.

3.3 CogAff Architecture

The view of mind as a control system requires that the many control processes be combined into a coherent structure. From the interaction of this structure and its environment, intelligent behaviour should emerge. One such architecture is CogAff (see figure 3.1) (Sloman, 2001).

This is a three tier, three column architecture. The various control processes are positioned within the 3×3 grid depending on their function. The columns from left to right consist of perception processes such as vision processing systems; central processing units that make decisions based on perceptual input and the agent's current internal state; finally, the action processes that control the architectures output, e.g. control of a robot's motors to achieve a desired movement.

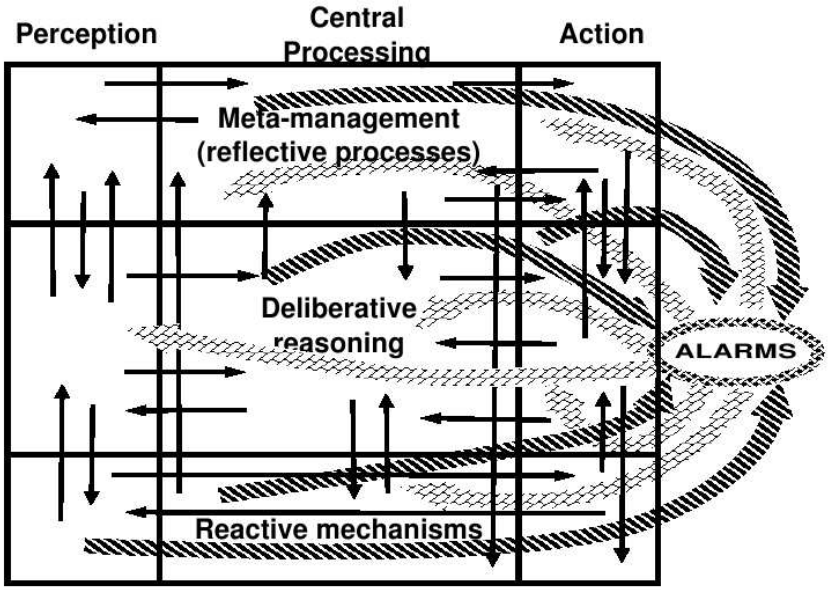


Figure 3.1: CogAff architecture (Sloman, 2001).

The three tiers from bottom to top consist of reactive mechanisms, deliberative processes and meta-management. Reactive mechanisms are very simple systems where input is directly connected to the output using a pre-determined association. For example a mobile robot may have a mechanism that states

$$\textbf{IF}(\text{sensor 1 value} < x)\ \textbf{THEN}(\text{increase speed of motor 1})$$

where x is a predetermined value. More on reactive systems can be found in section 2.5.

Deliberative mechanisms are more complex and are used to analyse and reason about their input. Though the design of a deliberative process is dependent on what the agent is designed to do, there are a few common themes. Deliberative mechanisms generally use structured representations of their environment or internal state in their operations. They often have a memory of previous occurrences that can be factored into the decision making process. However the term deliberative has a more constrained meaning within this thesis (see section 1.3.2).

The meta-management level monitors and controls the interaction of the reactive and deliberative levels. By overseeing the entire architecture, the reflective level looks to suppress unwanted behaviour, and promote useful behaviour. An example of this is to note when an important goal has not been achieved, and to find out why. Another example is to notice circular behaviour. This is when goal **A** requires the completion of goal **B**, but goal **B** requires the completion of goal **A**.

The final component in the CogAff architecture are alarms. Alarms are emergency responses to situations that the agent is otherwise unable to deal with. They are purely reactive in nature. Alarms enable an agent to deal with emergency scenarios. If the agent's environment is changing too quickly for its internal processing systems to keep up, an alarm may be triggered. This alarm may redirect the whole system by deactivating non-essential processes, and focusing the system on essential ones.

3.4 Belief Desire Intention Model

The belief desire intention (BDI) model (Bratman, 1987; Georgeff and Rao, 1995; Georgeff et al., 1999) provides a method to control the flow of information through a deliberative architecture. Beliefs are a way of representing the state of the environment. Desires or goals represent some desired end state. One way of achieving a goal is to simply execute an action based on the beliefs the agent has at that time. Unfortunately if the environment changes the agent's beliefs are modified accordingly. This means the previous action is no longer valid. As the action has no access to the new beliefs, it will still act and may fail to achieve its goal.

It is for this reason that intentions are required. Intentions are

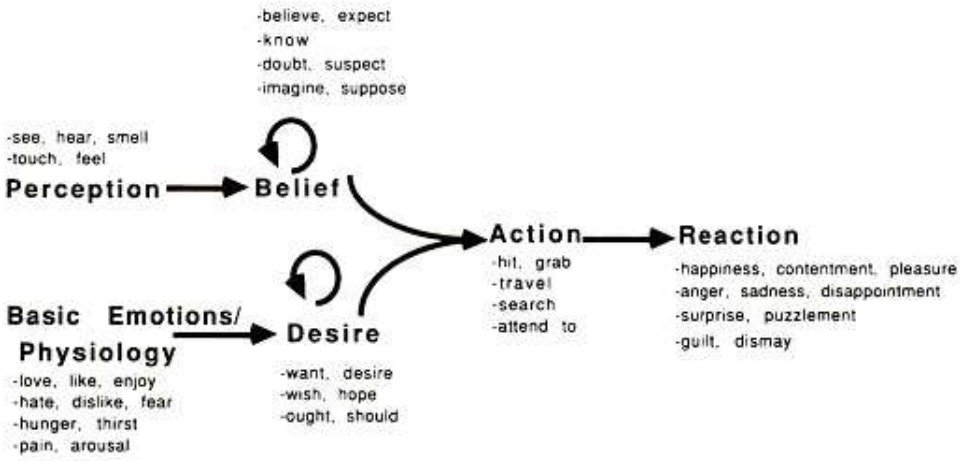


Figure 3.2: Belief-Desire reasoning scheme (Bartsch and Wellman, 1989).

plans that are based on the agent's desires. They also have access to the agent's beliefs. If the environment changes and the agent's beliefs are updated then the agent's intentions can be modified. This prevents the agent from failing to achieve its goals. The BDI schema of the CRIBB cognitive model can be seen in figure 3.2

The inputs into this model are perception, basic emotions and physiology. The agent's beliefs are determined by its perception of the environment and its previously held beliefs. The agent's desires are determined by its emotions, physiology and its previous desires. The coupling of an agent's belief and desire manifests itself as an action. This action leads to a reaction such as a change in the agent's environment or emotional state. This reaction in turn can alter what the agent perceives, its basic emotions or its physiology.

3.5 CRIBB and a-CRIBB

The CRIBB (Children's Reasoning about Intentions, Beliefs and Behaviour) model was designed to investigate the belief-desire reasoning model in young children(Bartsch and Wellman, 1989). The model states that a person's actions can be explained by their beliefs and

desires. It is based on the BDI schema shown in figure 3.2.

CRIBB is a computer model that simulates knowledge and the inference processes of a child solving problems (Wahl and Spada, 2000). The types of problems it is designed to solve consist of various questions in the form of stories. Neutral stories:

> *"Jane is looking for her cat under the piano. Why is Jane doing that?"*

Anomalous desire stories:

> *"Jane hates frogs. Jane is looking for a frog under the piano. Why is Jane doing that?"*

Finally, anomalous belief stories:

> *"Janes' cat is hiding under a chair. Jane is looking for her cat under the piano. Why is Jane doing that?"*

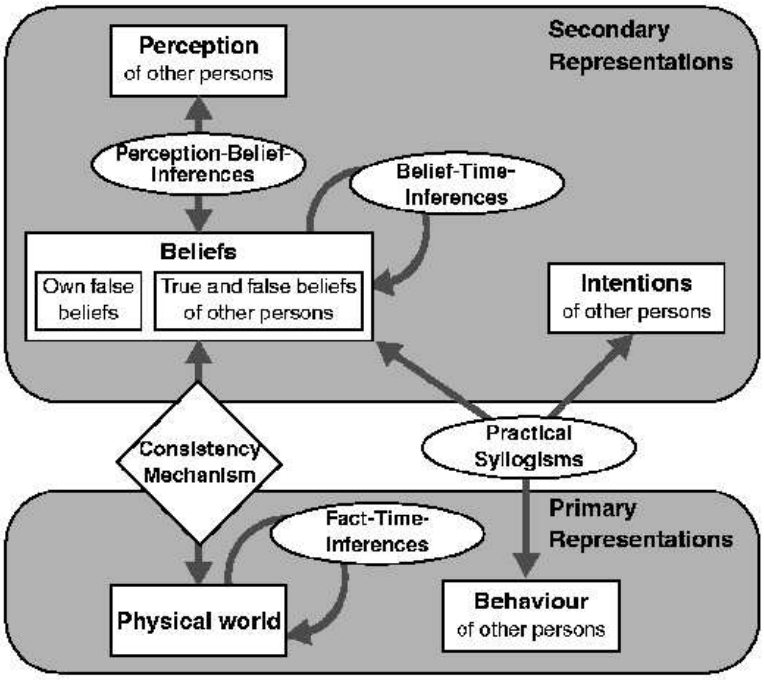


Figure 3.3: CRIBB architecture (Wahl and Spada, 2000).

The CRIBB architecture can be seen in figure 3.3. The two main bodies in this architecture are primary and secondary representations. Primary representations describe the agent's simulated environment. Within the primary representation there are statements about the simulated environment, and the behaviour of other agents within that environment.

Secondary representations describe the agent's internal state. They include statements about the agent's beliefs, intentions and what it perceives. Within this framework are a set of inference schemata (ellipses in figure 3.3). These are based on the belief desire model in figure 3.2. Perception-belief inferences represent knowledge about the relationship between perceptions and beliefs. If the agent perceives X then agent believes X. Fact-time and belief-time inferences deal with facts and beliefs along a time line. If an environmental statement or belief is true at time t_i then it is true at time t_{i+j}, unless there is information to the contrary.

Practical syllogisms represent the relation between intentions, beliefs and the behaviour of another agent. There are four types of practical syllogism implemented. The first is used to predict behaviour. If an agent desires something and believes a specific action will achieve that desire, the agent will perform that action. The second infers the intention of an agent given its behaviour and beliefs. The third infers an agent's belief from its intention and behaviour. The fourth is the attribution of the first practical syllogism to another agent i.e. assumes other agents use the same practical syllogisms. CRIBB is a competence model and always answers correctly. It is used to study what knowledge is required and what inference schemas are used to solve a problem.

The affective CRIBB (a-CRIBB) model is an extension of the BDI schema used in the CRIBB model (Lewis, 2004). This can be seen

in equation 3.1. A represents the affect values that order the various propositions. P represents the agent's perceptions. B, G, and I represent the agents beliefs, goals, and intentions respectively.

$$\begin{aligned} A \otimes P \otimes B &\rightarrow B' \quad \text{Belief updates} \\ A \otimes B' \otimes G &\rightarrow G' \quad \text{Goal set selection} \\ A \otimes B' \otimes G' \otimes I &\rightarrow I' \quad \text{Intention mechanism} \end{aligned} \tag{3.1}$$

The first major change is the environment in which the framework is implemented. Rather than a child that is asked questions, a-CRIBB is a simulated fungus eater (Toda, 1982). The agent's motivation is to collect minerals and fungus rather than answer questions. The original CRIBB architecture shown in figure 3.3 did not include the emotional and physiological input seen in figure 3.2. The second extension therefore is the inclusion of an emotion-like element. The various extensions to the CRIBB model are,

- A distributed model of affect.
- The inclusion of affective affordances applied to the perceptions of the system.
- A revision of the consistency mechanism to include affective correspondences.
- A description of a goal base including goal maintenance, goal importance and goal achieved mechanisms.
- A central monitoring system.

A distributed model of affect means that rather than having a separate emotion module that interacts with other sub-systems, affect is distributed across all the sub-systems. This distribution of affect can cause communication and synchronisation problems between the

agents various sub-systems. This problem is controlled by the central monitoring system (CMS). The CMS communicates with all the sub-systems via the exchange of semantic and control messages. Control messages are simple messages that turn sub-systems on and off. This depends on internal or external events. The use of control messages allows the CMS to control the whole cognitive system.

Events are evaluated with respect to a goal base. Goals in a-CRIBB have five elements (Lewis, 2004). The first two elements represent the name and the type of the event under evaluation. The third element represents the required state of the event. The fourth element of the goal describes its importance value. The importance value is dynamic and is affected by internal and external events. The goal importance is also determined to some extent by the agent's drives. Each drive value is determined by internal and external events. When a drive value exceeds its threshold level, the relevant goal importance value is increased. Alternatively if the drive value falls below a particular threshold, the goal's importance decreases.

The final element of a goal is its threat value. This value keeps track of the number of threats to a goal. It is determined by the goal maintenance mechanism. Goals are associated with the belief of an event. However, new information may enter the system that contradicts the current belief. If the goal importance is high then the goal maintenance mechanism rejects the new information. This is to maintain the goal. Doing this also raises the goal's threat level. If the threat level exceeds a specific value then the goal is rejected.

If several goals are present the one with the highest importance is chosen. Sub-systems are then activated to achieve it. This in turn could create new goals in order to achieve the preceding goal. Once the action has been performed, the goal achievement mechanism examines the environment. This is to determine whether or not the goal has

been achieved. The achievement or failure of a goal has an affect on the importance value of that goal. It can also affect other goals present in the system, as well as create or remove goals.

Depending on whether or not a goal is achieved, the affective correspondence of the belief attached to that goal is altered. Affective correspondence is a value that is associated with each source of information to the agent (Lewis, 2004). The value is an indication of the reliability of the information source. If a belief is found to be incorrect then the affective correspondence value for the source of that belief is reduced. Conversely if a belief is found to be correct, then the affective correspondence value for the source of that belief is increased. The more reliable a source of information, the higher its affective correspondence.

Within a-CRIBB each element in the agent's environment has an affective affordance value that can vary (Lewis, 2004). Affective affordances are an extension to the theory of affordances (Gibson, 1986). Affordances are properties offered to an agent by elements within its environment. Fire affords warmth (positive affordance) and also burns (negative affordance). This example demonstrates that affordances are both objective and subjective. Fire exists, it does provide warmth, it does burn, and therefore these are objective observations. The affordances are subjective in the sense that an agent will only perceive the properties relevant to it. An energy source appears more attractive to an agent if it is hungry than if it is not. In a-CRIBB the affordance value allows the agent to direct its perceptual system. The more significant an object in the environment, the higher its affective affordance. Objects within the environment can be ordered in terms of this value. This means that things with a high affective affordance can be attended to first. This has the affect of directing the agent's perception to the most important object in its field of vision. Affective

affordances can be altered by the experiences and requirements of the agent. For example if the fungus eater is low on energy then fungus has a higher affective affordance. The affective affordance can also be affected by the outcome of goals.

3.6 Blackboard Systems

A blackboard system (as used by the BB1/Guardian discussed in section 2.6) provides a method for solving problems, and consists of three elements (figure 3.4). These elements are knowledge sources, a blackboard, and a control component (Corkill, 1991).

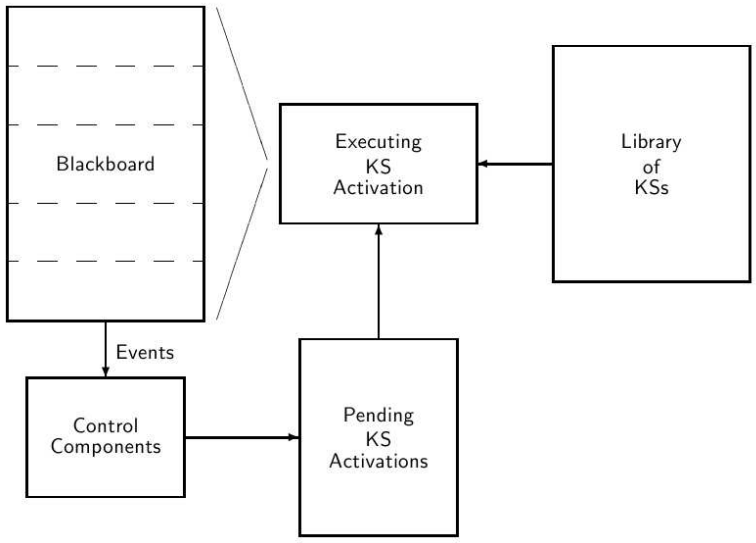


Figure 3.4: Blackboard system (Corkill, 1991).

The blackboard is a global structure and is available to all the components of the system. It contains information on the problems being solved such as, partial solutions, alternative solutions, previous solutions etc. It also contains a method of communicating that information with the rest of the system. Finally it has a knowledge source trigger. The trigger mechanism simply informs a knowledge source that there is information relevant to it. This in turn prompts the knowledge source into action.

A knowledge source is a process that manipulates or performs a calculation with information from the blackboard. Once the knowledge source has performed its operation, the information is posted back to the blackboard. Each knowledge source is separate and independent of the other knowledge sources.

The final component of a blackboard system is the control component. At any one time there may be several knowledge sources wishing to access the blackboard. The control component decides which knowledge source is allowed to access the blackboard at any one time. The control component also has access to the blackboard. It can therefore determine if the current problem is similar to a previous problem that has already been solved. In this way it can allow the most appropriate knowledge source access to the blackboard. This can hopefully provide the solution in the most efficient manner possible.

Some blackboard systems use a distributed method of control (Craig, 1989) as opposed to a control component. This approach has not been adopted and so will not be discussed in any detail here.

3.7 Motivational Control Process

In psychology, motivation is the driving force behind all actions of an organism (Beck, 2000). If an agent is to control its actions it must have some form of desire or motivation in order to act (Dennett, 1984b; McFarland and Bösser, 1993). If it is taken that actions are performed to achieve a positive internal state, then motivation is the search for positive internal states, and the avoidance of negative ones. Positive internal states can be taught, learnt or pre-programmed into the agent.

There are many different ways in which to incorporate motivation into an agent (see chapter 9). Robo-CAMAL takes the approach

that mind is a control system. From this viewpoint motivation can be thought of as a control process (Sloman, 1993; Davis, 2001). A motivation control process, or motivator, often includes the following components (Sloman et al., 1994).

- Semantic content, a proposition P that represents a possible state that might be true or false.
- A motivational attitude to P e.g. make true, make false etc.
- A rational if the motivator arose from explicit reasoning.
- An indicator of the current belief about P e.g. true, false etc.
- An importance value such as high, low etc.
- At the deliberative level, a plan or set of plans for achieving a motivator.
- A commitment status such as adopted, rejected, undecided.
- A dynamic state e.g. being considered, nearing completion etc.

Figure 3.5 shows several of the motivational control states that have been proposed so far (Davis, 2001, 2007). A more detailed discussion on these control states can be found in section 9.5.

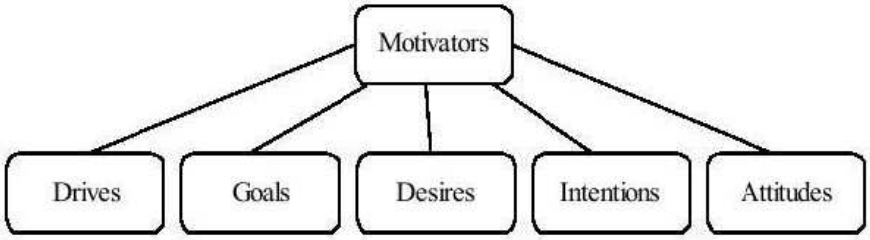


Figure 3.5: Motivational control states (Davis, 2001).

Much of the previous work done on motivation as a control process involved agents in simulated environments (Davis, 1996, 2001; Nunes, 2001; Bourgne, 2003). It is therefore important to see whether this

concept of motivation can translate from a simulated agent to a situated and embedded robot. A more detailed discussion on motivation can be found in chapter 9.

3.8 CAMAL Architecture

The various issues explored in the previous sections are brought together in the CAMAL architecture. This architecture can be seen in figure 3.1 (Davis, 2004).

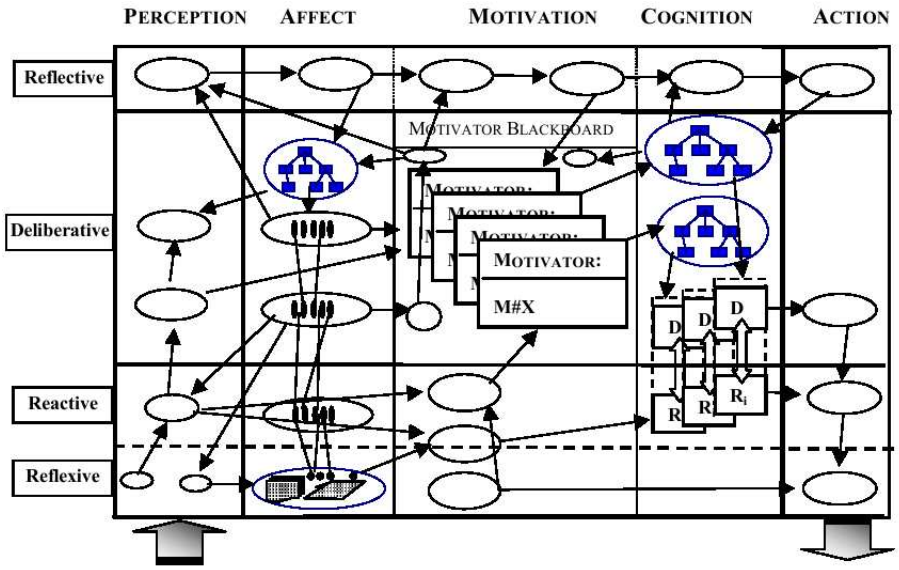


Figure 3.6: CAMAL architecture (Davis, 2004).

The basic structure is that of CogAff (see section 3.3). This includes the perception, central processing, and action columns. It also includes the reactive, deliberative, and meta-cognitive tiers. The various control processes are contained within this matrix. Their location within the matrix is dependent on there function.

CAMAL has been developed to investigate the use of affect and motivation within a cognitive architecture. For this reason the central processing column is divided into three sections.

The affect column contains the various affect models. For example

this could include the distributed affect model as used in a-CRIBB (see section 3.5). The cognition column makes use of various cognitive processes. This can include planning mechanisms such as the BDI model (see section 3.4). The motivation column contains the architectures motivational constructs (see section 3.7). It also contains the motivational blackboard which is at the heart of the architecture.

The motivation blackboard controls the flow of information through the architecture. It also coordinates the various different control processes. It is this blackboard that allows the various affect models to update their values. It also allows the cognitive processes access to the information they require to carry out their task.

Robo-CAMAL is a subset of the CAMAL architecture. Robo-CAMAL differs in that it does not contain the meta-cognitive tier. Also the affect model is a simplified version of the one found in a-CRIBB. However, the BDI model central to CRIBB and a-CRIBB remains. The motivational blackboard contained in CAMAL is also at the heart of robo-CAMAL.

3.9 Summary

This chapter covered previous work that has a direct impact on the design of robo-CAMAL. It described the framework within which the architecture was designed. It then went on to discuss the CogAff architecture, the belief desire intention model, the CRIBB and a-CRIBB models, blackboard systems, and motivational constructs. Finally the CAMAL architecture was discussed. This also included how robo-CAMAL fits in to the CAMAL framework.

Chapter 4

Robot Architecture Overview

4.1 Introduction

This chapter describes the robo-CAMAL architecture that has been developed to investigate the anchoring problem. It looks at the platform on which robo-CAMAL has been developed i.e. its environment and the robot. It then gives a brief overview on how the cognitive architecture controls the robot. Finally, it details the specific technical issues that need to be addressed in order to achieve the aims of the thesis.

4.2 Architecture Platform

Robo-CAMAL is a situated and embodied agent. This means that the agent's environment and its physical body play an important role in its cognitive processes. This section will describe the environment in which robo-CAMAL operates, and its physical body.

4.2.1 The Robot

The robot used is an amigobot (Activemedia, 2000) and can be seen in figure 4.1. It has a two wheel drive with a rear stabilising wheel. It senses the environment through an array of eight sonar sensors. Four

facing forward, one placed on either side, and two at the rear. In addition to this an omnidirectional vision system has been attached. This allows the robot a 360° field of vision (see chapter 5). The amigobot can pass all internal readings such as sonar readings, battery level etc. to a desktop computer via a radio modem. All the vision data is passed via a USB cable (figure 4.2).

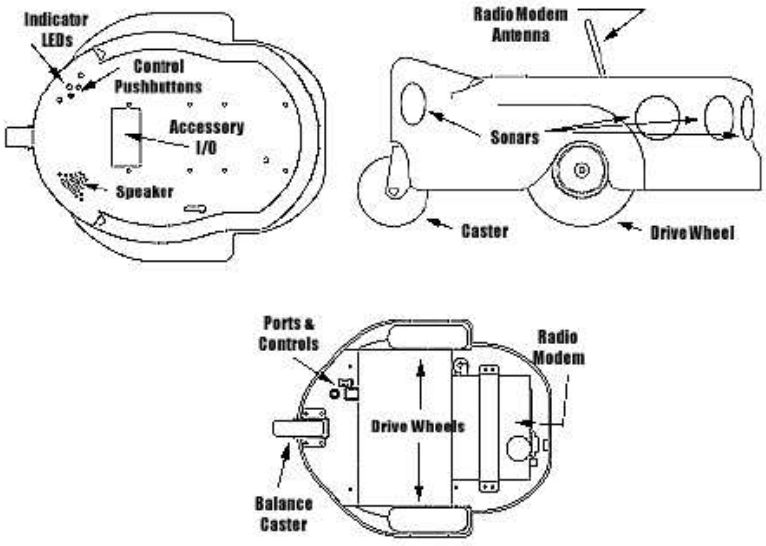


Figure 4.1: The Amigobot (Activemedia, 2000).

The user can control the amigobot using the ARIA software suite (Activemedia, 2003). ARIA consists of a number of algorithms implemented in C++. They allow a user direct access to the robot's internal values such as sonar readings, robot heading etc. This allows routines to be written that use the information to control the robot's actions. The control messages are sent to the robot via the radio modem. As well as the algorithms already present, various new ones have been developed. These algorithms integrate new behaviours and the vision system into the software package (see chapters 5 and 6).

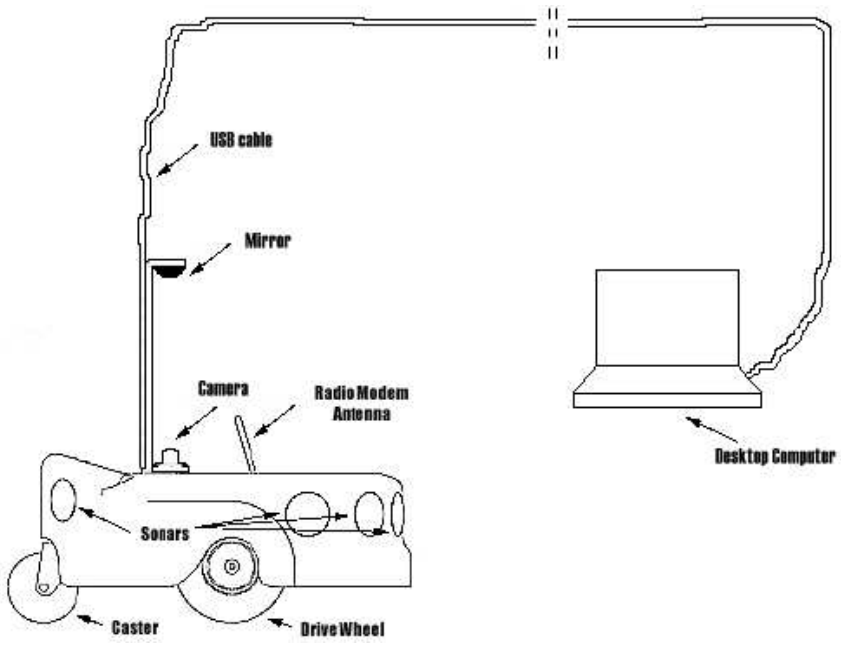


Figure 4.2: Amigobot with onmidirectional vision system added.

4.2.2 The Environment

The robot's environment is an enclosed area approximately two metres square. It has a one metre partition wall roughly halfway along one side (see figure 4.3). The environment consists of various possible objects; a *blueball*; a *redrobot*; or a *blackrobot* (see figure 4.3). The black tape at the base of the walls is to aid the vision system. This increases the contrast between the walls and the floor.

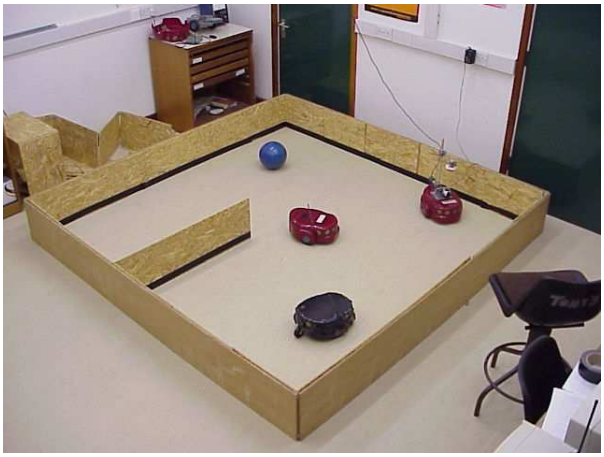


Figure 4.3: The robot's environment.

4.3 The Architecture

The architecture consists of two main components, the reactive level and the deliberative level. A schematic of the architecture can be seen in figure 4.4.

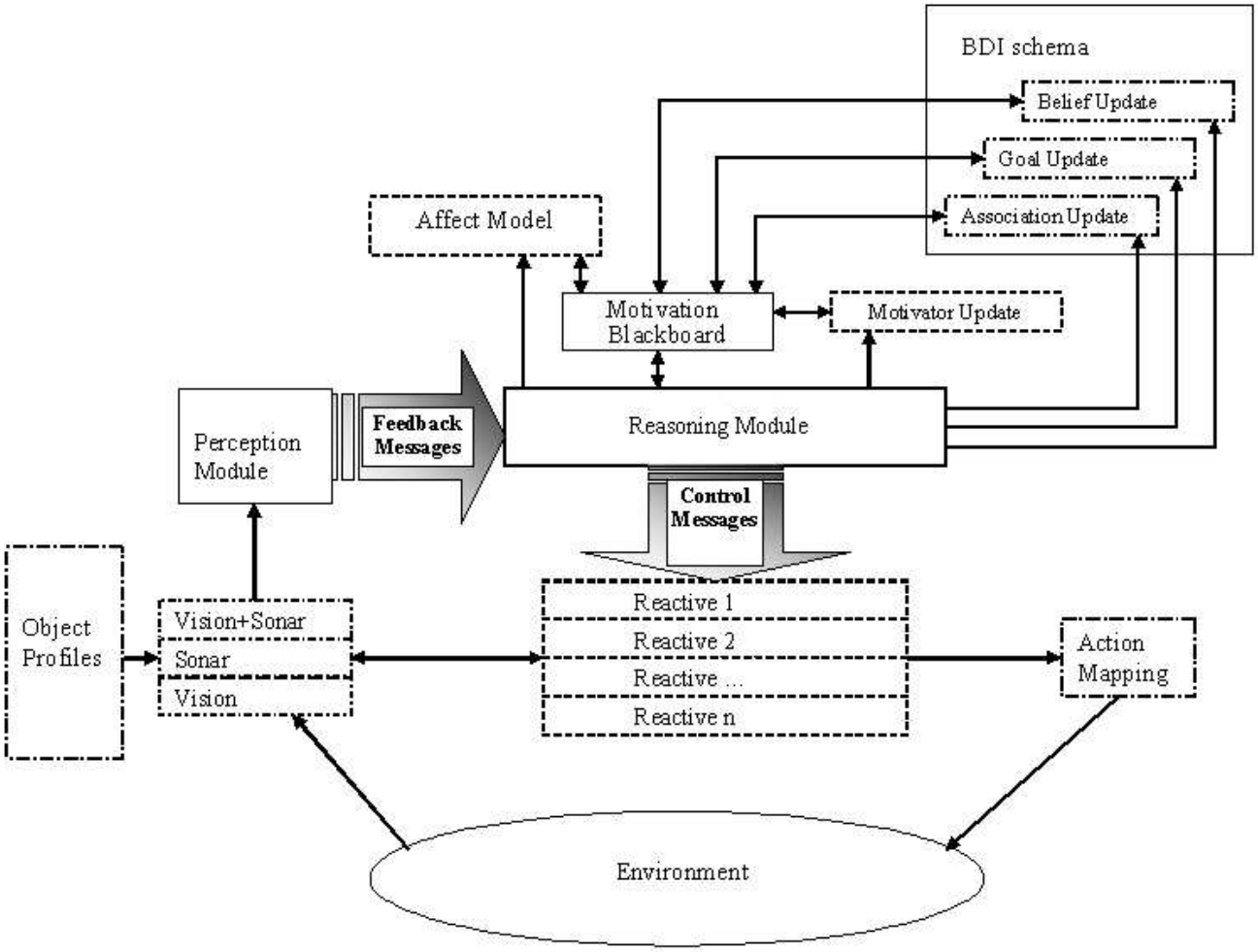


Figure 4.4: The robo-CAMAL architecture.

The reactive level consists of multiple micro-behaviours. A number of micro-behaviours have been designed to achieve the same goal using different methods. For example, one micro-behaviour uses sonar to avoid objects on the right, where as a second uses the vision system to achieve the same goal.

The micro-behaviours can be grouped in specific ways to produce

macro-behaviours. For example *hit(ball)* or *avoid(objects)*. The micro-behaviours can be combined using four different arbitration methods. The four arbitration methods, plus the three possible perceptual modes (sonar, vision, sonar and vision), means each macro-behaviour has twelve different ways of achieving its goal.

Each macro-behaviour and its different possible methods are referred to as individual reactive architectures. This is not strictly true as they are all part of a larger architecture. This method of labelling has been done for clarity. For more details on the reactive component see chapter 6.

The specific reactive architecture used is determined by the deliberative component. The deliberative component is implemented in **prolog**. The reactive system is modified by the deliberative component using parameters sent via the control messages (shown in figure 4.4).

The deliberative component is a blackboard system (see section 3.6). It has a motivational blackboard that contains the relevant information. The reasoning module represents the control component. The various update modules and the affect model are the knowledge sources.

The deliberative component works as follows. At the reactive level sensor data is passed into the perception module. The perceptual module uses a rule table (see table 6.2) to determine the appropriate feedback message. The feedback message is passed to the reasoning module which posts it on the motivational blackboard.

The reasoning module then allows the various knowledge sources access to the blackboard in a specific order. First the belief update uses the new information to modify its belief set. Then the goal update uses the new belief set to determine if the current goal has been achieved, and what the new goal is. The association update then uses

the new belief and goal set to determine the relevant associations. This follows the BDI schema as used in a-CRIBB (see section 3.5).

The belief set, goal set, and associations can all be constructed using predicates defined by the domain model. These predicates are contained on the motivational blackboard. At each step the affect model is used to update the relevant affect values. The affect model used is similar to that of a-CRIBB (see section 3.5). The affect values are distributed across the motivational blackboard. The affect module box in figure 4.4 is the knowledge source that manipulates the affect values. Each goal, belief, and association have their own affect value. Finally, the motivator update is called to determine the new motivator. The appropriate control messages are then constructed from the new motivator. The reasoning module then returns control to the reactive component with these messages.

The following example provides a walkthrough of the architecture performing a simple task. The reactive architecture is initially set-up to *avoid(objects)* and *find(blueball)*. The vision system uses the object profiles to determine if an object it has detected is the *blueball*. If the *blueball* is found a feedback message is sent to the reasoning module. This states that a *blueball* has been found. It also states after how many processing cycles success came. The reasoning module updates the motivational blackboard. The reasoning module then causes the belief set to update its self i.e. there is a *blueball* present and it has been found. The beliefs are then ranked in terms of their reliability. The reliability of a belief is determined by the source of that belief. It is ranked according to the affect model.

Once the belief set has been updated, the reasoning module allows the goal update to use the new belief set and the current goal state to chose a new goal. That is: "it was looking for a *blueball* and has now found it". Goals are chosen based on their importance value. As

the current focus of the system is on the *blueball*, all goals associated with the *blueball* have high importance values. As the *blueball* has just been found the importance value of that specific goal is reduced. This means that the goal *hit(blueball)* now has the highest importance value and is promoted. The goal importance is calculated using the affect model.

Once the goal has been chosen, the reasoning module allows the association update to chose the appropriate associations. This is chosen based on the current current belief and goal set. That is the belief the ball has been found, and the goal is to hit the ball. At this point the new association values are determined by the affect model.

Finally the motivator update uses the current goal and the relevant associations to update the motivator. The association with the highest rank is used to update the motivator. The association's rank is determined by its goal, its belief, and its value (see section 8.6). This is because it contains the plan most likely to achieve its goal given the current belief.

The control message is constructed from the new motivator and contains the specific reactive architecture most likely to achieve the agent's goal. The control message also contains the number of cycles the reactive component should run for.

It should be noted that the deliberative component becomes dormant while the reactive component is operating. Though the deliberative component only becomes active when triggered by the feedback message, it still retains the values of its previous state before becoming dormant. At the reactive level the deliberative component is called in much the same way as a micro-behaviour. This method of implementation proves less time consuming than having a deliberative component constantly monitoring the reactive level.

4.4 Objectives and Problems

This thesis looks to answer several questions (see section 1.2). However, in order to answer these questions several technical problems need to be overcome.

- The integration of a vision system into the architecture (chapter 5).
- The development of the reactive component (chapter 6).
- The adaptation and integration of previous work into robo-CAMAL (chapters 8 and 9).
- The development of an interface between the reactive component (C++) and the deliberative component (prolog).
- The development of a domain model with sufficient information to achieve the aims of the thesis (chapter 7).
- The development of experiments to test robo-CAMAL (chapter 10).

These issues are addressed in the subsequent chapters.

4.5 Summary

This chapter gives an overview of the robo-CAMAL architecture. It provides a basic summary of the robo-CAMAL architecture, which shows how the reactive and deliberative components can work together to achieve goals, without competing for computational resources, or control of the robots final actions. It describes the environment in which robo-CAMAL operates, and the physical body of robo-CAMAL. It then goes on to give a description of how the architecture is constructed. It shows how the deliberative blackboard system is used to

control the BDI schema, and how the affect model is integrated. It finally looks at some of the technical problems that need to be overcome.

Chapter 5

Vision System

5.1 Introduction

For any autonomous agent to interact with, or react to, its environment, it must first perceive that environment. In addition to the sonar array robo-CAMAL has been fitted with an omnidirectional vision system.

Autonomous robot navigation using omnidirectional vision is an area that has attracted a great deal of attention (Adorni et al., 2002; Okamoto and Grassi, 2002) as it provides a simple way to obtain a 360° field of view.

This chapter will describe the onnidirectional vision system used by robo-CAMAL, and how it works. It will then go on to describe some basic image processing techniques that can be used to analyse the image provided by the image system. It will also describe the method used by robo-CAMAL to detect an object using its vision system. Finally, it will describe the technique used to identify an object found by the object detection algorithm.

5.2 Omnidirectional Vision

Omnidirectional vision systems are optical systems that provide a 360° field of view. The first omnidirectional vision system was conceived in 1843 (McBride, 2005) and consisted of a rotating camera. Since then, omnidirectional cameras have been redesigned and successfully used for systems that require a large field of view such as surveillance (Sogo et al., 2000) and 3D reconstruction (Gaspar et al., 2001; Sturm, 2000).

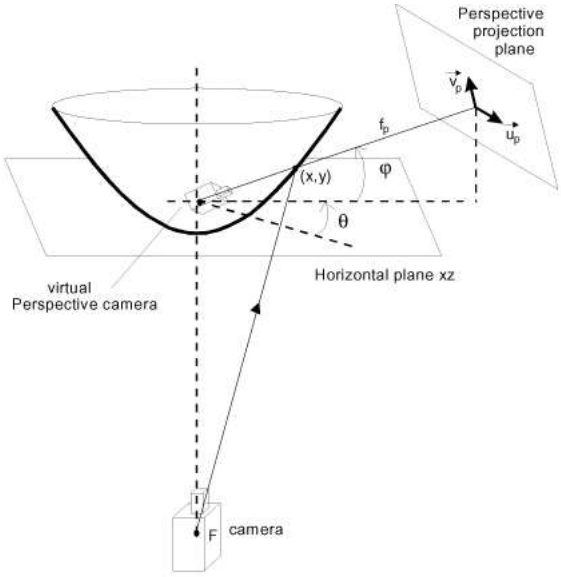


Figure 5.1: Omnidirectional vision system.

The single camera - single mirror system (see figure 5.1) is the option implemented here. This has been chosen for its simplicity, and its low cost. The mirror used is a spherical mirror, and was chosen again for reasons of simplicity and cost.

Figure 5.2 shows an image taken using the vision system installed on robo-CAMAL, and allows robo-CAMAL to view the area surrounding it.

Figure 5.2: Image from robo-CAMALs' vision system.

5.3 Basic Image Segmentation

Once an omnidirectional image has been captured by the camera it needs to be interpreted. One important tool used in image interpretation is image segmentation. The main goal of image segmentation is to define regions within the image that correlate with objects in the physical environment. This section will introduce a few basic techniques for image segmentation. It will then go on to describe in more detail the Sobel edge detection method which is the basis of the segmentation model chosen for robo-CAMAL.

Thresholding is a very simple segmentation method that uses the greyscale image intensity to determine if an object is present (Sonka et al., 1993).

$$g(i,j) = \begin{cases} 1 & \text{if } f(i,j) \geq T \\ 0 & \text{if } f(i,j) < T \end{cases} \tag{5.1}$$

A threshold T is defined. If the value of an image pixel $f(i,j)$ is less than T, the resulting pixel value of the segmented image is set to 0. If the value is greater than T the segmented image pixel value is set to 1 (see Equation 5.1). This is a very heavy handed

approach to segmentation that has problems if the intensity values of the objects in an image are similar to the background. This method can be advanced to use a threshold for each of the three RGB values of a colour image. Though this adds an extra feature with which to segment the image, it still suffers from the same problem; this time with an inability to identify objects of similar colours. The use of more sophisticated threshold parameters can aid the segmentation, for example, by ensuring the standard deviation of the pixels surrounding the target pixel is below a specific threshold; the false alarm rate can then be reduced.

Region based segmentation algorithms construct regions directly as opposed to finding the border between two regions (Sonka et al., 1993). Each pixel starts as a region. Criteria for merging two regions are defined, for example if the two regions have a similar standard deviation or mean intensity then they will merge. If two regions meet the criteria for merging, they do so. This method has difficulty finding an accurate boundary between two regions. This is primarily because the features measured across boundaries often vary which can lead to incorrect decisions made on the merging of two regions.

Edge-based segmentation processes images using edge detection operators (see 5.4 for description of several edge operators) on a greyscale image (Sonka et al., 1993). Edge detectors work by measuring the rate of change in the intensity of the image in a local neighbourhood. The direction of the edge θ is simply at a 90° angle from the direction of the maximum intensity gradient (Equation 5.2).

$$\begin{aligned} |grad| &= \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \\ \theta &= arg\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \end{aligned} \tag{5.2}$$

As images are digital, the gradient must be approximated by the difference between the pixel values (Equation 5.3) where n can be any integer small enough to approximate to the derivative.

$$\begin{aligned} grad_x &= f(x,y) - f(x+n,y) \\ grad_y &= f(x,y) - f(x,y+n) \end{aligned} \tag{5.3}$$

The edges then need to be joined to produce a linked edge chain that separate the two distinct regions.

5.4 Detecting Objects Within Robo-CAMAL

One of the key constraints in developing robo-CAMAL's vision system is that it must be fast. For this reason the method proposed for analysing the omnidirectional image makes use of a simple threshold technique to construct a linked node chain. The edge detection algorithm used by robo-CAMAL is the Sobel operator. Other edge detection methods were tested, but the Sobel operator proved to be the most appropriate for robo-CAMAL. It proved to be very fast as well as accurate in determining an object edge.

The Sobel operator uses a 3×3 convolution mask about the pixel (Equation 5.4).

$$C_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} C_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} C_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{5.4}$$

By using the convolution masks C_1 and C_3, a measure of the pixel gradient in the horizontal and vertical plane can be found. If the image response to C_1 is i, and its response to C_3 is j then the edge

magnitude and direction are calculated using equation 5.5

$$\begin{aligned} g &= \sqrt{i^2 + j^2} \\ \theta &= \tan^{-1}(i/j) \end{aligned} \tag{5.5}$$

Once the image has been transformed using the Sobel operator, the various edges need to be linked to form a continuous node chain. This chain is then used to separate one area from the next. The algorithm developed here examines an area of interest within an omnidirectional image. The area of interest (AOI) is defined as an area around the robot. In the case of robo-CAMAL, the area is defined by the radius R_{max} such that the maximum amount of the agent's environment is captured within the image. The AOI is extracted with a mask over the robot's position in the image to eliminate it from the image analysis. An example of the AOI extraction performed on a test image can be seen in figure 5.3.

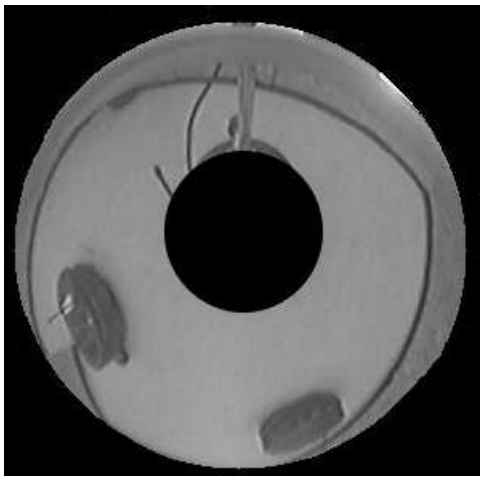


Figure 5.3: Extraction of an area of interest from robo-CAMAL image.

The first issue that needs addressing is the segmentation routine that is to be used on the image. The geometry of the image lends itself to a spherical based node placement algorithm.

The first step is to generate an edge image from the extracted AOI image. This is done using the methods described in the previous

section (see figure 5.4).

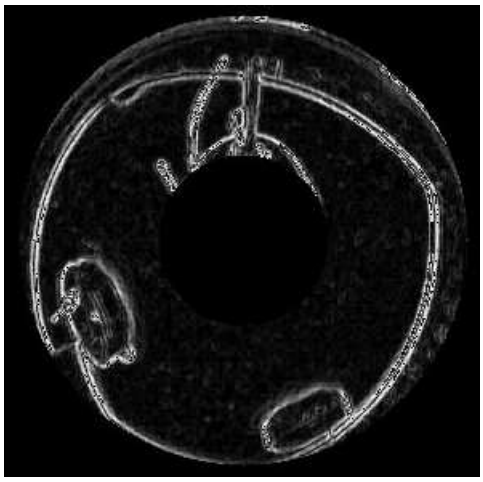


Figure 5.4: Edge image generated using Sobel operator on figure 5.3.

The edge image is then searched in arcs through 360° through a step angle θ. θ is determined purely by the number of segments, n, searched through (Equation 5.6) The greater the value of n, the better the vision systems resolution. This is because there are a greater number of segments analysed for node detection, and therefore better delineation of an objects leading edge..

$$\theta = \frac{n}{360^\circ} \tag{5.6}$$

For each segment a decision is made on whether a node is present. This decision is made based on a threshold technique. If there is an edge magnitude above an experimentally determined threshold, then a node is present. The position of the node is determined by its proximity to the centre of the AOI. The closest point to the AOI's centre that has a value greater than the edge threshold is considered to be the node's position. This is assumed to correspond to the leading edge of any object within the AOI. If a node is found to exist in the adjacent arc then it is linked to the previous node to form a continuous node chain. Figure 5.5 shows the result of the node chain generation.

The use of a proximity threshold is used to determine the leading

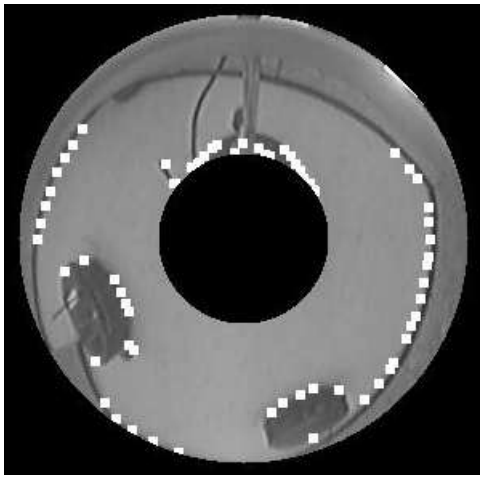


Figure 5.5: Node chain generation using 5.4.

Figure 5.6: Node chain generation using a proximity threshold.

Figure 5.7: Node chain generated using only gradient threshold.

edge of an object and to keep the node chain as continuous as possible. If this is not done and the position is determined simply on the location of the largest edge gradient the node chain becomes discontinuous and jagged. Figures 5.6 and 5.7 show this effect on a test image. Figure 5.6 shows the result of the node placement algorithm if the proximity threshold is used, Figure 5.7 shows the result without a proximity threshold. It can clearly be seen that the proximity threshold produces a more continuous node chain.

The value for the edge threshold was determined experimentally, and the value that appeared to give the best results was chosen. Figure 5.8 shows three test images that use three different edge thresholds. The first is too low and generates nodes that do not correspond to any edge. The second provides a good result for node location whereas the third has the threshold set too high, resulting in several edges being missed.

Figure 5.8: Node chain generated using low, good and high edge threshold respectively.

5.5 Node Placement Algorithm Limitations

Although the algorithm described in the previous section can find the leading edge of an object, it does have some limitations. There are various situations that the algorithm fails to provide the correct information. One problem discovered is that objects on the same

radial line can obscure each other (see figure 5.9). This becomes a problem if the node chain is to be used for identification purposes. However, as other properties are to be used for object identification, this problem is not an issue here.

Figure 5.9: Mug obscuring book on right hand side.

A second problem occurs when the vision system is on a patterned background. Figure 5.10 shows the effect of a tiled floor on the algorithm. This problem is a severe limitation for the vision system and prevents it being used for object avoidance in such an environment.

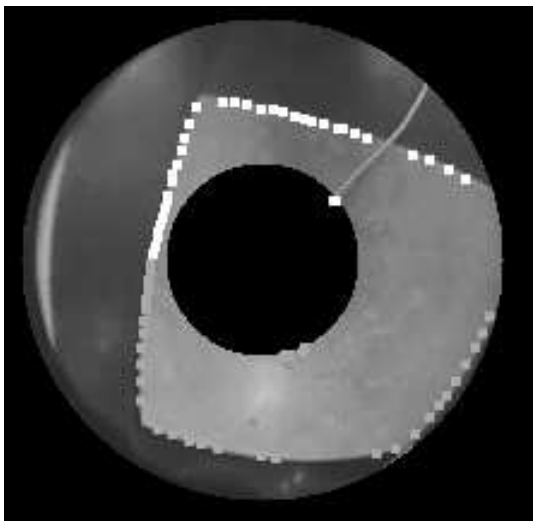


Figure 5.10: Effect of a patterned background on edge detection algorithm.

The limitations of the vision system must therefore be taken into consideration when developing the domain model for robo-CAMAL. The assumption made is that the agent will operate in an environment with an un-patterned floor, and any node chain found will correspond to the leading edge of an object (this assumption makes up part of the domain model which is discussed in chapter 7).

5.6 Object Detection

The second task of the vision system is to differentiate between the different objects within the agent's environment. The algorithm uses colour to differentiate between objects. It was for this reason that the objects used within the agent's environment were chosen based on their colour.

The objects that the vision system can detect are a *blueball*, a *redrobot*, and a *blackrobot*. A colour profile was generated for each object, and was calculated as follows. An image of each object was taken from three different angles. For each image of the object, a sample of 1000 pixels was taken. This gave a sample of 3000 pixels for each object. Each pixel consisted of an RGB value giving three variables. The average value and standard deviation of the pixel values was calculated for each object. The results can be seen in table 5.1. This table represents the colour profile of each of the three objects present within the agent's environment.

Object	**RGB value**					
	Red		*Green*		*Blue*	
	mean	σ	*mean*	σ	*mean*	σ
blackrobot	78.0	3.8	77.5	3.6	75.9	3.1
blueball	79.6	6.0	83.2	6.9	102.9	8.0
redrobot	103.5	4.2	89.7	5.1	79.9	3.9

Table 5.1: The calculated colour profile for the three objects.

The figures 5.11 to 5.13 show the effect of using each colour profile in table 5.1 as a filter on an image containing the three objects. If a pixel value was found to be within 1 standard deviation of the average value for that particular colour profile, it was retained. All other pixels were discarded. Figures 5.11 and 5.13 clearly show the pixels corresponding to the *redrobot* and *blueball* respectively. However, figure 5.12 shows that when the black colour profile is used, a number of pixels corresponding to the *blueball* are retained.

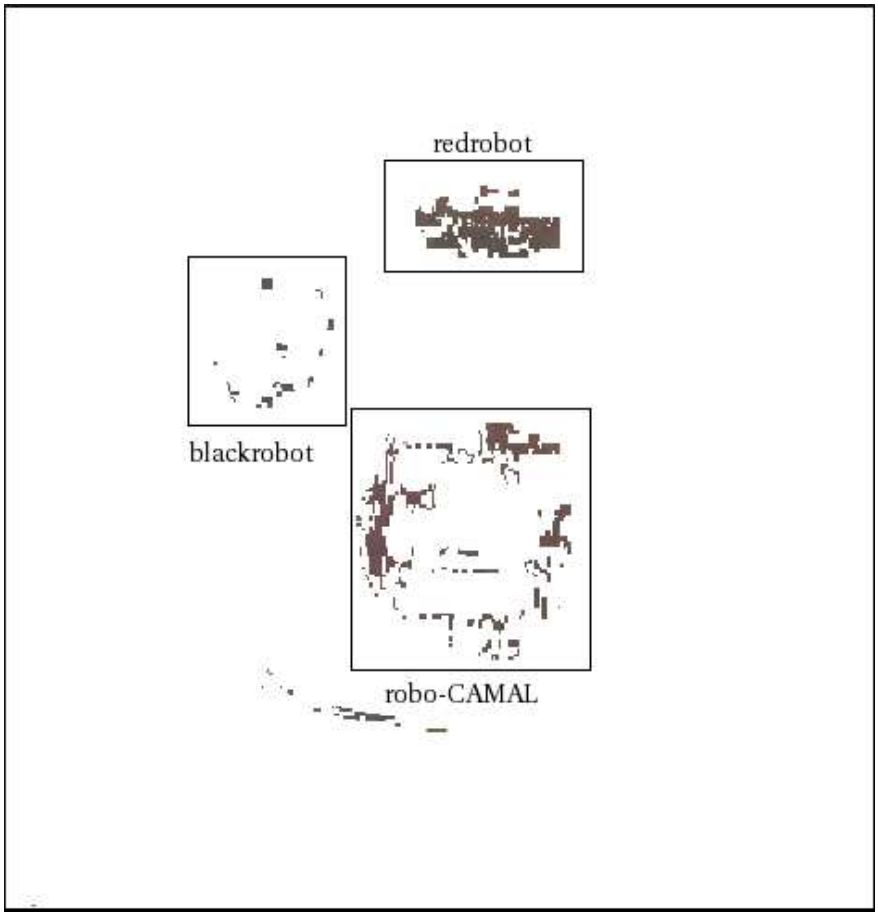


Figure 5.11: Effect of red colour profile mask on robo-CAMAL image.

The actual identification of an object made use of the edge detection algorithm to provide an object area of interest. It is this area of interest that is analysed to identify the object. Equation 5.7 and figure

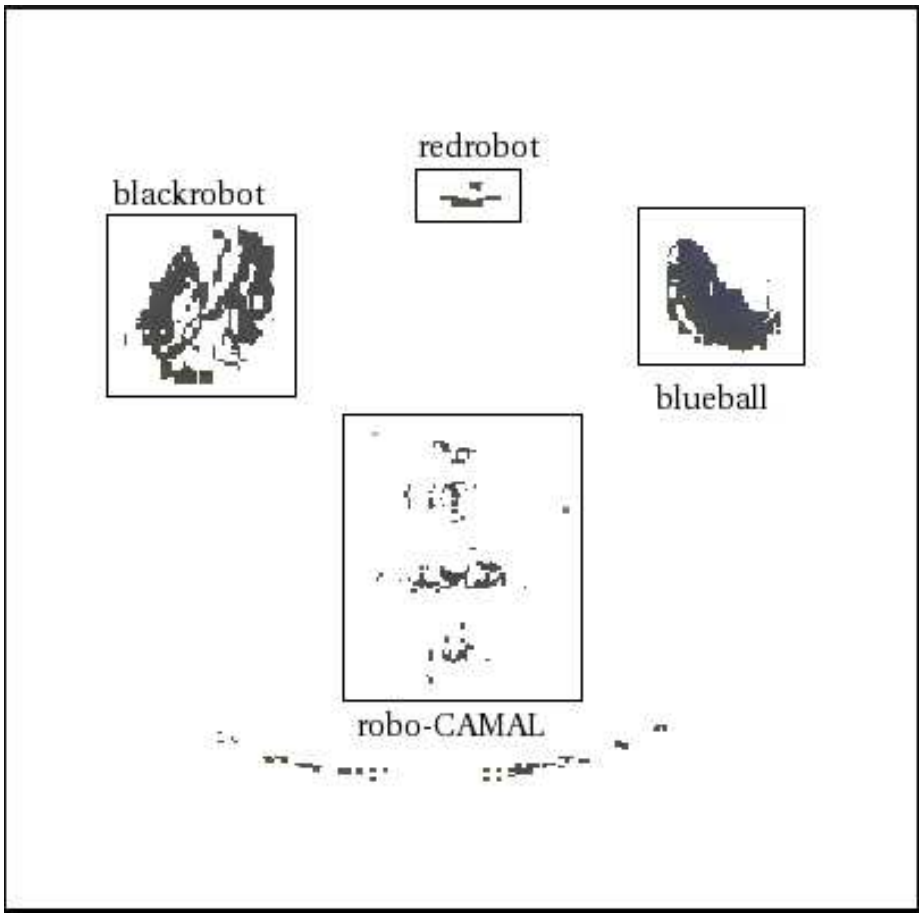


Figure 5.12: Effect of black colour profile mask on robo-CAMAL image.

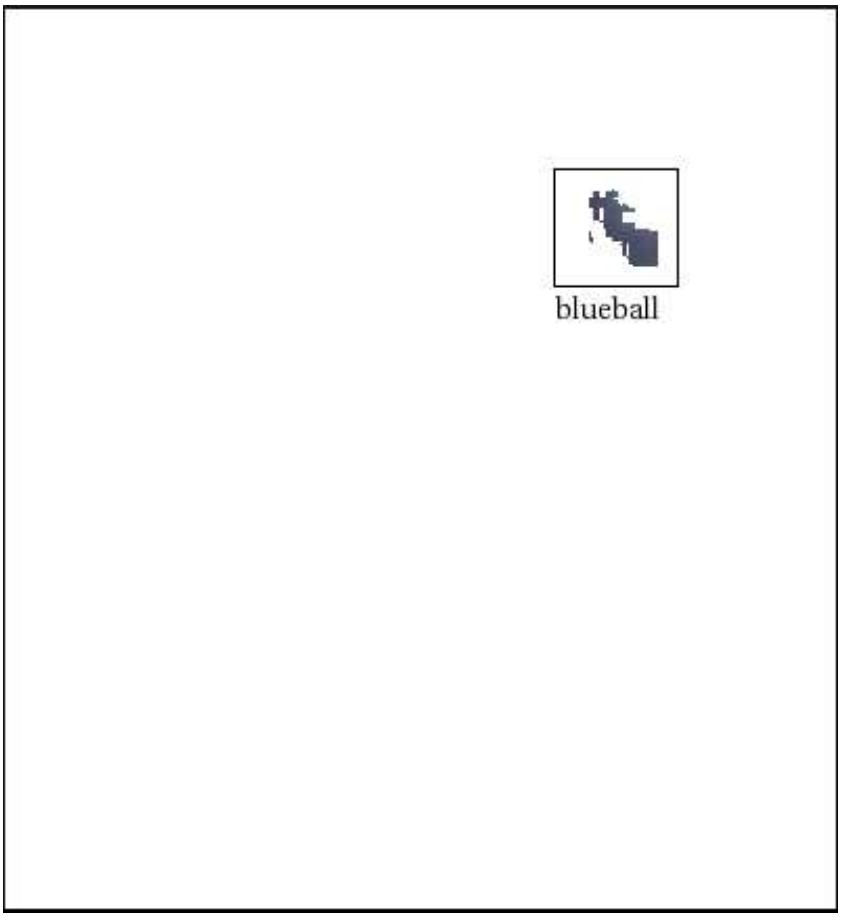


Figure 5.13: Effect of blue colour profile mask on robo-CAMAL image.

5.14 show how the object area of interest (Object-AOI) is defined.

$$Object - AOI = \frac{\theta}{2}(r_2^2 - r_1^2) \tag{5.7}$$

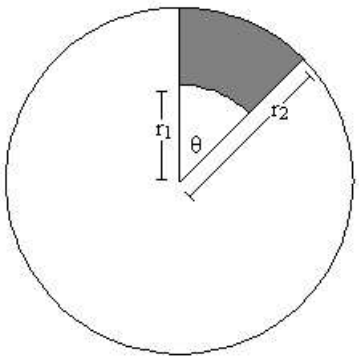


Figure 5.14: The Object-AOI as defined by 5.7.

Here r_1^2 is the radial distance to the node closest to robo-CAMAL. r_2^2 is the radial distance to the edge of the segment and has a value of $r_1^2 + x$ where x is a pre-determined value (for robo-CAMAL x was set to a value of 100 pixels). θ is the angle, in radians, between the first and last nodes in the linked node chain. A graphical representation of the area of interest can be seen in figure 5.15.

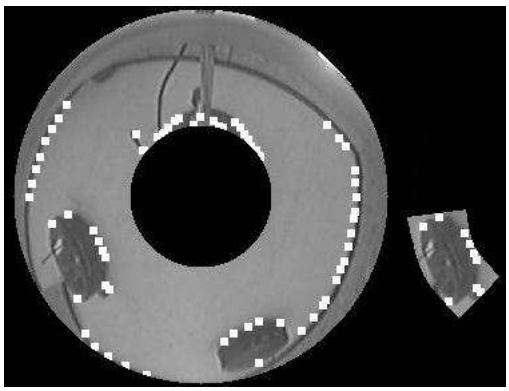


Figure 5.15: The AOI of an image that is used to extract the Object-AOI.

Once the Object-AOI has been extracted, all the pixels contained within it are examined. A pixel is considered to conform to a profile if

its value is within one standard deviation of the average pixel element values seen in table 5.1. This analysis of the Object-AOI produces three values:

- number of pixels conforming to *blueball*.
- number of pixels conforming to *redrobot*.
- number of pixels conforming to *blackrobot*.

The identity of the object within the area of interest is determined by these three values.

- if(*blueball pixels* > *blueball threshold*) then (object = *blueball*)
- if(*redrobot pixels* > *redrobot threshold*) then (object = *redrobot*)
- if(*blackrobot pixels* > *blackrobot threshold*) then (object = *blackrobot*).

The threshold for each object is based on the size of that object. As the *blueball* is smaller than the other objects, and therefore contains less blue pixels, it has a lower threshold value. If the area of interest conformed to two object profiles then the greatest *object pixels* to *object threshold* ratio is chosen. If an area of interest did not conform to an object profile then the object is classed as unknown.

5.7 Summary

This chapter has described what an omnidirectional systems is, explaining why the particular system used here was chosen. It then went on to describe the different ways in which an image can be examined making the case for the use of single linked chain of nodes. It then described the Sobel operator used in determining the node placement of node chain. Finally, the chapter described how the leading

edges were used to identify the various objects within the image with the use of a colour profile.

The results of the image processing described here provide the basis for descriptors used to link the low-level vision system to the cognitive architecture controlling the robot. The assumptions made, and thresholds used, make up part of the lower level domain model (see section 7.2.1).

Chapter 6

Reactive Robo-CAMAL

6.1 Introduction

A reactive robot consists of a coupling between the system's sensors and actuators. A more detailed description of a reactive system can be found in section 2.5.

This chapter will discuss the specific reactive components utilised by robo-CAMAL. It will start off by describing the client/server approach that is used to control the robot. This includes the client software package ARIA. It will then describe the robot's sensor system. This includes a sonar array and an omnidirectional vision system.

The chapter will then introduce the micro-behaviours that have been developed to control the robot's actions. The micro-behaviours each provide a possible action that can be carried out. Therefore a method of determining the correct action from the various possibilities is required. It is at this point that the four possible micro-behaviour arbitration methods are introduced. The chapter then describes how the micro-behaviours can be grouped in different combinations to produce macro-behaviours.

Next the deliberative interface is described. This includes how the perceptual module creates feedback messages for the deliberative component. It also describes how the configuration module uses the

control messages from the deliberative component to alter the reactive architecture. Finally the chapter gives an overview of reactive robo-CAMAL, and how it operates.

6.2 The Client/Server System

The robot uses a client/server control architecture. The robot acts as the server. The client application is implemented on a desktop computer. Information is sent between the client and server using a either a radio modem or a cable leash. The client application is written using the ARIA software suite (Activemedia, 2003). The client processes the information sent by the server and sends the appropriate control messages back to the robot. This process can be seen in figure 6.1.

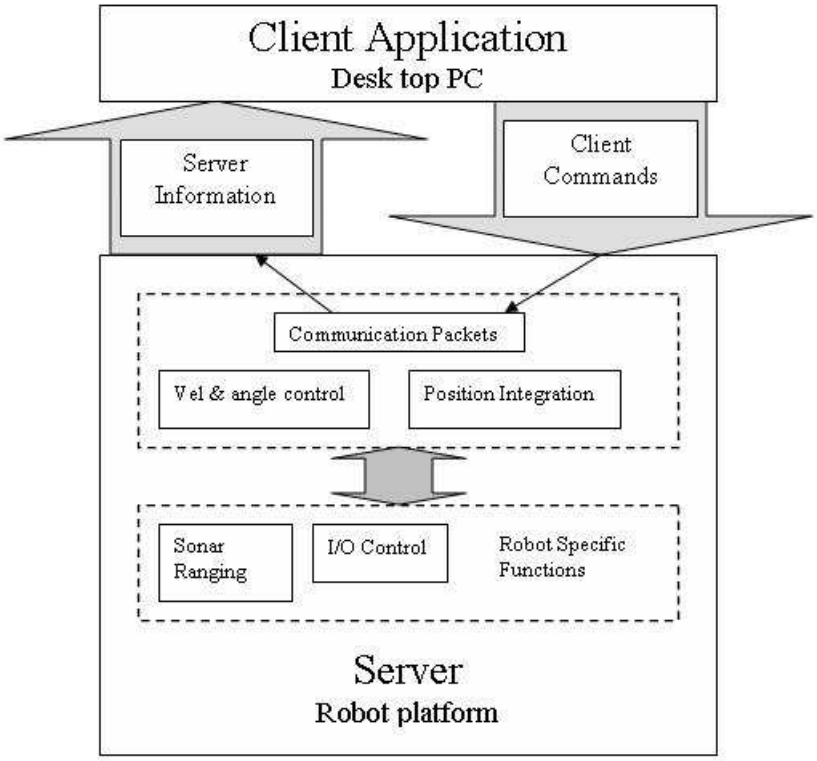


Figure 6.1: The client/server structure for the amigobot.

The robot specific functions collect and process all the information about the robot's state. These include battery level, all the various sonar readings, and the current left and right wheel motor speeds. These values can then be used to calculate several quantities such as the robot's current velocity, the distance to an obstacle, or the robots position relative to its starting point. These values are then passed via the radio modem to the client application.

The client application is written using the ARIA software suite. ARIA consists of a set of functions that access information sent by the server robot platform. The user then defines a set of actions, or micro-behaviours. These micro-behaviours are added to the main robot connection loop. They are fired once every processing cycle. Each micro-behaviour uses the sensor information to produce an action. Each action consists of a desired heading (a value between $\pm\pi$) and speed.

Once the micro-behaviours have produced an action list the resolver module determines the specific action to be sent to the robot. For reactive robo-CAMAL, the original resolver has been modified to include four arbitration methods. The desired action is then sent to the robot via the radio modem in the form of a control message.

Control messages include the desired heading and speed. This information is then used by the robot specific functions to control the left and right motor speeds. The cycle then repeats with the server passing the robot's current state to the client.

6.3 Sensor System

Robo-CAMAL's perceptual system consists of a sonar array and a vision system. The sonar array consists of eight sonar sensors. Four facing forwards, two rear facing, and one on each side of the robot.

The sonar ranging module on the server (see figure 6.1) takes the response from each sonar sensor and converts the value into a distance measurement. This information is sent to the client application via the radio modem. The sonar data is then placed in a memory buffer. The buffer can be accessed by the micro-behaviours. For example the function *getSonarReading*(**X**) returns the distance value in mm of the closest object to sonar **X**.

In addition to the sonar array, an omnidirectional vision system has been developed (see chapter 5). The vision system allows the robot a 360° field of vision. The image is sent to the client via a USB cable. The image is then processed and the results are placed in the vision buffer. The vision buffer consists of three main components. The node locations, the object parameter, and the object location. This provides information on the objects type and location.

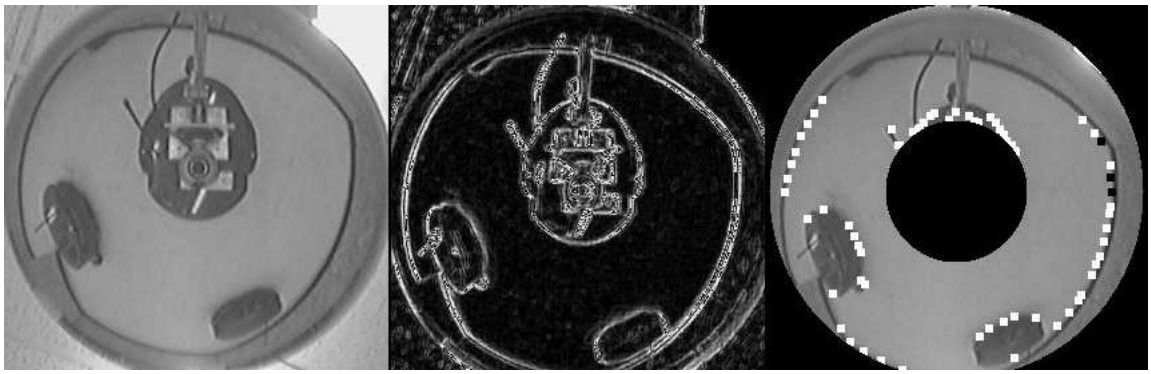


Figure 6.2: Omnidirectional image processing.

The object parameter contains the colour profile of the specific object of interest. This is set using the control message from the deliberative component. The system examines the area behind the closest node from the initial image processing. The colour profile of this area is compared to that of the object parameters. The object location is updated based on this comparison. If the profiles do not match then the object location is set to **NULL**. If the profiles do match then the value is set to the object's location. For a more detailed

discussion of the vision system see chapter 5.

6.4 Micro-Behaviours

Micro-behaviours are actions that can be added to the main robot connection loop. They are defined by the user. The micro-behaviours have access to the sensor buffers. Each micro-behaviour accesses the relevant sensor data once each cycle. It then produces an action based on that data. The micro-behaviours are all reactive in nature. The following is a list of all the micro-behaviours and their function.

- *ConstVel*. Drives the robot in a straight line at a constant speed.
- *VisionAvoidFrontNear*. Stops the robot and turns it away from close objects in front. Object distance determined by the vision system.
- *VisionAvoidFrontFar*. Turns the robot away with a constant linear speed from more distant objects in front. Object distance determined by the vision system.
- *VisionAvoidSide*. Turns the robot away with a constant linear speed from objects to the side. Object distance determined by the vision system.
- *SonarAvoidFrontNear*. Stops the robot and turns it away from close objects in front. Object distance determined by the sonar array.
- *SonarAvoidFrontFar*. Turns the robot away with a constant linear speed from more distant objects in front. Object distance determined by the sonar array.

- *SonarAvoidSide*. Turns the robot away with a constant linear speed from objects to the side. Object distance determined by the sonar array.
- *SonarAvoidRear*. Speeds the robot up if an object is detected to the rear. Object distance determined by the sonar array.
- *IdentifyObject*. Used to determine the identity of an object. Produces no directly observable action.
- *TrackObject*. Turns the robot in the direction of the object present in the vision buffer.
- *HitObject*. Speeds the robot up if the object present in the vision buffer is in front.

The avoid micro-behaviours make use of a proximity threshold to determine their desired action. This is in the form

IF(sensor data < threshold) **THEN**(action = desired)
ELSE(action = **NULL**)

The proximity threshold is simply a predefined value determined by the domain model (see chapter 7). It represents a physical distance within the agent's environment. Any object within that distance triggers the micro-behaviour.

The object based micro-behaviours use data from the vision buffer to interact with the various objects.

Once each micro-behaviour has been fired, it produces a set of actions that it wishes to carry out. The desired actions from each micro-behaviour are placed into an action list. The next step is to choose the appropriate action from the list, and attempt to achieve it.

6.5 Arbitration Methods

Once a list of actions has been created a method of determining the desired action is required. To this end four arbitration methods have been developed. These are the priority, aggregate, winner, and suppression methods.

6.5.1 Priority Method

The priority method requires that each micro-behaviour be assigned a priority when it is initialised. The code

avoid_front(20);
avoid_side(19);
constant_velocity(18);

initialises each behaviour with a priority equal to the parenthesised argument. The higher the number the higher the priority. This value can vary between 1 and 100. A micro-behaviours priority is relative to the other micro-behaviours. Therefore the code

avoid_front(99);
avoid_side(19);
constant_velocity(7);

would produce the same result as the previous example.

6.5.2 Aggregate Method

The aggregate method uses a weighting system to determine the appropriate action. Each behaviour is assigned a weight depending on the response in the perceptual system. This weight value is based on a weighting function. The weighting function for the *avoid_front* and *constant_velocity* behaviours can be seen in figure 6.3.

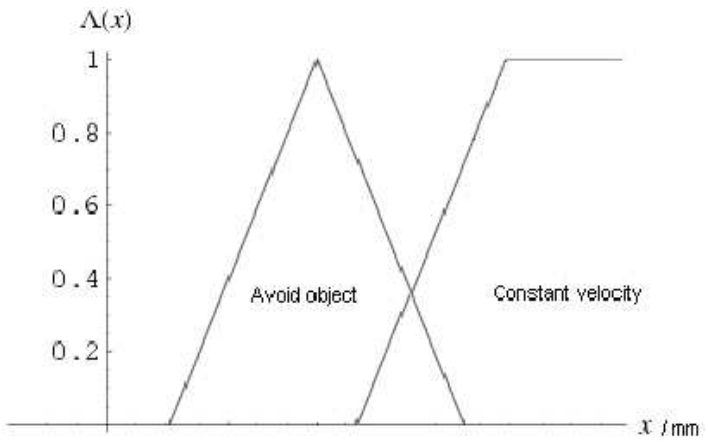


Figure 6.3: Behaviour weighting functions.

This shows that the behaviour's weight is a function of an object's proximity to the robot. For example, in the *constant_velocity* behaviour, the more distant an object from the robot, the greater the weight is. The value of the weights can only vary between 0 and 1. The weighting functions have been designed such that there is always a micro-behaviour with a weight value. This means that there is always at least one active micro-behaviour. The resolver module then examines the weights of all the behaviours. It combines their desired actions based on these weights. This can be summed up by equation 6.1.

$$\frac{\sum_{i=1}^{n} \theta_i \Lambda_i}{n} = \theta_d \qquad \frac{\sum_{i=1}^{n} S_i \Lambda_i}{n} = S_d \tag{6.1}$$

First the individual micro-behaviour headings, θ_i, multiplied by the active micro-behaviours weight, Λ_i, is summed. The desired heading, θ_d, is this sum divided by the total number of active behaviours, n. The desired speed, S_d, is calculated in the same manner.

6.5.3 Winner Method

The winner method uses the same weighting system as the aggregate method. The difference between the two methods is that the desired action is the behaviour with the largest weight. This is also similar to

the priority method with each priority calculated at runtime at each processing cycle. The priority method differs in that it assigns these values prior to the robot's activation.

6.5.4 Behaviour Suppression Method

The final arbitration method uses a behaviour suppression technique. In this method each micro-behaviour has a predefined set of micro-behaviours that it can deactivate. This deactivation occurs when its own conditions are met. For example if the *avoid_front* micro-behaviour is active it deactivates the *constant_velocity*, *avoid_rear* and the *avoid_side* micro-behaviours. The behaviour suppression hierarchy is determined prior to runtime. This method may seem similar to the priority method. However, each micro-behaviour needs to re-activate all the relevant suppressed micro-behaviours once it becomes inactive. This means the robot's reactions to events tend to be slower and far less responsive.

6.6 Macro-Behaviours

Each of the micro-behaviours can be turned on and off individually. This means that different combinations of micro-behaviours can be active. For example, the only active micro-behaviours may be the avoid actions. In this case the robot will remain stationary until an object moves close to it. A grouping of micro-behaviours in this way is referred to here as a macro-behaviour. A macro-behaviour is therefore a specific grouping of micro-behaviours to produce an general overall behaviour. The various macro-behaviours can be seen in table 6.1.

By activating the appropriate micro-behaviours the robot exhibits the corresponding macro-behaviour. As the micro-behaviours can be combined in four different ways (see section 6.5) each macro behaviour

Avoid objects	**I**dentify object	**T**rack object	**H**it object
*C*onstVel	*C*onstVel	*C*onstVel	*C*onstVel
*V*isionAvoidFrontNear	*V*isionAvoidFrontNear	*V*isionAvoidFrontNear	*V*isionAvoidFrontFar
*V*isionAvoidFrontFar	*V*isionAvoidFrontFar	*V*isionAvoidSide	*S*onarAvoidFrontFar
*V*isionAvoidSide	*V*isionAvoidSide	*S*onarAvoidFrontNear	*T*rackObject
*S*onarAvoidFrontNear	*S*onarAvoidFrontNear	*S*onarAvoidSide	*H*itObject
*S*onarAvoidFrontFar	*S*onarAvoidFrontFar	*S*onarAvoidRear	*I*dentifyObject
*S*onarAvoidSide	*S*onarAvoidSide	*T*rackObject	
*S*onarAvoidRear	*S*onarAvoidRear	*I*dentifyObject	
	*I*dentifyObject		

Table 6.1: Macro-Behaviours.

has four possible configurations. In addition, the micro-behaviours in each macro-behaviour use different sensor data. This means, for example, that all the micro-behaviours using sonar can be deactivated. This leaves the macro-behaviour using vision only. Therefore each macro-behaviour has three different sensor modes. Sonar, vision, and both sonar and vision. The three sensor modes and four combination methods means that each macro-behaviour has twelve different configurations. As there are four macro-behaviours there are a total of forty-eight different reactive configurations. For clarity, each such reactive configuration is labelled as an individual reactive architecture.

6.7 Deliberative Interface

This project involves the use of a deliberative component to control a set of reactive architectures. It is therefore important that the reactive component can receive instructions from, and pass information to, the deliberative component. To this end a configuration and perception modules have been developed.

The perception module uses the reactive components internal state to construct a message describing the agents current environment. This message is then passed to the deliberative component. From the reactive component's perspective, the deliberative element can be

thought of as another micro-behaviour. The only difference is that the module does not produce a list of actions as it is not the job of the deliberative component to directly control the robot's actions, only to guide its behaviour. Information is sent to and received from the deliberative component depending on the current state of the robot.

The information received from the deliberative component consists of five elements.

- Sensor type.
- Arbitration method.
- Macro-behavior.
- Object of interest.
- Cycle number.

The configuration module is used to activate the appropriate reactive architecture. It uses the the information provided by the deliberative component to trigger the appropriate reactive architecture. The sensor type and arbitration method are straight forward. They indicate which sensor type and arbitration method should be used; for example, use sonar and suppression, or use vision and priority. The macro-behaviour details which micro-behaviour combination should be active. The object of interest configures the vision buffer's object parameter to reflect the desired object. The final element represents the number of cycles the reactive component should operate for. The reactive element will run for X cycles before calling the deliberative component, unless an event occurs. This means that once the information from the deliberative call has been received, the robot will run with the given architecture until the number of processing cycles is complete, or an event occurs.

It is the perception module that passes feedback to the deliberative component. The perception module checks the sensor buffer every cycle to determine if the robot has encountered an event. It checks to see if the robot has hit anything, if there is an object in the buffer, and if the current cycle value has been reduced to zero. If any of these events occurred, the deliberative component is called and passed two elements of information, the event that caused it to be called, and the object currently in the object buffer. Table 6.2 shows the events that cause a deliberative call and the corresponding messages. Here the current object refers to the object currently in the agents vision buffer. The unknown object refers to an object that has broken the proximity threshold, but does not conform to one of the known object profiles.

Event	**Event message**	**Object message**
Lower proximity threshold violated and object in buffer	hit	current object
Lower proximity threshold violated and no object in buffer	hit	unknown object
Upper proximity threshold violated and object in buffer	near	current object
Upper proximity threshold violated and no object in buffer	near	unknown object
object in buffer	found	current object
Process cycles = 0	avoid	unknown object

Table 6.2: Deliberative message rule table.

Essentially the reactive component will run until one of several things happen. If nothing happens then the number of cycles is decremented by one. When this value reaches zero the deliberative component is called and passed the statements *avoid* and *object*. This indicates that the robot has avoided all objects. If however there is an object present in the object buffer, then the statement *found* with the object value is passed.

There are also two proximity thresholds defined. The lower proxim-

ity threshold defines a distance from the robot within which an object is considered to have collided with the agent. If the lower proximity threshold has been breached then the statement *hit* is passed.

The upper proximity threshold defines a distance from the robot within which an object is considered to be near the agent. If the upper proximity threshold is breached then the statement *near* is passed. Although this is not an ideal solution for determining if the robot has collided with something, it is the most convenient as touch sensors are beyond the resources of this project. If a proximity threshold has been breached, the object message is then dependent on the current state of the vision buffer.

6.8 Reactive Robo-CAMAL Overview

Figure 6.4 shows a more detailed picture of reactive robo-CAMAL. The robot's sensors gathers information which is passed to the sensor buffer via the radio modem and the USB cable. The perception module uses information in the sensor buffer to determine whether an event has occurred. If so, feedback messages are sent to the deliberative component. The deliberative component processes this information and returns a control message. The configuration module uses the control message to determine the appropriate reactive architecture. It configures the sensor buffer to the appropriate object profile. It also configures the resolver module to use the appropriate arbitration method.

After this, the active micro-behaviours fire to produce a desired action list. This list is used by the resolver module to determine the appropriate action. This action command is sent to the robot via the radio modem. The robot then carries out that action.

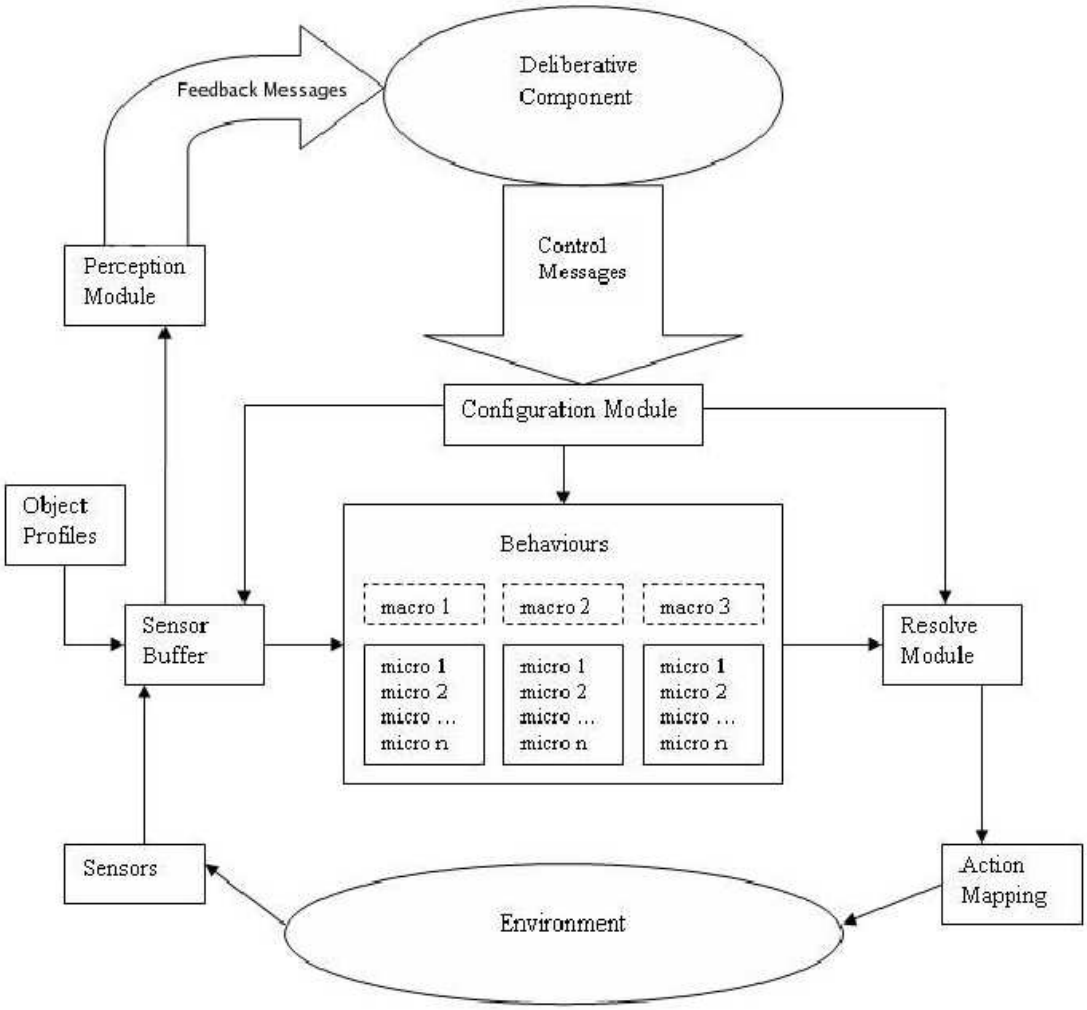


Figure 6.4: Reactive robo-CAMAL.

6.9 Summary

This chapter described the reactive component of robo-CAMAL. It started by describing the client/server system used to control the robot. It then looked at the sensors used by the robot to observe its environment. It then discussed the various micro-behaviours that have been developed, and how those micro-behaviours can be combined. The method of generating more general macro-behaviours was then introduced, followed by how the reactive component interacts with the deliberative component. Finally the chapter gave a brief overview of robo-CAMAL's reactive component.

Chapter 7

The Domain Model

7.1 Introduction

The BDI schema and the motivational blackboard used by robo-CAMAL are generic. It can be implemented upon any agent in any environment. All that is required is a method of instantiating the relevant information about the agent and its environment into the architecture. This is achieved using the domain model. The domain model is distributed across the entire architecture. It provides information on the agent's physical attributes and its environment.

The first part of this chapter will detail the specific information built into Robo-CAMAL using the domain model. It will then discuss some of the issues surrounding the design of a situated and embodied agent, and how the domain model addresses these issues. Finally it will describe the anchoring problem and how the domain model is used to anchor symbols to sensor data within robo-CAMAL.

7.2 The Domain Model

The domain model provides Robo-CAMAL with information about its environment and its physical form. The model is distributed across the architecture at both the reactive and deliberative level. The domain

model can be divided into two main elements.

- Statements about the agent's environment
- Statements about the agent's physical body.

7.2.1 Environment Domain Model

There are several components of the environment domain model present at the reactive level. The first is an assumption made when processing the images provided by the vision system. The use of an edge image to place nodes that represent the leading edge of an object (see section 6.3) assumes the agent operates on an unpatterned floor with no shadows.

The second component provides information on the objects within the environment. There are three distinct objects that can be present in the environment (see section 4.2.2). Each object produces a distinctive set of pixel values at the relative location in the vision system's image (see chapter 5). The response of the vision system to each object can be seen in table 7.1. These values have been hard coded into the agents vision processing module. This allows the agent to compare what is sees with the object profiles. In 7.1 $V_{o,c,1}$ and $V_{o,c,2}$ represent the lower and upper pixel colour channel values, c, within which a pixel is considered to conform to that object profile, o.

Object	**R channel**	**G channel**	**B channel**
blueball	$V_{b,r,1} < x < V_{b,r,2}$	$V_{b,g,1} < x < V_{b,g,2}$	$V_{b,b,1} < x < V_{b,b,2}$
redrobot	$V_{r,r,1} < x < V_{r,r,2}$	$V_{r,g,1} < x < V_{r,g,2}$	$V_{r,b,1} < x < V_{r,b,2}$
blackrobot	$V_{bl,r,1} < x < V_{bl,r,2}$	$V_{bl,g,1} < x < V_{bl,g,2}$	$V_{bl,b,1} < x < V_{bl,b,2}$

Table 7.1: Object profiles.

The third component present at the reactive level can be found in the micro-behaviours. For example, the avoid micro-behaviours contains the reactive control rule

IF(sensor data < threshold) **THEN**(turn)

As the threshold represents a physical distance in the agent's environment it can be thought of as part of the domain model.

Finally there is a perceptual module that passes the appropriate feedback messages to the deliberative level (see section 6.7). The messages are determined by the current state of the agent's perceptual systems, and the rules from table 6.2 that are defined by the domain model.

Statements about the agent's environment at the deliberative level pertain to the possible beliefs the agent can have. These constitute the possible beliefs used in the BDI schema. These statements are present on the motivational blackboard. The environment statements are

environment_descriptor_list([sparse, cluttered, dynamic, static])
clutter_threshold(2)
object_list([object, blueball, redrobot, blackrobot])
object_predicate_list([lost, found, near, hit])

The *environment_descriptor_list* details all the possible states the environment can be in. The *clutter_threshold* determines the minimum number of objects that must be present for the environment to be considered cluttered. The *object_list* provides information on the possible objects present within the environment. The term *object* is used when an unidentified object is detected. The *object_predicate_list* details the possible states the various objects can be in.

These statements allow the agent to construct beliefs about its environment. These take the form

environment(item, source, time)

where $item \in environment\_descriptor\_list$

object_list(object, source, time)

where *object* $\in$ *object_predicate_list*

The source and time terms detail the source of the belief (i.e. assumed, deduced, or perceived) and the time it was formed.

The final statements on the blackboard relating to the environment detail how the various beliefs relate to each other. These are

domain_synonym([X, Y])
negation(X, Y)

The *domain_synonym* states that if the second statement about an object is true, then so is the first. For example, the agent can deduce the belief *found(blueball)* if the belief *near(blueball)* is present. The negation statements details which beliefs are mutually exclusive. For example, The agent should not hold the beliefs *environment(cluttered)* and *environment(sparse)* at the same time. Details on the domain synonyms and the negation model can be found in appendix A.

The final element of the environment domain model to be found at the deliberative level is the value for which beliefs are retained.

belief_retain_time(25)

This represents the number of deliberative processing cycles a belief is held for. This value is related to the length of time a belief is considered valid. For example, the deliberative component may trigger the reactive component to run for 50 cycles. This means that for every one deliberative cycle, the reactive component runs for 50 cycles. Each reactive cycle takes approximately 0.2 seconds to run. If the agent is in a sparse and unchanging environment, the reactive component may run through the complete 50 cycles. This means that a deliberative processing cycle will occur once every 10 seconds. If a belief is removed after 25 deliberative processing cycles then the lifetime of the belief

will be approximately 4 minutes 10 seconds. However, in a rapidly changing environment the deliberative component may be called after only 10 reactive cycles. Under these conditions a belief would be removed after one minute and 40 seconds.

This means that the amount of time a belief about the agent's environment is retained is dependent on the nature of that environment. A rapidly changing environment means beliefs are short lived. A sparse and unchanging environment means that beliefs are long lived.

One interesting point to note is that if the belief retain time is set to 0, the whole system becomes in essence a reactive system. This is because a belief is disregarded as soon as it is formed, which has the effect preventing the association mechanism from forming relevant associations. Once this happens, the agent is unable to couple the correct belief, desire, intention combinations, and is therefore prevented form learning the best course of action given its current belief set.

7.2.2 Physical Form Domain Model

All statements pertaining to the agent's physical body are present at the deliberative level. They are on the motivational blackboard. They provide information on the actions the agent can take and the goals the agent can have. These constitute the intentions and desires used in the BDI schema.

The possible actions refer to the different possible reactive architectures that can be activated. This statement takes the form

architecture(sensor_mode, arbitration_method, macro_behaviour)

The *sensor_mode* represents the three possible sensor modalities. The *arbitration_method* details one of the four possible micro-behaviour combination methods. The macro-behaviour is one of the four general behaviours possible (see chapter 6).

The *desire_list* details all the possible desires the agent can have in relation to the various objects. It takes the form

desire_list([track, hit, avoid])

This can be combined with the object list to give the possible desires the agent can have and takes the form

desire(object)

where *desire* $\in$ *desire_list*
and *object* $\in$ *object_list*

The final component of the physical domain model relates to the goals the agent has. For example, the desire *track(blueball)* has been achieved if the belief *near(blueball)* is present. The *goal_importance* and *goal_threat* are the goal affect values(see section 8.6). They are determined by the predicates

goal_minmax(0.1, 0.9)
goal_threat_max(10)

The *goal_minmax* statement determines the clipping values for the *goal_importance* value. This means the goal importance value will always remain between the values 0.1 and 0.9. The *goal_threat_max(10)* determines the maximum *goal_threat*. This value represents the number of times a goal can fail before its *goal_importance* is drastically reduced. See section 8.6 for more details on the goal affect mechanism.

The final element of the domain model relates to the number of cycles the reactive component is triggered for. This value is determined by the predicates

reactive_cycles(50)

This shows that the reactive component is triggered for 50 cycles. Each reactive cycle takes approximately 0.2 seconds. This means that the agent has 10 seconds to achieve its goal. This value was determined by the size of the agent's environment.

The agent's environment is approximately 4 m^2. The agent's speed is approximately 0.5 m/s. This means that the agent can cover its environment in around 6 seconds. Given this 10 seconds was approximately found to be a reasonable time limit within which the agent can achieve its goal.

7.3 Situated and Embodied Cognition: A Domain Model Perspective

From section 7.2 it can be seen that there are two distinct elements to the domain model. Statements about the agent's environment, and statements about the agent's physical form. These statements reflect the situated and embodied nature of robo-CAMAL. It is situated in that it has sensors that directly detect its environment. It is embodied because it has a physical presence in its environment. This means that robo-CAMAL has to address some of the issues raised by the situated and embodied cognition perspective.

7.3.1 The Situated and Embodied Approach

The situated and embodied approach takes the view that an agent's behaviour and cognitive processes are context dependent. That is an agent's behaviour and cognitive processes are dependent on its environment and physical form (Smith, 1999).

An ant's path across a pebbled beach is complex due to the complexity of the environment, not necessarily the complexity of the ant. This shows that an agent's behaviour is determined to a large extent,

by its environment (Simon, 1996). A bird's ability to jump from a roof and fly is due to the design of its wings. This shows that an agent's behaviour is determined to a large extent, by its physical form.

The situated/embodied approach is often taken one step further and used to simplify the agent's cognitive processes. The view can be taken that an agent's environment is its own best model (Brooks, 1991). This means that the environment can be used to do the agent's computing. For example, an individual riding a push bike does not use the laws of motion to calculate how to shift their weight in order to maintain their balance. They simply shift their weight and use the feedback provided by the environment to determine if they have altered their position correctly. This approach promotes a shift of focus from abstract deduction within an agent's cognitive system, to a behaviour based act and observe system.

This shift in focus works well for reactive agents. The use of the environment as its own model allows a reactive agent to use perceived states of its environment to avoid explicit symbolic representation and reasoning (Brooks, 1997; Agre and Chapman, 1987; Rosenschein and Kaelbling, 1996). If however the agent wishes to make use of symbolic representation and reasoning in order to decide its next action, problems arise. How, for example, is knowledge about the agents environment and physical form instantiated into the agent's cognitive architecture? Also, how do the symbolic representations attain their meaning?

7.3.2 The Frame Problem

The difficulties with incorporating information about the agent's environment and physical form are encapsulated by the frame problem. The frame problem was initially conceived as the problem of how an agent keeps the model of its environment consistent and up to date

(McCarthy and Hayes, 1969; Pylyshyn, 1987).

One example often used to demonstrate the frame problem is as follows (Franklin, 1995). A system is tracking an individual with a key in their hand. The system notes that the individual has moved from room A to room B. How does the system know the key is now in room B.

One solution is the use of a frame axiom that informs the system about what happens to keys that are being held. Now the individual places the key on a table, and moves to room C. Unless a second frame axiom is created that deals with the dropping of keys, the system will assume the key is in room C. A second individual may then pick up the key and throw it out the window. This would require a third frame axiom.

The problem results in a combinatorial explosion for all but the simplest environments. This is because almost any aspect of a situation, under some circumstance, can change. This requires a frame axiom for all possible circumstances in order to handle that change (Dennett, 1984a).

A second solution would be to check the state of the key after every observed change in the environment. This solution has problems if the environment is complex and contains a large number of objects. In this situation it becomes computationally expensive, and impractical, to monitor the state of every object at every moment in time.

The frame problem leads some to believe that there is a deep problem within symbolic A.I. (Pylyshyn, 1987). How does an agent ignore information that is obviously irrelevant to its goals (Dennett, 1987)? For example, an individual attempting to cross a road may know there is a possibility of being struck by a meteorite. However, this information is not used when calculating the risk of crossing the road at a particular point. A second question is how an agent keeps track of

noticeable side effects, without constantly checking its environment (Haugeland, 1987).

There have been many different approaches to find a solution to the frame problem (Shanahan, 1997). However, Robo-CAMAL does not solve the frame problem, it simply circumnavigates it. It does this with the use of the domain model.

For a reactive agent, the frame problem does not exist as there are no symbolic representations. This means that at the reactive level of robo-CAMAL, there is no frame problem. The domain model filters the information that is passed from the reactive to the deliberative level (see section 6.7). This enables robo-CAMAL to ignore information that is irrelevant to its goals.

For example, the agent's goal may be *hit(blueball)*. The vision system is configured accordingly. As the vision system is configured to detect only the *blueball*, the hit micro-behaviour will only react to that object. The reactive architecture will not react to a red robot because of its colour. This means that information concerning the robot's colour will not be passed to the deliberative level. If however, the agent accidentally hits the *redrobot*, then the appropriate event message is constructed and passed to the deliberative component.

In fact, only information relevant to the agent's goal is passed to the deliberative level. This includes the achievement, or failure of its goal due to unforeseen events. This means the agent never uses irrelevant information when reasoning about its goals. This is because it only ever receives relevant information.

The domain filter has an important effect. The reactive component can cope with complex dynamic environments. The domain model filters the relevant information to pass to the deliberative level. This allows the model of the environment at the deliberative level to be a simple one. This means that only a small number of frame axioms are

required.

A second point to note is how robo-CAMAL keeps track of changes within its environment. Robo-CAMAL approaches this difficulty from two different angles. The first is through the use of the domain model. Robo-CAMAL will only check for changes that are relevant to its current goal. These are defined as the success condition within the goal predicate

goal(desire, success condition, importance, threat)

The agent simply checks for the success conditions.

The second approach concerns the frequency with which robo-CAMAL checks the environment. This is determined by the interaction between the reactive and deliberative components. The deliberative component triggers a reactive architecture for X number of cycles, then becomes dormant. The reactive component will only pass information to the deliberative level when a relevant event occurs, or after X cycles have passed.

This means that from the deliberative component's perspective, it checks the environment after every processing cycle. In reality however, due to the nature of the reactive component, the agent checks the environment after a predetermined time period has elapsed, or once a specific event has occurred.

Although the domain model enables robo-CAMAL to cope with the frame problem, it creates other technical difficulties. As the domain model instantiates all the information required about the agent's environment and physical form, very careful consideration must be taken when constructing it. Any inaccuracy or instability in the domain model will have a major effect on the agent's ability and behaviour.

A fixed domain model will mean that the agent's ability in an alien environment will be severely limited. If the agent is able to modify its

domain model, the problem of its construction is shifted up a level. What domain model elements can be changed, and how should they be changed? Finally, the agent could construct its own domain model. This is when the agent runs into the symbol grounding problem (see section 7.3.3).

The problems of the domain model, and their ramifications for robo-CAMAL, are discussed in chapter 11.

7.3.3 The Symbol Grounding Problem

If a cognitive agent uses symbols to reason, how are those symbols generated, and how do they attain their meaning? This problem is known as the symbol grounding problem (Harnad, 1990). A common problem used to highlight the problem is the Chinese room argument (Searle, 1980). An individual that has no understanding of the Chinese language, is isolated in a room. They are given a set of instructions on how to correctly respond to Chinese text. The only form of communication available is by passing them notes written in Chinese. The individual uses the instructions in order to respond, and passes a note back. An observer with no knowledge of the instructions may believe the individual in the room understands Chinese. This however is not the case.

A second more practical example involves **MYCIN** (Buchanan and Shortcliff, 1984). **MYCIN** is an expert system designed to diagnose certain infectious diseases. It uses a knowledge base and production rules to determine the most likely cause of a disease. A rule in the **MYCIN** program may take the form (Clancey, 1997)

IF (AND (SAME CNTXT TREATINF MENINGITIS)
(SAME CNTXT TYPE BACTERIAL)
(SAME CNTXT SURGERY

(SAME CNTXT NEUROSURGERY))
THEN(CONCLUDE* CNTXT COVERFOR TALLY
((STAPHYLOCOCCUS 400)
(STREPTOCOCCUS 200)
(E.COLI 300)))

If a patient has **BACTERIAL MENINGITIS** and has undergone **NEUROSURGERY**, then the most likely causes of the infection are either **STAPHYLOCOCCUS**, **STREPTOCOCCUS**, or **E.COLI**. However if these symbols were replaced with gibberish such as **A75F** or **B16Y**, **MYCIN** would still operate in the same way. The symbols were pre-programmed by the systems designer, and the results are interpreted by a human user.

The previous examples show the need for an agent to attribute meaning to its symbols. If the agent can not attribute meaning to its symbols, then it becomes a mindless symbol manipulator. It also asks the question about how symbols are created. Can an agent be considered to be autonomous if an external system is required to create its symbols?

Robo-CAMAL does not need to solve the whole grounding problem. Within robo-CAMAL, the domain model has been pre-programmed. This means that any symbols used are also pre-programmed. The question of symbol generation goes beyond the scope of this project.

Although robo-CAMAL does not need to generate its own symbols, it does need to attribute meaning to those symbols present. This is achieved through the use of the domain model, and the system's solution to the anchoring problem.

7.3.4 The Anchoring Problem

Symbolic manipulation performed by a cognitive agent should be about something. This means that the symbols need to be coupled to objects or events, to which the agent can attach meaning. For a situated and embodied agent, this means coupling those symbols to information received from its perceptual system (Clancey, 1997; Maturana and Varela, 1992; Harnad, 1990; Wittgenstein, 1953). What is required is a process that connects the symbol system to the sensor data in the correct way (Fodor, 1980, 1985). This process is known as the anchoring problem.

Anchoring is the process of creating and maintaining the correspondence between a system's symbols and its sensor data. The anchoring problem is the problem of how to perform anchoring in an artificial system (Coradeschi and Saffiotti, 2003).

The way in which a symbol is anchored is dependent on the agent's perceptual system. Different perceptual systems generate different types of data. This means that the way an agent analyses its perceptual data is highly dependent on its physical form. For example robo-CAMAL's vision system produces a two dimensional pixel array, where as its sonar sensors produce eight separate sonar values. This data has to be processed in the correct way if links are to be created with the appropriate symbol.

There are many different ways in which symbols can be anchored to perceptual data. One such example uses fuzzy logic (Coradeschi and Saffiotti, 1999). Consider an agent with the task of finding a small red ball. The properties small and red can be represented by two fuzzy sets (see figure 7.1).

The agent's vision system measures the hue and area of objects currently within its sight. Due to the uncertainty and imprecision

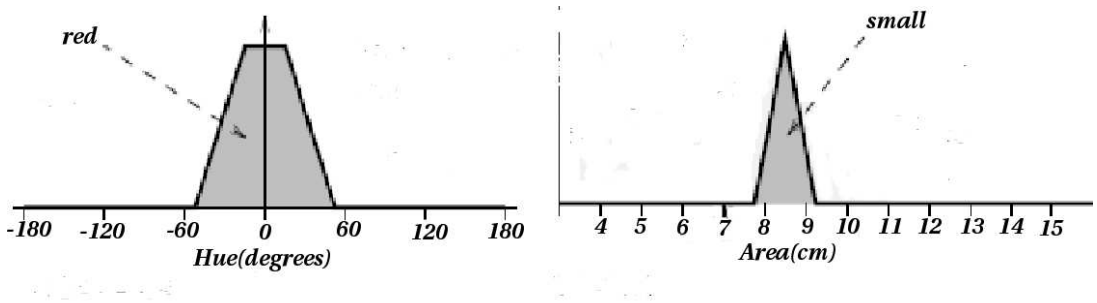


Figure 7.1: Two fuzzy sets representing red and small.

inherent in all perceptual systems, the various properties measured by the vision system can also be represented by a fuzzy set.

By comparing the perceived fuzzy set, A, with the fuzzy set of the required property, B, the degree to which they match can be calculated. For example, let the agent's perceived hue of an object be fuzzy set A. The fuzzy set that represents the colour red is B. The degree to which the two sets match can be calculated using fuzzy set operations. An example of two such operations can be seen in equations 7.1 and 7.2.

$$match_1(A, B) = \sup_{x \in X} \; min\{A(x), B(x)\} \tag{7.1}$$

$$match_2(A, B) = \frac{\int_{x \in X} min\{A(x), B(x)\} \mathrm{d}x}{\int_{x \in X} B(x) \mathrm{d}x} \tag{7.2}$$

Equation 7.1 gives the value at which the two sets intersect. Equation 7.2 calculates the area to which the two sets overlap. This can be seen in figure 7.2.

The degree of matching for each feature can then be combined. This gives an overall degree of matching between the object's perceptual profile, and the desired object's profile. The object with the highest matching value is then linked to the symbol small red ball.

The way in which robo-CAMAL anchors is similar to, but somewhat simpler than, the fuzzy logic method described above. The first

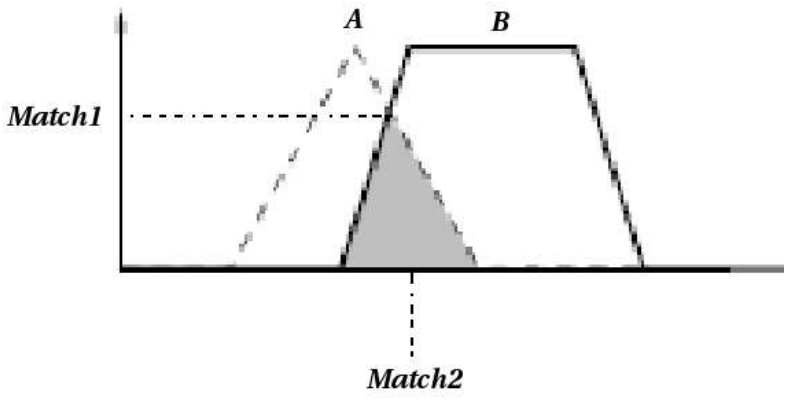


Figure 7.2: Two methods to match fuzzy sets A and B.

point to note is that almost all of the anchoring process within robo-CAMAL is controlled and maintained by the domain model. The symbols that represent the objects are pre-programmed via the domain model. The response of the vision system to the objects is pre-programmed and instantiated via the domain model. Finally, the coupling of the perceptual data to the symbol is controlled by a rule table defined by the domain model.

The way in which robo-CAMAL performs anchoring is as follows. At the deliberative level the agent has a goal that concerns an object, for example *hit(blueball)*. Contained within the control message used to trigger the reactive component is the symbol *blueball*. This symbol is used to configure the agent's vision system.

Different objects generate a specific response in the agent's vision system. The vision system produces a two dimensional pixel array. Each pixel has three values, R, G, and B. Each object has a specific RGB profile. As with the fuzzy logic anchoring example, each object can have a fuzzy set that represents its possible R, G, and B values. An object profile then contains three fuzzy sets. One representing each of the RGB pixel values.

Once the symbol *blueball* has been received by the reactive component, the vision system is configured to search for objects that conform

to the appropriate object profile. However, unlike the previous fuzzy logic example, robo-CAMAL does not generate a fuzzy set from the perceptual data. The vision system examines the pixels of the desired object. If a pixel's *RGB* values conform to the object profile, the pixel is counted. If the number of counted pixels is above a predefined threshold, then the object is said to conform to the object profile. At this point the object buffer is updated with the comparison success information, and the object's location. For more details on how the vision system identifies objects, see chapter 5.

Once a successful comparison has been made, a feedback message is sent to the deliberative component. The message states that an object has been found that conforms to the desired object's profile. In this example that means the *blueball*. Once the deliberative component receives this message, it creates a belief about the *blueball*. Once this belief has been create, the symbol *blueball* is considered to be anchored.

The other important aspect to anchoring symbols is maintaining the anchor. Within robo-CAMAL this is done with the use of the belief management system. The deliberative component creates a belief about the *blueball* when the reactive component's vision system responds in a specific manner. If the vision system looses sight of the *blueball*, the belief about the *blueball* still remains. This is akin to saying that even though the agent can not see the *blueball*, it knows that it is present within the environment.

The way in which robo-CAMAL anchors is in a top-down, bottom-up direction. Information must be passed from the deliberative to the reactive level (top-down), and from the reactive to the deliberative level (bottom-up). Only once the relevant information has been passed both ways is the object considered to be anchored.

The anchor to *blueball* can be broken in two ways. The first is

if robo-CAMAL attempts a task involving *blueball*. If robo-CAMAL fails the task then all beliefs about *blueball* are removed. This has the affect of saying the *blueball* has been lost. The second involves the age of the belief. If a belief is older than the predefined threshold *belief_retain_time*, then it is removed. Once all the beliefs about *blueball* are removed, the anchor is broken.

One drawback in the anchoring process used by robo-CAMAL is that its anchors lack persistence. An anchor has persistence if the agent loses sight of the anchored object, and still knows its location from internally stored position data (Coradeschi and Saffiotti, 2003).

Although information is recorded on the object's location, this is done locally at the reactive level. Location data is not passed to the deliberative component. This means that although the agent may know that an object is present, it only knows its location when looking at it.

Anchoring can also be applied to correspondences between symbols and sensor data that refer to events and actions (Chella et al., 2003; Steels and Baillie, 2003). Again, the anchoring of symbols that represent events and actions are controlled by the domain model.

If robo-CAMAL's perceptual system detects an event that is relevant to the agent's goals, a feedback message is sent to the deliberative component. The deliberative component then generates a belief about that event. For example, if an *object* is detected within a specific distance, then the appropriate feedback message is sent to the deliberative component. The belief *hit(object)* is then generated. This shows the event *hit* has been anchored to a specific perceptual response. The anchoring of events is in a bottom up direction.

At the deliberative level robo-CAMAL has symbols that represent its actions. These symbols are passed to the reactive level via the control message. The action symbol is used to configure the reactive

architecture. The agent's actions are determined by the reactive architecture's configuration. The agent's environment responds to the agent's actions. This response is perceived by the agent, and the relevant information is passed to the deliberative component. The agent anchors action symbols by observing the consequences of those actions on the environment. This process continues in an ongoing feedback loop. This process is controlled by the learning process described in section 8.5.3. These anchors are generated in both a top down and bottom up direction.

7.4 Summary

This chapter looked at some of the issues associated with situated and embodied cognition. It started by introducing the domain model. This is used to instantiate information about the agent's environment and physical form into the cognitive architecture. It then went on to discuss some of the specific problems associated with situated and embodied cognition. First the frame problem was described, as well as how robo-CAMAL circumnavigates the problem with the use of the domain model. It then went on the describe the symbol grounding problem, and the related anchoring problem. The chapter finished by describing how the anchoring problem is resolved with the use of the domain model.

Chapter 8

Associations

8.1 Introduction

Within robo-CAMAL associations are at the core of the BDI schema. Each association contains a belief, desire, and intention combination, as well as an associated affect value. Associations are one of the main factors in deciding the agent's next action at the deliberative level.

This chapter will first describe the association construct. This will include how it relates to the BDI schema and how it is used to decide the agent's next action. It will then describe the association value in more detail. It will then introduce the two applications for the association construct investigated within this thesis. These are how associations can be used to enable the agent to learn about its own behaviour, and how the agent can adapt to its changing environment.

8.2 Association Construct

One of the main components within robo-CAMAL is the BDI schema (see section 3.4). This is where an agent has a belief and a desire that combine to provide an intention or set of plans. The intention is then acted upon to achieve the agent's goal. For example, an agent may be hungry. It may also believe that food is most likely found close to a

tree. The agent's intention would then be to find a tree and look for food. The agent would then act upon this set of plans.

Robo-CAMAL maintains a set of beliefs about its environment. It also has several possible goals that relate to that environment. It also has a number of different plans or intentions that it can act upon. Associations provide a method for robo-CAMAL to keep track of all the possible belief, desire, and intention combinations. They also contain a value that indicates their relevance to the agent at that time.

Each association takes the form

association(belief,goal,architecture,value)

The belief element is simply one of the beliefs the agent can hold. The goal element is one of the possible goals the agent can have. The architecture refers to one of the possible reactive architectures that can be activated. This is the equivalent of the agent's intention. The *value* is a measure of how likely the architecture is to achieve the goal given the belief. For example the association

association(found(blueball),
hit(blueball),
architecture(hit blueball),
value)

should have a higher *value* than the association

association(found(blueball),
hit(blueball),
architecture(avoid blueball),
value)

The various associations can be pre-programmed prior to runtime. However, as there are so many possible BDI combinations it is impractical to pre-program all possible associations. A method has therefore

been built in to robo-CAMAL to generate associations during run time. If the agent comes across a belief-goal combination it has never encountered it generates all the possible belief-goal-intention combinations relevant to the agent at that time. All new associations are initialised with the *value* set at 0.5.

Associations are chosen on based on the current goals and beliefs of the agent. For example, if the agent's current goal is to *hit(redrobot)* then all associations that do not contain the goal *hit(redrobot)* are disregarded. The same is also true for the agent's current belief set. If the belief *environment(sparse)* is not present then all associations containing the belief *environment(sparse)* are disregarded. Finally the most appropriate association is chosen based on the agent's current affect values that relate to its beliefs, goals, and associations (see section 8.6).

8.3 Association Value

As mentioned previously the association value is a measure of the likelihood that a specific architecture will achieve the agent's goal based on its belief. The association value can fluctuate, and is based on feedback from the agent's previous actions.

The feedback mechanism works as follows. The most relevant association is chosen based on the agent's current beliefs, goal set, and affect values. This choice is based on the association value, the time the belief was formed, and the goal importance value. The associations are ranked according to equation 8.11 in section 8.6. The association with the highest rank is chosen. If the two highest ranked associations have the same association value, the first in the list is chosen. The reactive architecture of this association is then activated. Once the reactive architecture has run its course the robot's sensors provide in-

formation on the state of the environment. This information is used to determine if the goal's conditions have been met. If the goal has been achieved the association value is increased. This reinforces the association and increases the chance that it will be chosen again if the same conditions arise. If however the goal has not been achieved, the association value is reduced. This means that it it is less likely to be chosen, given the same conditions.

The association value is increased or reduced according to the equations 8.1 and 8.2 respectively.

$$v_{i+1} = v_i + ((0.95 - v_i) * 0.1) \tag{8.1}$$

$$v_{i+1} = v_i - ((v_i - 0.15) * 0.1) \tag{8.2}$$

Here v_i is the current association value, and v_{i+1} is the new association value. The variation in the association value can be seen in figure 8.1.

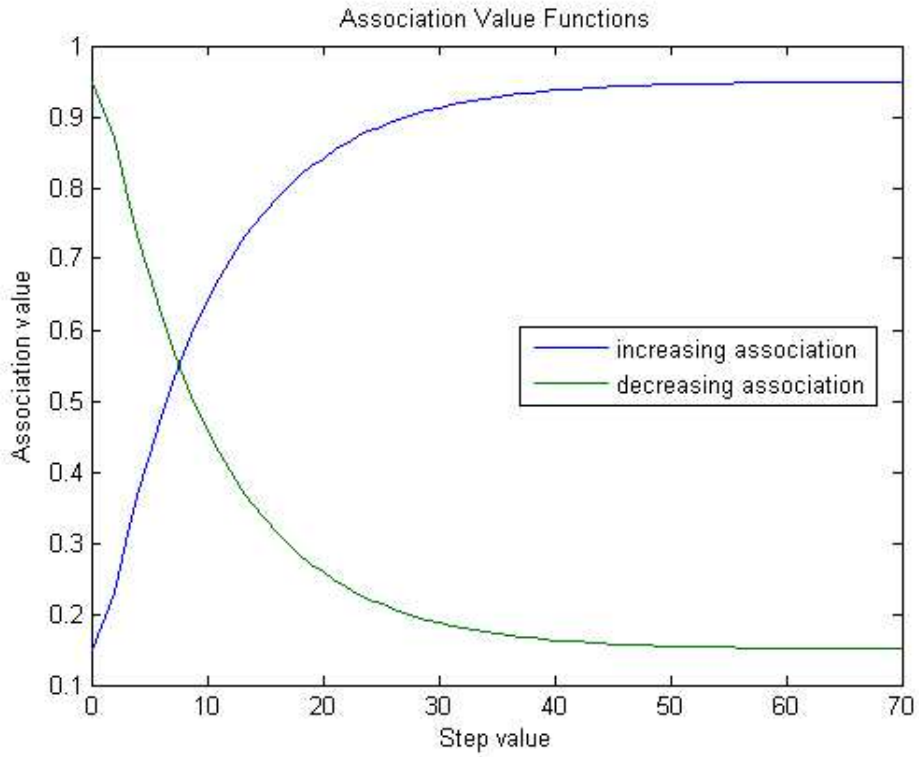


Figure 8.1: Association value functions.

The values have been clipped at 0.95 and 0.15. This prevents the associations from becoming too high or too low. If the environment changes then a specific association may become inappropriate. For example, whilst a *blueball* is present in the environment, an association with the intention architecture*(hit(blueball))* will continue to succeed. However, if the *blueball* is removed the association will fail. If the association value is too high, this association will continue to be chosen long after the ball has been removed. As a consequence the agent will fail to adapt quickly to a changing environment. For this reason the association values have been clipped.

8.4 Association Creation

Associations can be pre-programmed prior to run time. However, this has the effect of controlling which actions are to be used for each belief/goal combination. For example, programming the association

association(found(blueball),
hit(blueball),
architecture(hit (blueball)),
0.8)

links the goal *hit(blueball)* to the specific *architecture(hit blueball).*

If the associations are not pre-programmed then robo-CAMAL has a set of actions with no indication of their purpose. In this case robo-CAMAL needs to generate its own associations. It then needs to test the new associations to determine which is the most appropriate for each situation.

New associations are created as follows. First, all the agent's goals are placed in a list. Each goal is paired with each belief the agent has at that time. Each belief/goal pair is combined with every possible reactive architecture. Each new association has its association value

set to 0.5. This means that for each belief/goal pair 48 new associations are created. For example if the agent has the goal *hit(ball)*, the beliefs *environment(sparse)* and *found(ball)*, the associations created are:

association(environment(sparse),
hit(blueball),
architecture(1...48),
0.5)
association(found(blueball),
hit(blueball),
architecture(1...48),
0.5)

giving a total of 96 new associations.

The following experiment gives a practical example of association creation. The system was set up as follows. First all the micro-behaviours were deactivated. The reactive component of robo-CAMAL was reduced to nothing more than a vision system. The deliberative component was initialised with the goal *find(redrobot)*, and the belief *environment(sparse)*. The reactive cycle number was set to 20. The vision system was initialised to detect an object corresponding to the object profile produced by the *redrobot*. A stationary *redrobot* was placed in front of robo-CAMAL within its lower proximity threshold.

Once the experiment was started, the reactive component provided feedback that the *redrobot* had been found. After one minute the *redrobot* was replaced by the *blackrobot*. At this point associations involving the beliefs *found(blackrobot)* and *environment(cluttered)* were created. After a minute the *blackrobot* was replaced with the *redrobot*. Some of the relevant associations can be seen in figure 8.2.

Figure 8.2 shows several associations with the same goal and intention. Each line represents an alternative belief. Initially associ-

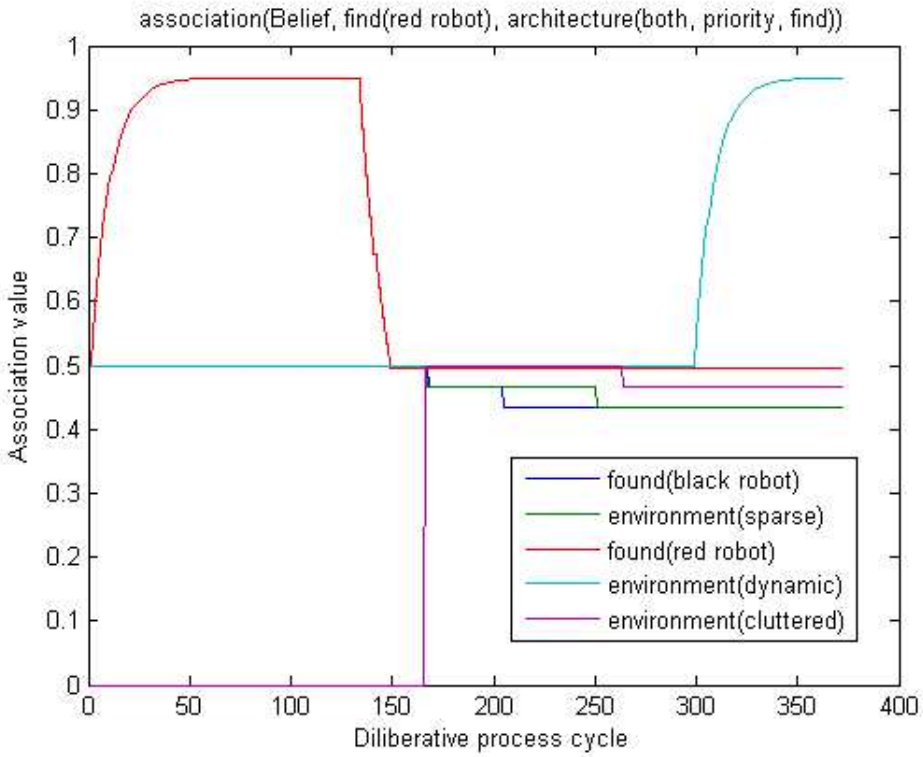


Figure 8.2: Association generation process.

ations with the beliefs *environment(sparse)*, *environment(dynamic)*, and *found(redrobot)* are created. The belief *environment(dynamic)* comes from the domain model assumption that if a robot is present, then the environment is dynamic. Initially the association with the belief *found(redrobot)* increases as it achieves its goal. Once the *redrobot* is removed then the association fails and its association value is reduced.

Once the *blackrobot* is introduced then new associations involving the beliefs *found(blackrobot)* and *environment(cluttered)* are created. The belief *environment(cluttered)* is generated due to the domain model assumption that more than two objects means a cluttered environment. This appears in figure 8.2 as the association values that jump from 0 to 0.5. The gap between the reduction of the *found(redrobot)* association and the creation of the new association is due to the finite time required to swap the two robots. As the reac-

tive cycle number was set at a low value, the time taken to change the robots over is significant. Once the *redrobot* was reintroduced the association with the belief *environment(dynamic)* increases.

This experiment appears to show that robo-CAMAL is successfully learning the appropriate behaviour to achieve its goal. However no hard conclusions can be drawn from this as the experimental set up was so contrived. These results are given purely as an example of association generation.

8.5 Machine Learning Through Associations

Robo-CAMAL makes use of associations in order to learn about the effect of its actions on its environment. This section will briefly look at machine learning, in particular reinforcement and Q learning. It will then go on to describe how robo-CAMAL uses associations to learn.

8.5.1 Machine Learning

Machine learning involves building systems that can use example data or past experience to optimise their performance (Alpaydin, 2004). There are many possible methods from supervised learning, where the system is provided with controlled training data, to unsupervised learning, where the system is given no labelled data and learns by its own means.

There are also many types of algorithms that can be used. One example makes use of neural networks (see section 2.4). The system is given a desired output. Then given a specific input, the weights of the neural network fluctuate until the desired output is achieved (Franklin, 1995). A second involves classification. This is often used in image analysis (Sonka et al., 1993). The system may have various objects to identify within an image. Different objects will have differ-

ent features. The system can be given training images of each object. The system can take a statistical measure of the various features of the object. For example it could measure the intensity values of the pixels corresponding to the object. White objects will have a higher average intensity than black objects. Therefore a distinction can be made between black and white objects based on their average intensity. This demonstrates how a simple law can then be instantiated to classify objects based on their average intensity.

There are many more possible algorithms. However the most relevant to robo-CAMAL is reinforcement learning, and in particular Q learning.

8.5.2 Reinforcement Learning

Reinforcement learning involves an agent performing an action within its environment. It then receives a reward or penalty based on the result of that action. By trying several different actions and using the feedback provided, the agent attempts to learn how to maximise the total reward (Alpaydin, 2004).

A more formal description of reinforcement learning is as follows (Mitchell, 1997). The agent can perceive a set of S states of its environment. The agent can perform a set of A actions. At time t, the agent senses the environment in state s_t. It then chooses action a_t. The environment responds giving the agent a reward $r_t = r(s_t, a_t)$. The environment is then in state s_{t+1}. The task of the agent is to learn a policy π for selecting the next action based on the observed state s_t. That is $\pi(s_t) = a_t$. The aim of this policy is to maximise the reward provided by the environment over time. The cumulative reward is defined as $V^{\pi}(s_t)$. This is the cumulative reward generated by following policy π from the initial state s_t. This value can be calculated using

equation 8.3.

$$\begin{aligned} V^{\pi}(s_t) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned} \tag{8.3}$$

Here γ is a constant between 0 and 1. It controls the weight between immediate and future reward. The higher the value for γ, the greater the weight given to future reward.

From this the agent's task is to find the optimum policy π^* by maximising $V^{\pi}(s_t)$ over all possible states. This can be seen in equation 8.4.

$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\ V^{\pi}(s) \tag{8.4}$$

The cumulative reward from following the optimum policy π^* is $V^*(s)$. That is the value reached by using the best policy π^*, starting from state s. The agent needs to learn the function V^*. The agent should prefer s_1 over s_2 when $V^*(s_1) > V^*(s_2)$. However the agent must choose between actions and not states. In certain circumstances V^* can be used to choose among actions (see equation 8.5)

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s,a) + \gamma V^*(\delta(s,a))] \tag{8.5}$$

Here $\delta(s,a)$ is the resulting state s' of action a in state s.

From equation 8.5 it can be seen that the optimal action in state s is the action that maximises the sum of the immediate reward and the value V^* of the immediate successor state.

However this equation can only be solved if the agent has perfect knowledge of the immediate reward function r, and the state transition function δ. In practice it is impossible for the agent to predict the outcome of every possible state-action transition.

This problem can be overcome by defining an evaluation function $Q(s,a)$. The value of Q is the reward received immediately after

executing action a from state s, plus the value of following the optimal policy from that point on (see equation 8.6).

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \tag{8.6}$$

Rewriting equation 8.4 in terms of Q gives

$$\pi^* \equiv \underset{a}{\operatorname{argmax}}\ Q(s,a) \tag{8.7}$$

This shows that if an agent learns the Q function, it can choose the optimum action with no knowledge of the r and δ functions. Equation 8.7 shows that the agent only needs to consider each available action a in its current state, and choose the action that maximises $Q(s,a)$.

If the Q function corresponds to learning the optimal policy then combining equations 8.4 and 8.7 gives equation 8.8

$$V^*(s) = \max_a\ Q(s,a) \tag{8.8}$$

This can be used in equation 8.6 to give

$$Q(s,a) = r(s,a) + \gamma \max_{a'}\ Q(\delta(s,a),a') \tag{8.9}$$

This demonstrates that Q can be approximated iteratively (Watkins, 1989) If $\hat{Q}$ is the approximation, then equation 8.9 becomes

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'}\ \hat{Q}(s',a') \tag{8.10}$$

This rule uses the current $\hat{Q}$ values for the new state s' to refine its value of $\hat{Q}(s,a)$ for the previous state.

The Q learning algorithm is defined as follows.

The Q learning algorithm

For each s, a pair, initialise the table entry $\hat{Q}(s,a)$

Observe the current state s
Repeat

- Select an action a and execute
- Receive an immediate reward r
- Observe new state s'
- Update the table entry for $\hat{Q}(s, a)$ using equation 8.10
- $s \leftarrow s'$

With this algorithm the agent's estimated $\hat{Q}$ converges in the limit to Q if the reward function is bounded, and actions are chosen so that each state-action pair is visited infinitely often.

8.5.3 Learning Using Associations

The method with which robo-CAMAL learns is very similar to the Q learning approach seen in section 8.5.2. The association value can be written as $A(s, g, a)$, where s is the state, g is the goal, and a is the action. Given a constant goal g, the algorithm for determining the association values is as follows.

Robo-CAMAL learning algorithm

For each s,a pair initialise the table entry $A(s, g, a)$
Observe the current state s
Repeat

- Select an action a and execute
- Observe new state s'
- Use s' to determine if r is positive or negative

- Use r to update the table entry for $A(s, g, a)$ using equations 8.1 and 8.2
- $s \leftarrow s'$

There are however some important differences between the robo-CAMAL and Q learning algorithm. The first is that the reward function is known to the agent. This means the agent can calculate its reward based on the observed new state. The second important difference is that robo-CAMAL is opportunistic. As the goal is constant, the action chosen is the relevant action-state pair with the highest association value. This means if the agent finds an action that provides a positive reward, it continues to execute that action. This is because only the immediate reward is considered. In the robo-CAMAL learning algorithm, the association update equation is the same as the Q update equation with γ equal to 0.

There are several different ways of initialising robo-CAMAL to achieve different levels of supervision when learning. The maximum level of supervision involves pre-programming the associations for a specific goal. For each macro-behaviour there are twelve possible architectures. Given all the associations are pre-programmed and the goal is constant, the agent only has to learn which architecture is the most successful. This is in essence telling robo-CAMAL which architectures should achieve its goal, and asking it to determine the best.

The next step down in the level of supervision is to allow robo-CAMAL to generate its own associations, but control the environment. In this scenario the agent has no indication of what effect each action will have on its environment. However, as the environment contains the correct object with which it can achieve its goal, it should learn

the most appropriate action. This method provides robo-CAMAL with training data in order to learn the best policy.

The minimum level of supervision allows robo-CAMAL to generate its own associations in an uncontrolled environment.

8.6 Adaptation Using Associations

One important requirement for any agent is that it has the ability to adapt to a changing environment. There are several ways in which an agent can adapt, evolutionary adaptation, physiological adaptation, sensory adaptation and adaptation by learning (McFarland, 1991). Evolutionary adaptation occurs when agents adapt to their environment over many generations via natural selection. Physiological adaptation refers to the physiological changes that occur in response to changes in the environment. For example, sweating is a response to an increase in temperature in the environment. Sensory adaptation is when the perceptual systems adjust to the strength of the stimulus that ther are sensitive to; for example, when the pupil dilates due to a change in light intensity. Adaptation by learning is a very general adaptation. It can refer to many kinds of things such as learning the quickest way to a specific location, or how best to avoid a predator.

The way in which robo-CAMAL adapts is through the use of associations. As shown in section 8.2, each association contains four elements, a belief, a goal, an action, and a measure of the likelihood of success of the action given the belief and goal. For robo-CAMAL to adapt, it needs to choose the appropriate association. That is, the association that corresponds to its current environment and internal state. This is done using a two stage process.

The first stage involves the agent learning which associations are most relevant for a given action-state pair. For example the associa-

tion

association(found(blueball),
hit(blueball),
avoid(blueball),
value)

will fail to achieve its goal more often than not. It therefore makes senses to remove this association in order to reduce computation time. However the association

association(found(blueball),
hit(blueball),
hit(blueball),
value)

should achieve its goal most of the time. It is the job of the learning process described in section 8.5.3 to determine the relevant associations.

In second stage of adaptation after the training phase, robo-CAMAL has multiple goals in a variable environment. The task then becomes to choose the appropriate association with which to achieve one of its goals. This choice is based on robo-CAMAL's internal state, and its environment.

In order for robo-CAMAL to choose an association, it needs to rank associations in order of their relevance. This ranking is calculated in terms of the agent's belief, goal, and association values. The value of each associations rank is calculated using equation 8.11.

$$Rank = \sqrt{a_v g_v}\ \frac{1}{b_a + 1} \tag{8.11}$$

Here a_v is the associations value, g_v is the association's goal importance value, b_a is the age of the association's belief. Figure 8.3 shows

how the association rank value varies with each of the three affect values a_v, g_v, and b_a. For each line on figure 8.3, two of the terms in equation 8.11 were kept constant, whilst the stated one was varied.

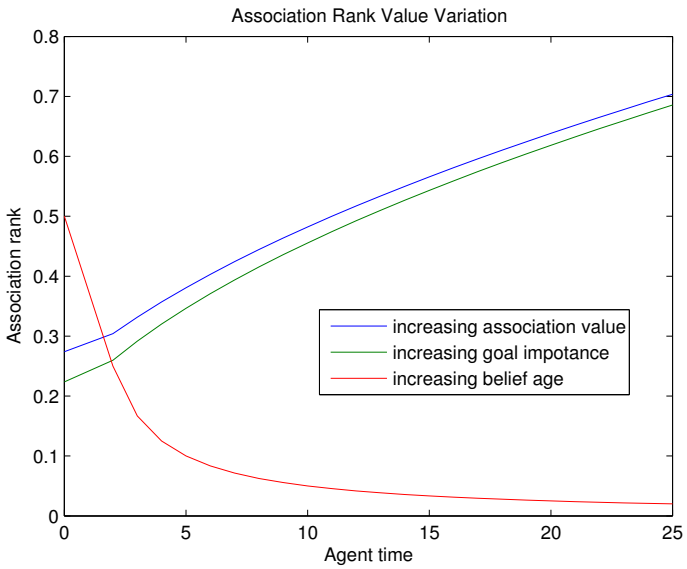


Figure 8.3: Association rank value.

It is clear from figure 8.3 that the rank of an association is based on the distributed model of affect. The rank value increases as a_v and g_v increase. It is also clear that the association rank decreases as the belief gets older. This means that for each association the higher its goal importance, the more recent its belief was formed, and the more likely the action is to achieve is goal, then the higher its rank value.

The association value varies according to equations 8.1 and 8.2. As described in section 8.3, if the agent achieves its goal, the relevant association value is increased using equation 8.1. If the agent fails its goal the relevant association value is decreased using equation 8.2.

The goal importance affect value is initialised at 0.5 for each goal. At each time step the importance value is increased by 0.02, up to a maximum value of 0.95. The only occasion in which the goal importance value is not increased is if the agent has failed to achieve that goal, or if the agent attempted to achieve the goal in the previous

deliberative cycle. If a goal is achieved its importance value is set to 0.5. If a goal consecutively fails X times, where X is determined by the domain model threshold *goal_threat_max*, its importance is then set to 0.1. The goal importance variation can be seen in figure 8.4.

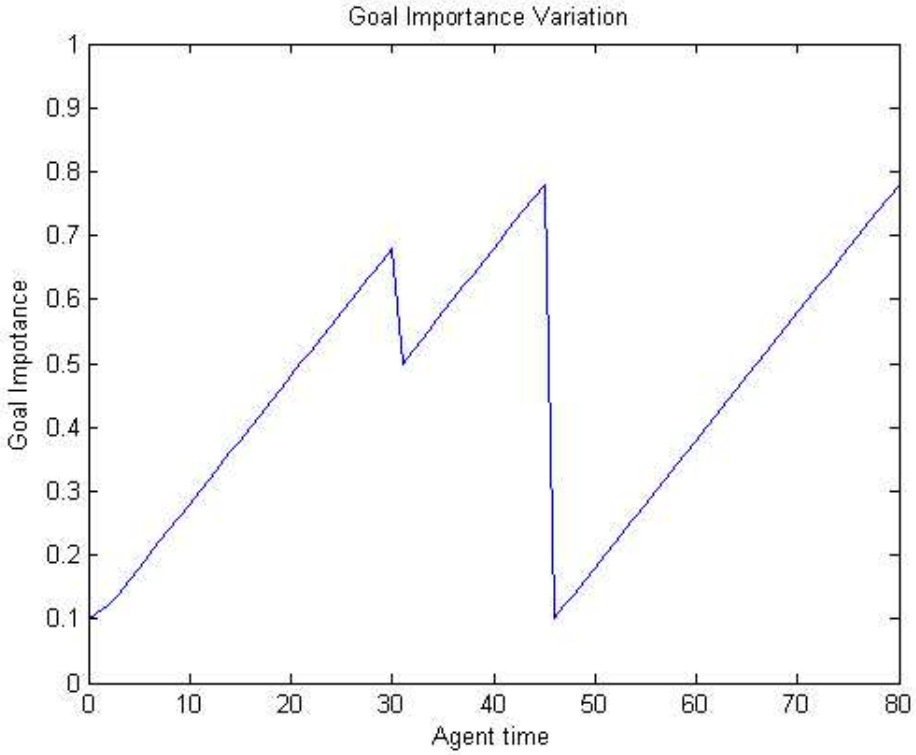


Figure 8.4: Goal importance value.

Figure 8.4 shows a goal with its importance value set at 0.1. Its value increases over time. At this point the goal is achieved, and its importance value is set to 0.5. The importance value then increases again, until the goal consecutively fails X times. At this point the importance value is reduced to 0.1.

The goal importance function has several effects. The first effect to notice is the goal failure strategy. if a goal fails once its value is not automatically reduced. This is because the failure may be due to something simple beyond the agent's control. For example, if the goal is *hit(blueball)*, the agent may simply miss, or a second robot may get in the way. However if robo-CAMAL consecutively fails on a number

of occasions, then it is likely that there is something preventing it from achieving that goal. For example the *blueball* may no longer be present in the environment. If this is the case then the goal importance is greatly reduced to prevent its selection.

The goal's success has two effects on the goal importance depending on its previous value. If the goal importance value was initially low due to a previous failure, its value is increased to 0.5. This reflects that the goal is now achievable. If the goal importance value was initially high, the value is reduced to 0.5. This reflects that the goal has been achieved and is no longer as important. This prevents goals with a high importance value being selected repeatedly.

The increase in the importance value over time reflects the variable nature of the environment. A goal that is unachievable at time t may become achievable at time $t+x$. For example, if there is no *blueball* in the environment, the goal *hit(blueball)* will fail. Its goal importance value will be reduced to 0.1. If a *blueball* is then introduced to the environment, the goal then becomes achievable. Unless the importance value increases then the goal *hit(blueball)* will not be selected. By increasing the goal importance values over time ensures that it is more likely to be chosen as time passes.

The time the belief was formed also effects the association's rank value. The older the belief, the lower the association's rank. This is to reflect the fact that older beliefs may not be as reliable or accurate as more recent beliefs. Beliefs that are older than X cycles, where X is determined by the *belief_retain_time* domain value, are removed. This means they are no longer relevant to the association selection.

This section has shown that agent adaptation within robo-CAMAL can be divided into two components. The first is a training phase. First robo-CAMAL uses the method described in section 8.5.3 to learn the most appropriate associations for each goal. The most successful

associations can then be retained by the agent.

Once the training phase is over, the agent's environment becomes variable, and it is given multiple goals. At this point robo-CAMAL uses its current beliefs, and the various affect values, to choose the most relevant association. The changing beliefs and affect values should mirror changes in the agent's environment.

8.7 Summary

This chapter looked in detail at the association construct. It first gave details on the associations and how they work. It showed how associations are at the core of the BDI schema within robo-CAMAL.

It then went on to show how associations enable robo-CAMAL to learn about the effect of its actions upon its environment. The form of reinforcement learning used by robo-CAMAL was shown to be similar to the Q learning algorithm.

It finally went on to show how robo-CAMAL uses associations to adapt to its environment. It showed that this adaptation is done with a two step process. First a training phase is used to learn the appropriate associations. Then the affect model and the agent's belief mechanism enables robo-CAMAL to adapt to changes in its environment.

Chapter 9

Motivation Within Robo-CAMAL

9.1 Introduction

Motivation is considered to be the driving force behind all the actions of an agent (Beck, 2000). Motivation can not be observed directly, but it can be inferred from the observable behaviour of an agent (Westen, 1996). If motivation is the driving force behind the actions of an agent then it must be incorporated into Robo-CAMAL in some way.

This chapter will first look as some of the perspectives that can be taken when trying to define motivation. It then goes on to give some examples of how motivation can be incorporated into a synthetic agent. The chapter then discusses the view of motivation as a control state, and how this applies to robo-CAMAL. Finally, the use of the motivational blackboard that is used to control robo-CAMAL is described.

9.2 Motivation

The word motivation is derived from the Latin word *movere*, to move. It refers to the moving force that energises behaviour. Motivation can therefore be considered the driving force behind all the actions of an agent (Beck, 2000). There are several issues that any theory of moti-

vation must consider (Westen, 1996). One issue is whether motivation is based primarily on the internal needs of an agent. This can include things such as hunger. Alternatively, is motivation based mainly on external stimuli, such as the presence of food? This essentially asks the extent to which an agent's motivation is based on its physical form, or its environment. Another issue is the relevance of thoughts and feelings to motivation. Can a thought or goal motivate an agent, or does that goal need to be associated with a sense of arousal?

There are several perspectives on motivation that are relevant to this thesis. The evolutionary perspective takes the view that some motivational systems are behaviours that have been selected via natural selection. These inherited behaviours maximise the likelihood of survival of the agent. One example of such a system is instinct. In biological agents instincts are inherited patterns of behaviour, or responses to certain kinds of stimuli (Tinbergen, 1951). With respect to robots this could refer to pre-programmed responses to perceptual data, or internal states.

The behavioural perspective makes use of the concept of drives. All agents have needs such as food, water, energy etc. Unfulfilled needs lead to drives which are states of arousal. It is these states of arousal that motivate behaviour (Hull, 1943). Drive reduction theories propose that motivation stems from a combination of drive and reinforcement (Carlson et al., 2003). Deprivation of need leads to tension in the agent. The agent exhibits behaviour in an attempt to reduce that tension. If the behaviour reduces that tension then it becomes associated with that particular drive. The behaviour is therefore reinforced.

There are two types of drives, primary and secondary drives (Westen, 1996). Primary drives are innate and built in to the agent. Secondary drives are learned. An originally neutral stimulus becomes associated

with a specific drive reduction behaviour and thus becomes a motivator. If this behaviour leads to a situation that allows the reduction of other drives then that behaviour becomes reinforced. For example, an agent may learn to associate trees with food. If the behaviour to look for a tree leads to a situation that reduces its hunger, then the drive to find a tree becomes a secondary drive. However, drive reduction theory may not be able to explain all behaviour. For example, the presence of food can cause a dog to eat, regardless of whether it is hungry or not.

The cognitive perspective focuses on conscious goals. Goal setting theory (Locke, 1996) states that conscious goals regulate an agent's behaviour. Agents have goals that specify desired outcomes that differ in some way from the agent's current situation. Expectancy value theory (Fishbein and Ajzen, 1974) views motivation as a joint function of the value the agent places on a goal, and the extent to which that goal is achievable.

These different perspectives on motivation demonstrate just a few of the ways in which the observable behaviour of an agent can be explained.

9.3 Grounding Motivation

In most cases an artificial agent's motivations are imposed upon it by the designer. However, the agent still has to adapt to its environment, and choose the correct motivation, independently of the designer. If an agent is to show independent motivated behaviour, it must be able to set its own goals . If an agent is to choose its own goals, then it must understand the context of those goals in relation to its environment. In essence this means that an agent's motives must be grounded in its environment. This problem can be viewed as part of the wider

grounding problem on how to integrate an agent with its environment (Savage, 2003).

One way of grounding motivation in an agent is the use of reactive systems. For example, a motor powered by a photosensitive plate may be motivated to seek light. There is an argument that evidence of motivation within reactive systems lies with the observer and is therefore not objective (Sharkey and Heemskerk, 1997). However, if this argument is left to one side then the system's motivation is embodied in the control architecture of the sensing and acting mechanisms (Sharkey and Ziemke, 2001).

If an agent's motivation is based on internal representations, the issue of grounding is no longer as straight forward as for a reactive system. These representations, and therefore the relevant motives, must be connected to the appropriate events/activities in the agent's environment (Savage, 2003). In essence an agent's motivation must be grounded to its environment and actions.

Section 9.2 described some possible sources of motivations such as goals and drives. However, these descriptions seek to explain behaviour rather than focus on the situated and embodied nature of motivation. One example that can be used to explain the situated and embodied nature of motivation is the fungus eater thought experiment (Toda, 1982).

In addition to the agent's drive to eat fungus, its actions are also determined by its previous interactions with its environment. For example, if the agent has been active and not eaten, the drive to eat fungus should increase. On the other hand if the agent has just fed then the drive to eat fungus should diminish. This line of reasoning shows motivation to be an interactive process. That is, motivational grounding is derived from the interaction of an agent with the appropriate aspects of its environment (Savage, 2003).

This section has discussed how motivation can be integrated with the agent's environment using a reactive system, or an interactive process in the case of symbolic motivational states. It has not, however, provided a way of determining whether a motivation is a simple reactive behaviour, or a symbolic representation. The model that describes the indices of motivation looks to provide a way of making such a distinction (Epstein, 1982; Savage, 2003).

This model provides three features that may distinguish between reactive and deliberative motivations. The first is *individuation* which describes an agent's ability to achieve the relevant goal using a number of different strategies. That is, the agent's ability to achieve a goal using an alternative method if its preferred response is blocked.

The second characteristic is the formation of *expectancies* relating to its goal object. This relates to cognitive representations reflecting aspects of the goal object, such as how it will react under certain conditions.

The third characteristic relates to the presence of an *affective* response towards the goal object. For example, the agent has an affect value associated with an object. The agent's affect value changes depending on the agent's interaction with that object.

9.4 Implementing Motivation

There are numerous ways in which motivation can be incorporated into an artificial agent. This section will give an example of two of them. The first is the use of a motivational vector. The combination of an agent's physiological/internal state, the perceived state of the environment and the consequence of the agents current action is considered its motivational state (McFarland and Bösser, 1993; McFarland, 1993). For example, some biological agents have a hunger

drive. Hunger is not based purely on the level of nutrients within the agent. It can also be based on the presence of food in the environment, or the time of day if the agent has a routine. It is therefore possible to represent the agent's motivational state as a vector in a multidimensional space. Each axis within that space corresponds to an event or internal state that is relevant to the drive in question. An agent can be programmed with several motivational vectors. The agent's actions are based on the values of the various motivational vectors.

Another agent that incorporates motivation into its architecture can be seen in figure 9.1 (Stoytchev and Arkin, 2001, 2004). This is a hybrid reactive/deliberative architecture. The deliberative component is used as a navigational planner. The deliberative subsystem plans a route around an office. The architecture has three components that link up to a robot (figure 9.1).

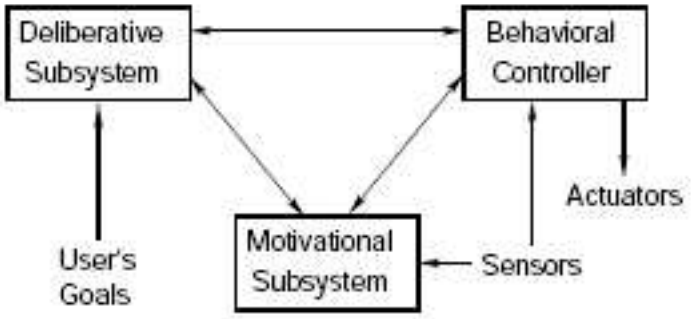


Figure 9.1: Hybrid agent architecture (Stoytchev and Arkin, 2004).

At the centre of the deliberative system is a path planner that takes user defined goals, and a predefined map of its environment. The output of the planner is a set of checkpoints that when followed take the robot to the goal. The sequence of points is used to configure a reactive controller. Within the behaviour controller there is a process that monitors the progress of the active reactive behaviours. By comparing the current progress of the agent to the required goal, the behaviour

controller can determine if good progress is being made. Alternatively it can determine if a new plan needs to be generated. The motivational subsystem is used to monitor the internal state of the agent. This modulates its behaviours using the behavioural controller.

The motivational subsystem contains several motivational variables. These are real numbers that can vary in the range from 0 to 1. The variables are organised in a motivational vector. Each element of the vector can be read or written to independently of the other elements. Perceptual triggers and behaviours can also read or modify the motivational vector.

Perceptual triggers fire a particular behaviour in response to an external event that is perceived by the agent (Arkin, 1998). Within the motivational subsystem, the motivational vector and perceptual stimuli fires the appropriate trigger. The motivational vector can also fire the trigger on its own, allowing behaviours to be fired directly by motivations. For example hunger fires the behaviour to search for food. Behaviour can also alter motivational variables, allowing internal and external events to alter the agent's motivational vector.

9.5 Motivational Control States

The use of control states to develop the robo-CAMAL architecture (see section 3.7) leads to the use of motivational control states. Figure 9.2 shows five motivational control states that have been proposed so far (Davis, 2001, 2007).

Drives are low level mechanisms and refer to the same types of systems as seen from the behaviourist perspective of motivation (see section 9.2). The onset of drives are dependent on variables that fluctuate in response to internal and external processes. If the variable crosses a specific threshold then the drive activates a pre-set behaviour

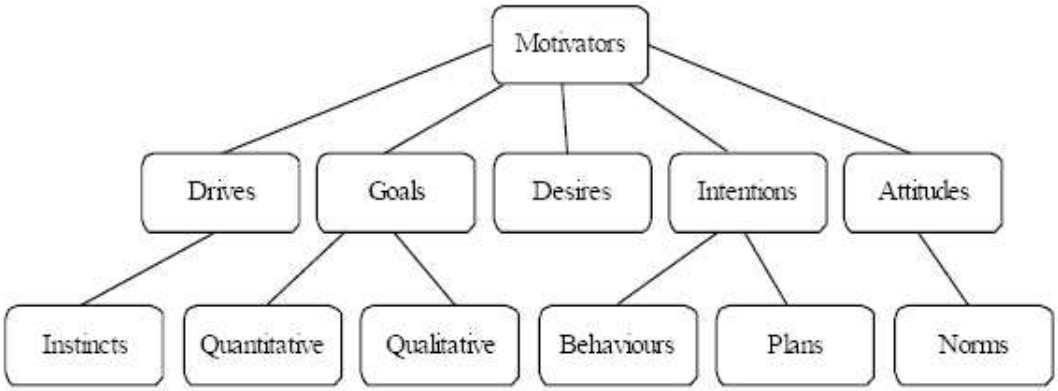


Figure 9.2: Proposed motivational control states (Davis, 2007).

or response. Instincts are highly constrained drives. Instincts and drives are present in robo-CAMAL at the reactive level. Each micro-behaviour can be considered an instinct. For example the reactive control rule

IF(left sonar value $<$ threshold) **THEN**(turn right)

is a pre-set response to a sensor variable. The combination of micro-behaviours to form macro-behaviours can be considered the agent's drives. For example, the drive to *hit* or *avoid* an object.

Goals come in two types, quantitative and qualitative. Quantitative goals are the same as goals used within control theory (Sontag, 1998). Within control theory the system has a specific output that it needs to achieve or maintain. The system uses feedback from its environment to modify its actions to achieve or maintain that state. Quantitative goals are present within Robo-CAMAL at the reactive level. For example, the macro-behaviours can be considered implicit quantitative goals.

Qualitative goals describe some desired end state for an agent. For example when an agent searches through problem space to find a solution or goal state. Qualitative goals are present in Robo-CAMAL at the deliberative level and take the form *hit(blueball)*. However, it is

more appropriate to think of these goals as the end states associated with desires.

Desires are symbolic statements that define a specific preferred environmental state. Desires here are the same as those used within the BDI schema (see section 3.4). Desires within Robo-CAMAL describe the specific goal, the belief required for the goal's success, and the desire's importance. Desires are present at the deliberative level within robo-CAMAL. These take the form *goal(Desire, Success_condition, Goal_importance, Threat_value).*

Intentions are also the same as those used within the BDI schema. They are strategies and plans that are used to achieve desires. Intentions are found within robo-CAMAL at the deliberative level. They take the form of predicates detailing the various possible reactive architectures.

Finally attitudes are pre-dispositions to respond in certain ways to certain perceptual or internal triggers. For example, consider an agent developed to play five aside football. The agent may choose to attack or defend. This can depend on team orders or the specific environmental situation (Bourgne, 2003). These attitudes affect which goals are chosen. If the attitude is to attack then the goal may be to hit the ball. If the attitude is to defend then the goal may be to get between the ball and the scoring zone. Attitudes are present within robo-CAMAL, but are pre-programmed prior to run time. An attitude in robo-CAMAL refers to the pre-defined goal set. Different attitudes or goal sets must be changed by the user off-line.

In summary, motivational control states are distributed throughout the robo-CAMAL architecture. Instincts, desires, and quantitative goals are present at the reactive level. Qualitative goals, desires, intentions, and attitudes are all managed at the deliberative level using a motivational blackboard system.

9.6 Motivational Blackboard

The motivational control states within robo-CAMAL at the deliberative level are managed by the motivational blackboard. The blackboard system can also trigger the appropriate instincts and drives at the reactive level.

The motivational blackboard system has three components.

- The blackboard.
- Knowledge sources.
- A control component (see section 3.6).

The blackboard holds the various motivational control states, as well as other relevant structures such as beliefs etc. The knowledge sources manipulate the structures present on the blackboard. This includes methods to update the various affect values and the processes within the BDI schema.

The control component has three main tasks. The first is to place the feedback messages received from the reactive level on the blackboard. It then controls the order in which the knowledge sources access the blackboard. Its last task in a processing cycle is to send the appropriate control message to the reactive level, and thereby trigger the appropriate drive.

From the deliberative level's point of view, the reactive component can be thought of as a knowledge source. The control component provides the reactive component with information on the blackboard. It then processes the information based on external input, and its internal state. The results of that processing is then placed back on the blackboard.

9.6.1 The Blackboard

The blackboard holds all the information relevant to the agent at any one time. This includes a description of the agent and its environment. It also contains information on its current and previous state. Many of the components of the domain model used at the deliberative are held on the blackboard.

The information held on the blackboard can be divided into several distinct areas:

- Beliefs the agent can have about its environment.
- Desires the agent can have about the objects in its environment.
- Intentions or plans the agent has to achieve its desires.
- Associations that are used to manage the BDI schema.
- A motivator that contains the result of the knowledge sources operations.

The first main component on the blackboard relates to the beliefs the agent can have. The elements from the domain model are (see appendix B)

environment_descriptor_list
clutter_threshold
object_list
object_predicate
domain_synonym
negation

There are also three other elements that allow the agent to construct and maintain its beliefs. These are

reactive feedback messages
belief_preference(X, Y)
agent_time(X)
belief_retain_time(25)

The reactive feedback messages allow new beliefs to be constructed using information from the robot's sensors. The *belief_preference* states which source for a belief is more reliable. For example, a belief drawn from the agent's sensors is considered more reliable than a deduced belief. The *agent_time* is a value to reflect the temporal nature of events. For example a belief with a high *agent_time* value will have been constructed more recently than one with a lower value. The *belief_retain_time* determines the number of deliberative processing cycles a belief is retained for.

The belief construct takes the form

belief(description, source, time)
where *description* ∈ *belief_descriptor*
and *source* ∈ *belief_source*
and *time* ∈ *agent_time*

The *belief_descriptor* can be constructed from the reactive feedback messages and the various domain model statements. It can take the form

environment(description)
where *description* ∈ *environment_descriptor_list*
object_list(object)
where *object* ∈ *object_predicate*
failed(architecture)

The *belief_source* details how the belief was constructed. The values this can take, in order of their reliability, are *sensor*, *deduction*, *assumption*.

The *agent_time* details when the belief was constructed. This value does not refer to the agent's physical time. This value is the number of deliberative processing cycles completed. Beliefs can be pre-programmed or constructed at run time by the agent.

The next main element on the blackboard is the intention list. This relates to the possible plans the agent can have. It is also part of the domain model and takes the form

architecture(sensor, resolver, behaviour)
where *resolver* $\in$ *arbitration_method*
where *behaviour* $\in$ *macro-behaviour*

The sensor represents one of the three possible sensor modalities. It can take the values *sonar*, *vision*, or *both*. The *arbitration_method* details one of the four micro-behaviour combination methods. It can take one of four possible values which are *priority*, *aggregate*, *winner*, or *suppression*. The *macro-behaviour* details the specific general behaviours and can take one of the values, *avoid*, *find*, *track*, or *hit*. This list details all the possible reactive architectures that can be activated. They are all pre-programmed prior to run time.

The goal element on the blackboard takes the form

goal(desire, state, goal_importance, threat_value)
where *state* $\in$ *success_condition*

The *desire* relates to the possible desires the agent can have about objects in its environment. It takes the form

desire(object) where *desire* $\in$ *desire_list* and *object* $\in$ *object_list*

where the desire list is

desire_list([track, hit, avoid, find])

The *success_condition* is the belief descriptor required for the desire to be achieved. The *goal_importance* is an affect value that controls how important the goal is to the agent. The *threat_value* is an affect value that details the number of times the agent has failed to achieve its goal. They are maintained by the predicates

goal_minmax(0.1, 0.9)
goal_threat_max(10)

The *goal_minmax* statement determines the clipping values for the *goal_importance* value. The *goal_threat_max(10)* determines the maximum *goal_threat*. This value represents the number of times a goal can fail before its *goal_importance* is drasticaly reduced.

The desires and their associated success beliefs are a component of the domain model. All goals are pre-programmed prior to run time.

The next element held on the blackboard are the associations (see chapter 8). They take the form:

association(belief, desire, intention, value)

The associations are at the core of the BDI schema.

The association value determines how likely the intention is to achieve the *desire* given the *belief*. Associations can be pre-programmed or constructed at run time by the agent. For a more detailed discussion on associations see chapter 8.

The final element on the blackboard is the motivator. It takes the form:

motivator(goal, association, cycles, value)

The motivator is a result of the knowledge source operations. It contains the agent's chosen *goal*. The appropriate *association* with the chosen reactive architecture to be activated. It also gives the number of cycles the reactive component should run for. The value of the cycle

number is dependent on the agent's goal and is a component of the domain model. This value is defined by the predicates

reactive_cycles(50)
find_reactive_cycles(100)

The motivator *value* gives the importance of the motivator to the agent. This value is relevant only if there is more than one motivator present. However, robo-CAMAL contains only one motivator at any one time. This value is a remnant of previous work done with the model (Lewis, 2004; Davis, 2007). It has been retained for any possible future work done using the robo-CAMAL architecture.

9.6.2 The Knowledge Sources

This section will give an overview of the various knowledge sources (KS) that access the blackboard. There are several general KS which access the blackboard in the following order:

time_update
belief_construction
belief_update
desire_update
choose_motivator
intention_update
reactive component

The first and most simple KS is the *time_update*. This increments the *agent_time* value by 1 for every complete deliberative cycle.

The second KS is *belief_construction*. This KS uses the information provided by the reactive component to construct a belief about its environment. It also accesses the motivator to determine if any new belief satisfies the agent's goal. If not then the belief

failed(intention, perception, agent_time)

is constructed.

The next KS is the *belief_update*. Firstly all old beliefs about failed intentions and hit objects are removed. This is because they are no longer relevant to the agent's operation. Next the differences are resolved between the new and existing beliefs.

The first resolution is that if two beliefs have the same proposition, then the older of the two is removed. The second checks the source of two beliefs of the same proposition. The belief from the preferred source is retained.

Next the negation model is used to resolve any conflicting beliefs. If two beliefs are found to conflict, then the older of the two is removed. Next the *domain_synonym* is used to deduce any beliefs that arise from an existing belief. At this point any belief that is older than X cycles, where X is determined by *belief_retain_time*, is removed. This is based on the assumption that beliefs older than X deliberative cycles are unreliable. This step also prevents the belief set becoming too large and unmanageable.

Finally beliefs about the agent's environment are evaluated. Firstly the number of objects in the environment is checked. This determines if the environment is cluttered. It then checks the type of objects present to determine if the environment is static or dynamic. The assumption is that if a robot is present then the environment is dynamic. These environment updates are determined by the domain model.

The next KS is the *desire_update*. Firstly the motivator is evaluated to determine if the agent's goal has been achieved. If the agent has failed its goal, the relevant *association_value* is reduced. The relevant *goal_threat* value is increased by one. If the goal *threat_value* reaches X, where X is determined by *goal_threat_max*, then the *goal_importance* value is reduced to the lower clipped value determined by the predi-

cate *goal_minmax(0.1, 0.9)*. This ensures that it is a repeated failure, and not a single failure, that affects a goals's *goal_importance* value. For example if the agent fails to hit the ball once, it may simply have missed its target. If however the agent fails ten times in a row it is likely that the agent is unable to achieve the goal for some reason.

If the agent achieves its goal then the relevant association value is increased. The relevant goal threat value is reduced to 0, and the importance value is set to 0.5. This is done to ensure that goals with a high importance value are not repeated over and over again. All other *goal_importance* values are increased by the default *goal_importance* increment (0.02 here), up to a maximum *goal_importance* (0.9 in the current work). This ensures goals with lower importance values will become more important over time.

The next KS is *choose_motivator*. This determines the motivator the agent wishes to instantiate. Firstly a list containing all the agent's goals is created. Any goal that has just been achieved is removed. Next all the relevant associations are chosen. This is based on the goal list and the agent's belief set. If no relevant associations exist then new ones are created.

The new associations are based on the agent's current belief and goal set. Each belief and goal pairing is combined with all the possible reactive architectures. This creates 48 new associations per belief/goal combination. Each new *association_value* is set to the default value of 0.5.

At this point each goal has a number of associations that are relevant. Each goal and association pair are ranked according to equation 8.11. For a more detailed description of the association ranking process see section 8.6.

The goal association pair with the highest rank is used to construct the motivator. The number of cycles contained within the motivator

is dependent on the goal. If the goal is to find an object then the cycle number is set to 100. This is to give the agent time to look for the object. All other goals have a cycle number of 50. These values are set by the predicates

reactive_cycles(50)
find_reactive_cycles(100)

The next KS is the *intention_update*. This extracts the relevant information from the motivator to configure the reactive architecture.

The final KS is the reactive component. This retrieves the chosen intention, and posts the results of its perceptual module upon the blackboard.

9.6.3 The Control Component

The control component has three tasks. The first is to place the feedback from the reactive component on the blackboard. It then calls the KS in the appropriate order. This order is as follows

time_update
belief_construction
belief_update
desire_update
choose_motivator
intention_update
reactive component

The control component's final job is to pass control, with the relevant configuration messages, back to the reactive component.

9.7 Summary

This chapter looked at motivation within robo-CAMAL. It first gave a brief overview on some of the different perspectives of motivation. It then gave some examples of how motivation can be used by a robotic agent. Next it looked at motivation as a control state, and its relevance to robo-CAMAL. It finally described the motivational blackboard. The blackboard is used to control the motivational control states at the deliberative level. A more detailed look at the motivational blackboard can be found in appendix A.

Chapter 10

Experimental Results

10.1 Introduction

This chapter will present a series of experiments with robo-CAMAL, and with their results. It will start by presenting the control experiments. These were designed to ensure the architecture performed as expected, as well as providing performance data with which to compare the subsequent experiments.

Section 10.3 details the learning experiments. In the first learning experiment robo-CAMAL was provided with the correct environmental beliefs, and all the possible actions it could take. Robo-CAMAL then attempted to learn the correct action with which to achieve its goal.

In the second learning experiment robo-CAMAL was given the correct actions with which to achieve its goal. Robo-CAMAL then attempted to learn the optimal action to take in order to achieve that goal.

The final experiment investigates robo-CAMAL's ability to adapt in a changing environment. Here robo-CAMAL was provided with the correct actions in order to achieve a set of goals. Robo-CAMAL was then required to observe its environment and alter its goals according to changes in that environment.

10.2 Control Experiments

Several control experiments were performed to test robo-CAMAL's reactive and deliberative components. The first two experiments tested the vision system, and the various micro-behaviours. They also provided data with which to compare the various learning experiments. The final two control experiments tested the deliberative component. This involved testing the association generation and the goal selection mechanisms.

10.2.1 The *hit(Object)* Control Experiment

The *hit(Object)* experiment was designed to test all the object based micro-behaviours. It also tested how well the vision system was integrated with the reactive architecture.

In this experiment the deliberative component was disconnected. The reactive component was pre-configured to hit a specific object. The reactive component was then allowed to operate in a number of different environments. Each experiment was run for five minutes. The number of times the robot collided with each object was recorded. Figures 10.1 to 10.3 show the number of times each of the three possible objects were hit by reactive robo-CAMAL in each of the possible environments..

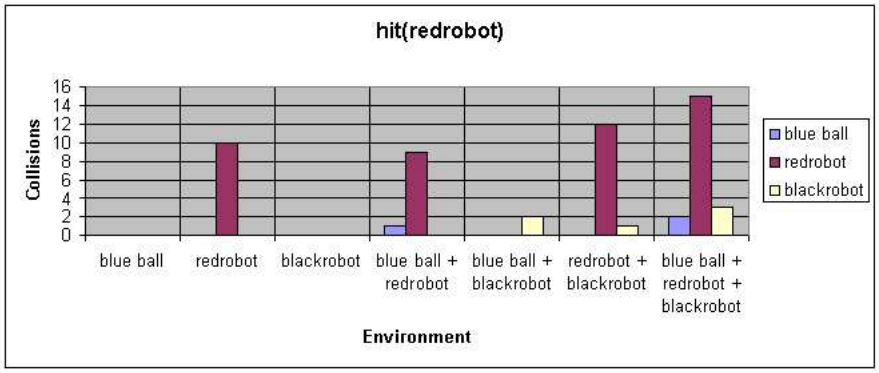


Figure 10.1: The *hit(redrobot)* control experiment.

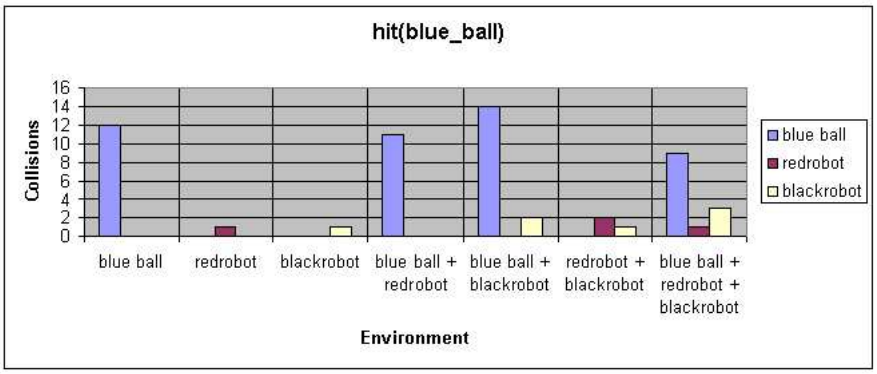


Figure 10.2: The *hit(blueball)* control experiment.

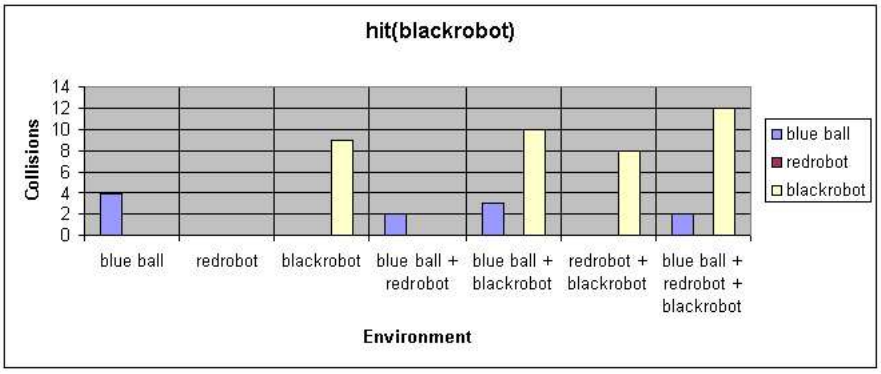


Figure 10.3: The *hit(blackrobot)* control experiment.

Figure 10.1 shows the results from the *hit(redrobot)* configuration. It can clearly be seen that there are a significant number of collisions with the *redrobot*. This clearly demonstrates that the various object based micro-behaviours are working as expected. It also shows that the vision system is correctly identifying the *redrobot*.

Figure 10.2 shows the results from the *hit(blueball)* configuration. It can clearly be seen that there are a significant number of collisions with the *blueball*. This shows that the vision system is correctly identifying the *blueball*.

Figure 10.3 shows the results from the *hit(blackrobot)* configuration. Although there are a significant number of collisions with the *blackrobot*, there are also a number of collisions with the *blueball*. This seems to indicate that although the vision system is correctly identifying the *blackrobot*, it may also be misclassifying the *blueball* as the *blackrobot*. This is due to the large number of black pixels generated by the vision system in response to the *blueball* (see section 5.6).

10.2.2 The *avoid(Object)* Control Experiment

The *avoid(Object)* control experiment was designed to test the avoid object micro-behaviours. It was also designed to provide control data with which to compare the results from the second learning experiment.

For this experiment the deliberative component was disconnected. The reactive component was configured to avoid objects. Reactive robo-CAMAL was run three times in each possible environment. Each run lasted five minutes. The number of objects within the environment was varied between zero and three. The dynamic objects were the *redrobot* and the *blackrobot*. The number of collisions was recorded for each run. The results can be seen in figures 10.4 to 10.9.

From the graphs it is clear that the suppression combination method

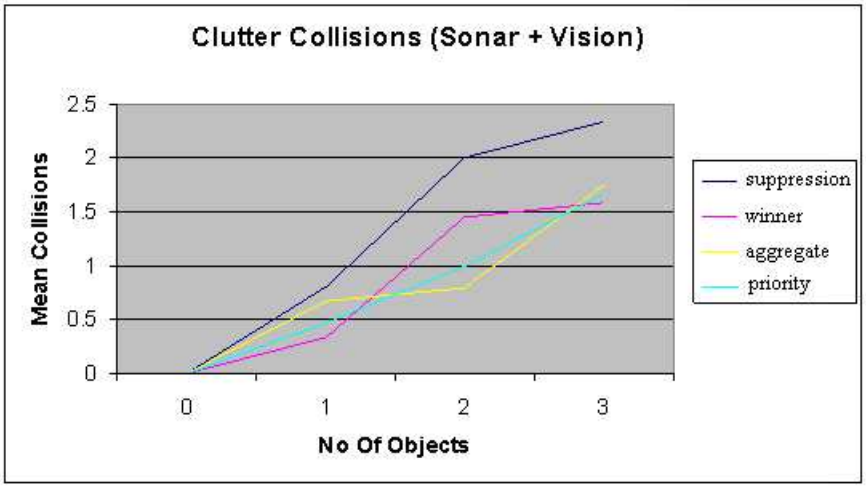


Figure 10.4: Mean collisions vs number of objects.

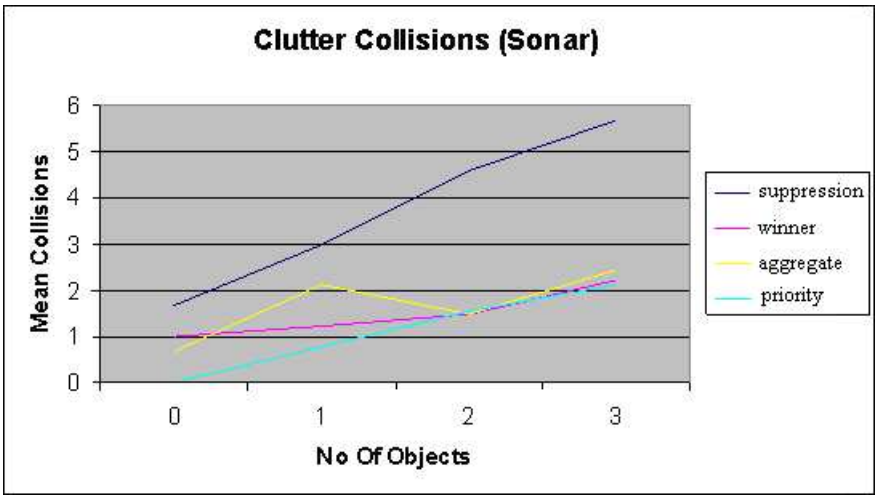


Figure 10.5: Mean collisions vs number of objects.

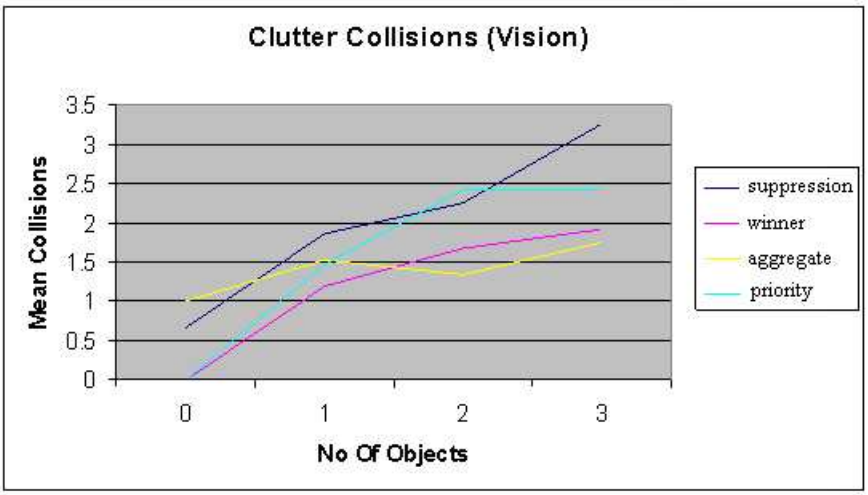


Figure 10.6: Mean collisions vs number of objects.

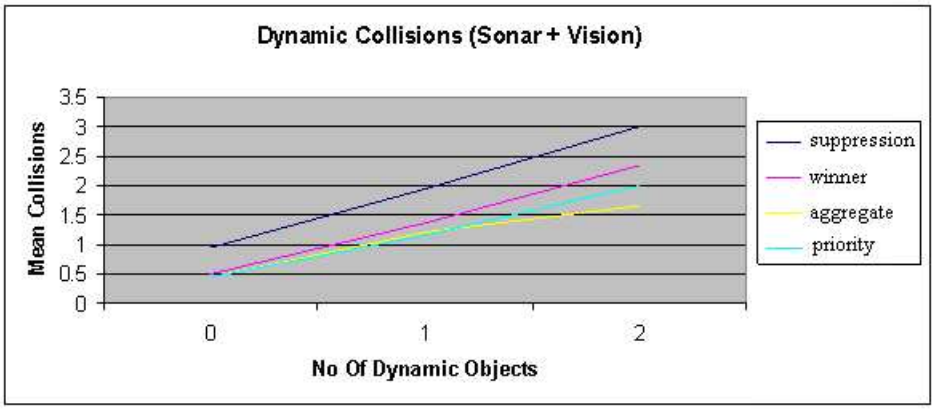


Figure 10.7: Mean collisions vs number of dynamic objects.

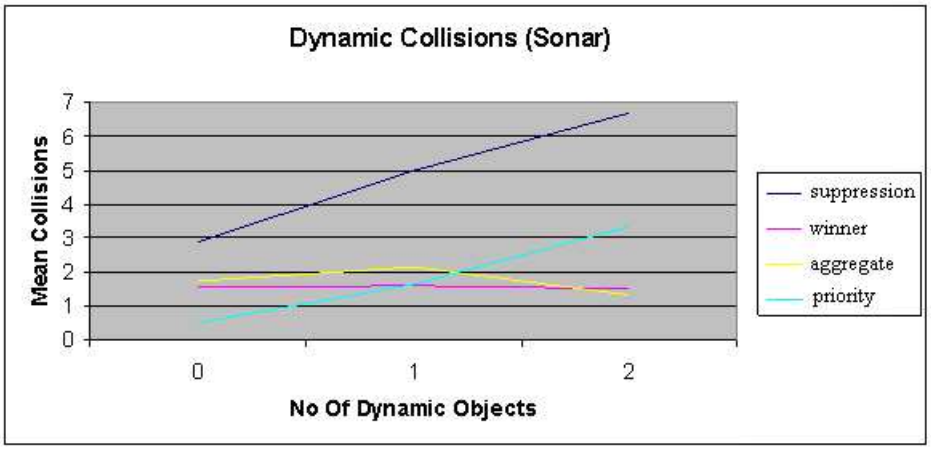


Figure 10.8: Mean collisions vs number of dynamic objects.

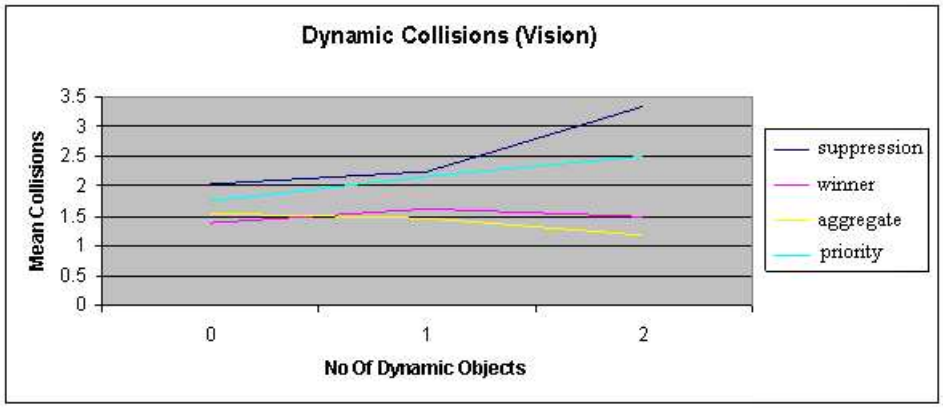


Figure 10.9: Mean collisions vs number of dynamic objects.

shows the poorest performance. This result is due to the nature of the algorithm. As the suppression method has to deactivate, then reactivate the various micro-behaviours, it takes longer to react. This results in a slower reaction time, and there by reducing reactive robo-CAMAL's ability to act quickly within its environment.

A second point to note is that the sonar sensor shows the poorest performance. This is due to the poor resolution of the sonar sensors. As the vision system can delineate the objects in the environment to a greater extent, it is better at avoiding/interacting with them.

Finally, the results seem to suggest that the aggregate combination method shows the best performance within a cluttered environment. This is followed closely by the winner combination method. However, as the results are so close, it is unclear whether this result is a true indication of the differences between the two methods. Due to the statistical nature of the data, more experiments are required to determine if there is a major difference between the aggregate and winner combination methods. For this reason, no definitive distinction can be made between the two best performing combination methods.

10.2.3 Association Generation Control Experiment

This experiment was designed to test the mechanism with which robo-CAMAL generates associations. For this experiment virtually all of the reactive component was disabled. The only element that remained was the vision system. The agent was given the goal *find(redrobot)* and the belief *environment(sparse)*. The agent's associations were not pre-programmed. The robot was stationary as only the vision system was enabled at the reactive level.

Once the experiment started the *redrobot* was placed in front of the agent. After 30 seconds the *redrobot* was replaced with the *blackrobot*. After a further 30 seconds the *blackrobot* was replaced by the *redrobot*.

The results can be seen in figure 10.10.

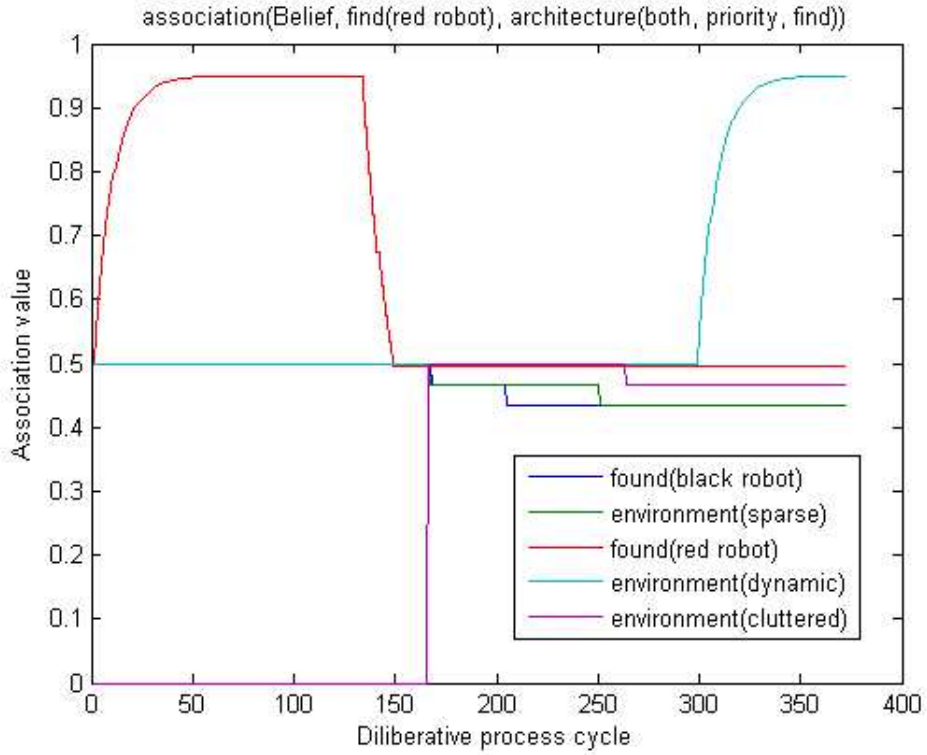


Figure 10.10: Association generation process.

Figure 10.10 shows several associations with the same goal and intention. Each line represents an alternative belief. Initially associations with the beliefs *environment(sparse)*, *environment(dynamic)*, and *found(redrobot)* are created. The belief *environment(dynamic)* comes from the domain model assumption that if a robot is present, then the environment is dynamic. Initially the association with the belief *found(redrobot)* increases as it achieves its goal. Once the *redrobot* is removed then the association fails, and therefore has its value reduced.

Once the *blackrobot* is introduced, new associations involving the beliefs *found(blackrobot)* and *environment(cluttered)* are created. The belief *environment(cluttered)* is generated due to the domain model assumption that more than two objects represents a cluttered environment. This appears in figure 10.10 as the association values that

jump from 0 to 0.5. Once the *redrobot* was reintroduced the association with the belief *environment(dynamic)* increases.

The results in figure 10.10 demonstrate that the deliberative component is working as expected. The agent is constructing the correct beliefs based on feedback from the vision system. This shows that the belief mechanism is operating as expected. The generation of new associations based on new beliefs shows that the association generation mechanism is working as expected. The association values increase in the presence of the *redrobot*, and decrease in its absence. This shows that the goal achievement and association update mechanisms are operating as expected.

10.2.4 Goal Selection Control Experiment

This experiment was designed to test the goal selection mechanism. For this experiment, the only active component of the reactive level was the vision system. The deliberative component was pre-programmed with the three goals

- *hit(redrobot)*
- *hit(blackrobot)*
- *hit(blueball)*

Beliefs that represent the state of the environment were pre-programmed into the agent. In addition associations were pre-programmed that link the agents current environment, and goals, to the actions specifically designed to achieve those goals in that environment. Once the experiment started the *redrobot* was placed in front of robo-CAMAL. This was then replaced by the *blackrobot*, which was in turn replaced by the *blueball*. The experiment was run for five minutes. The results can be seen in figure 10.11.

Initially, deliberative robo-CAMAL attempts to hit the *blueball* and the *blackrobot*. However, the agent fails as these objects are not present. This failure reduces the importance value of the respective goals. The agent then attempts to hit the *redrobot*, at which point it achieves its goal. Robo-CAMAL continues to attempt, and achieve the goal *hit(redrobot)*. At this point the importance value of the other goals starts to increase.

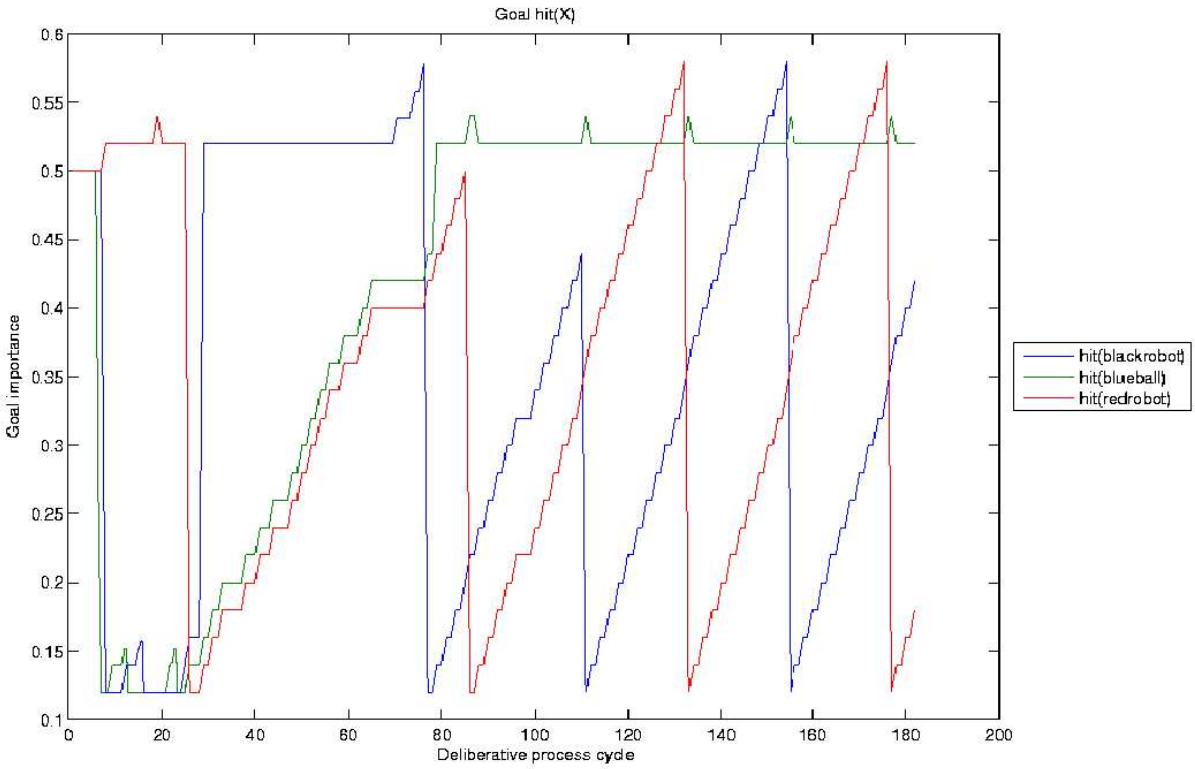


Figure 10.11: *hit(object)* control experiment.

After some time the redrobot is removed. At this point the *hit(redrobot)* goal fails and its importance is reduced. The *blackrobot* was then placed in front of robo-CAMAL and the *hit(blackrobot)* goal was achieved. During this period, robo-CAMAL continues to achieve the *hit(blackrobot)* goal. This can be seen as the goal importance value remains unaltered. The goal importance value of the other goals increase.

The *blackrobot* was then replaced by the *blueball*. This causes the *hit(blackrobot)* goal to fail, and the *hit(blueball)* goal to succeed. As the importance value of the other goals increases, robo-CAMAL attempts to achieve them. As the object of the goal is not present, the goal fails and the importance value is reduced. This process continues until the environment changes.

This result demonstrates that the goal selection mechanism works as expected. In general, the agent attempts the goal that it considers to be the most achievable. However, periodically robo-CAMAL attempts other goals to determine if they have become achievable. The goal is attempted when its importance value significantly outbalances the association value.

10.3 Learning Experiments

Two learning experiments were designed to test the learning ability of robo-CAMAL. The first experiment was developed to test whether robo-CAMAL can learn the correct action to achieve its goal. The second learning experiment was designed to test whether robo-CAMAL can learn the optimal action to achieve its goal i.e. given a list of actions that achieve the same thing, can robo-CAMAL determine the best one.

10.3.1 First Learning Experiment

The first learning experiment was designed to test robo-CAMAL's ability to learn the correct action to achieve a goal. To do this robo-CAMAL needs to be able to generate a list of associations, and select the correct association to achieve that goal.

For this experiment robo-CAMAL was instantiated with a single object based goal. It was also pre-programmed with the correct beliefs.

No associations were pre-programmed. Robo-CAMAL was run in one of a number of possible environments for five minutes. The experiment was repeated three times for each environment.

Robo-CAMAL was pre-programmed with *goal(X)*. The six environments used for the experiment can be seen in table 10.1. Here X is the object of the goal, with Y and Z representing the other possible objects. The experiment was run for every object based goal (i.e. *find*,

Environment	**Objects**
1	X
2	Y
3	Z
4	X,Y
5	X,Z
6	X,Y,Z

Table 10.1: The possible environment combinations.

track, and *hit)*, with every possible object as the focus of that goal.

Given three possible goals, three possible objects, with three experiments in six environments, the total number of experiments was 162. Each experiment produced a number of associations. The value of each association was recorded at every deliberative processing cycle. The results presented here are the most relevant ones used to highlight the main points. All of the results can be found in appendix C.

Figure 10.12 shows some of the association values recorded for the *hit(redrobot)* experiment. The environment contained a *redrobot*. The experiment can be tracked as follows. Initially robo-CAMAL has no associations. However it does have the goal *hit(redrobot)*, a set of pre-programmed beliefs, and a list of the possible actions it can perform. From this robo-CAMAL can produce a list of associations.

Figure 10.12 only shows the key associations that varied. This is because robo-CAMAL produces anywhere in the region of 50 associations per run. Some of these associations have low values, are never

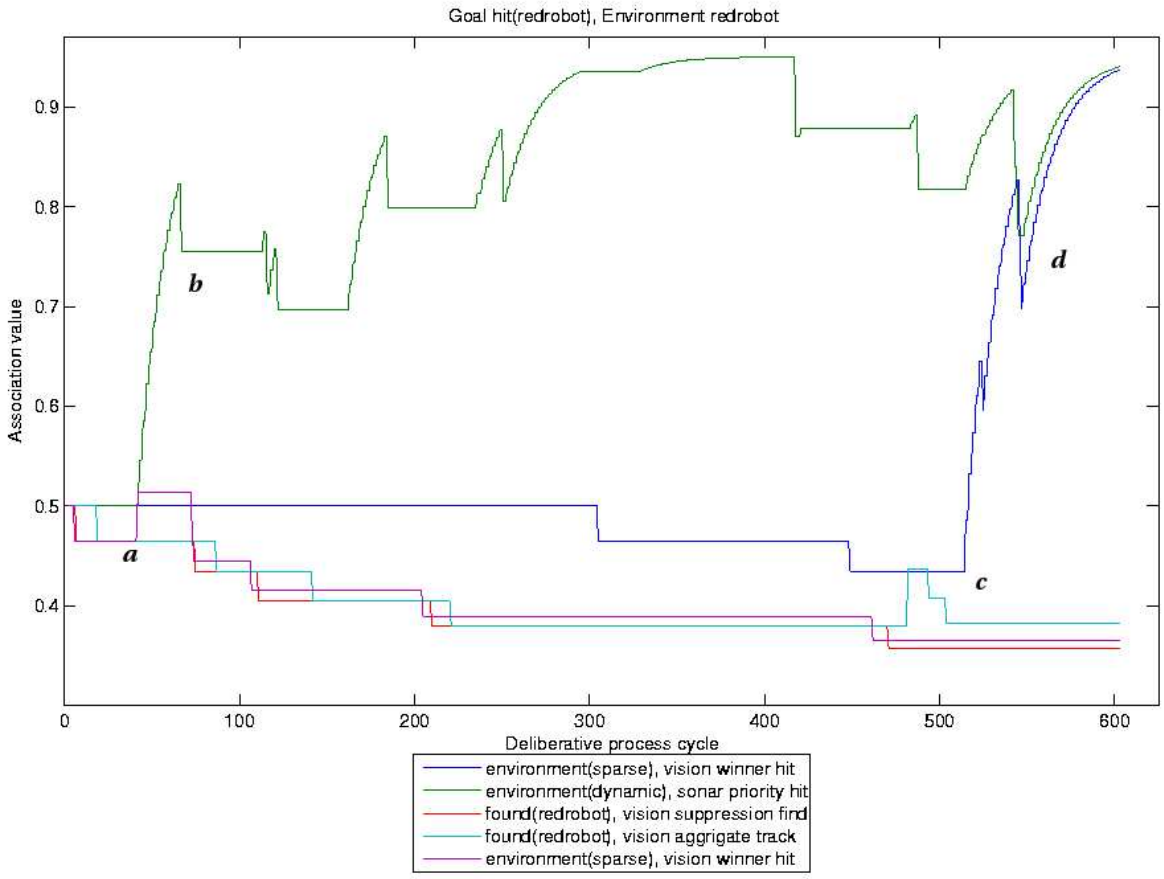


Figure 10.12: Associations in the *hit(redrobot)* learning experiment.

chosen, and so their value never changes.

Initially robo-CAMAL tries various associations in order to hit the *redrobot*, all of which fail. However, at point **a**, the *association(environment(dynamic), hit(redrobot), architecture(sonar, priority, hit))* succeeds. This can be seen in figure 10.12 as the green line with the increasing association value. After some time robo-CAMAL fails to hit the *redrobot*. This can be seen by the fall in association value of the green line at point **b**.

At this point robo-CAMAL stops using this association. This can be seen by the unchanging association value of the green line. As the agent only has one goal, the only factors influencing the choice in association is the associations value, and the agent's beliefs. At this point the association value is high, therefore the only reason the association is not chosen must be because the relevant belief is no longer present. This makes sense due to robo-CAMAL failing to hit the *redrobot*. This means the belief *lost(redrobot)* is constructed. If the *redrobot* has not been found within 25 deliberative processing cycles, then all beliefs regarding the *redrobot* are removed. With no object beliefs present, robo-CAMAL will deduce that the environment is static and sparse.

This pattern of behaviour continues until point **c**. At this point the *association(environment(sparse), hit(redrobot), architecture(vision, winner, hit))* succeeds. This is represented in figure 10.12 as the increase in association value of the blue line.

It can be seen in figure 10.12 that the blue and green lines closely mirror each other, and have almost have identical shapes. This can clearly be seen at point **d** in figure 10.12. This occurs because robo-CAMAL is switching between the two associations on each deliberative processing cycle, due to the way the belief update mechanism works.

Initially the association with the belief *environment(sparse)* succeeds. The belief is considered true so is left unmodified. However,

as the belief *found(redrobot)* is held, the belief *environment(dynamic)* is deduced. This belief is updated with the current time value. This means that the *environment(dynamic)* belief is more recent than the *environment(sparse)* belief. One of the factors affecting the choice of associations is the age of the belief. In this case, as the *environment(dynamic)* belief is more recent, the association containing this belief is therefore chosen. This cycle repeats for the *environment(sparse)* belief, thereby causing the chosen association to swap every deliberative processing cycle.

Not all the experiments showed the same level of success as seen in figure 10.12. Figure 10.13 shows the results for the *find(blueball)* goal, when the environment contained only a *blackrobot*. This failed in two ways. The first is that several associations appeared to find the *blueball* when it was not present. The second is that the associations themselves are wrong. The environment only contained one object and should therefore be considered sparse. However, figure 10.13 shows that the belief *environment(cluttered)* is present in the associations.

This result highlights the difficulty when using real sensors. The only way robo-CAMAL can construct the belief *found(blueball)* is if the vision system identifies a *blueball*. This means that for 10.13 the vision system incorrectly identified an object as a *blueball*. This also explains the presence of the belief *environment(cluttered)*. If robo-CAMAL believes it has found the *blueball* and the *blackrobot*, then it will deduce that the environment is cluttered (as more than one object is present). It is clear that the cause of this failure is the vision system incorrectly identifying a *blackrobot* as a *blueball*.

How can the effectiveness of robo-CAMAL's learning ability be assessed? For this two criteria were set for each association to meet. The first is the association value. If this value goes above a specific threshold, then the association is recorded. The thresholds chosen

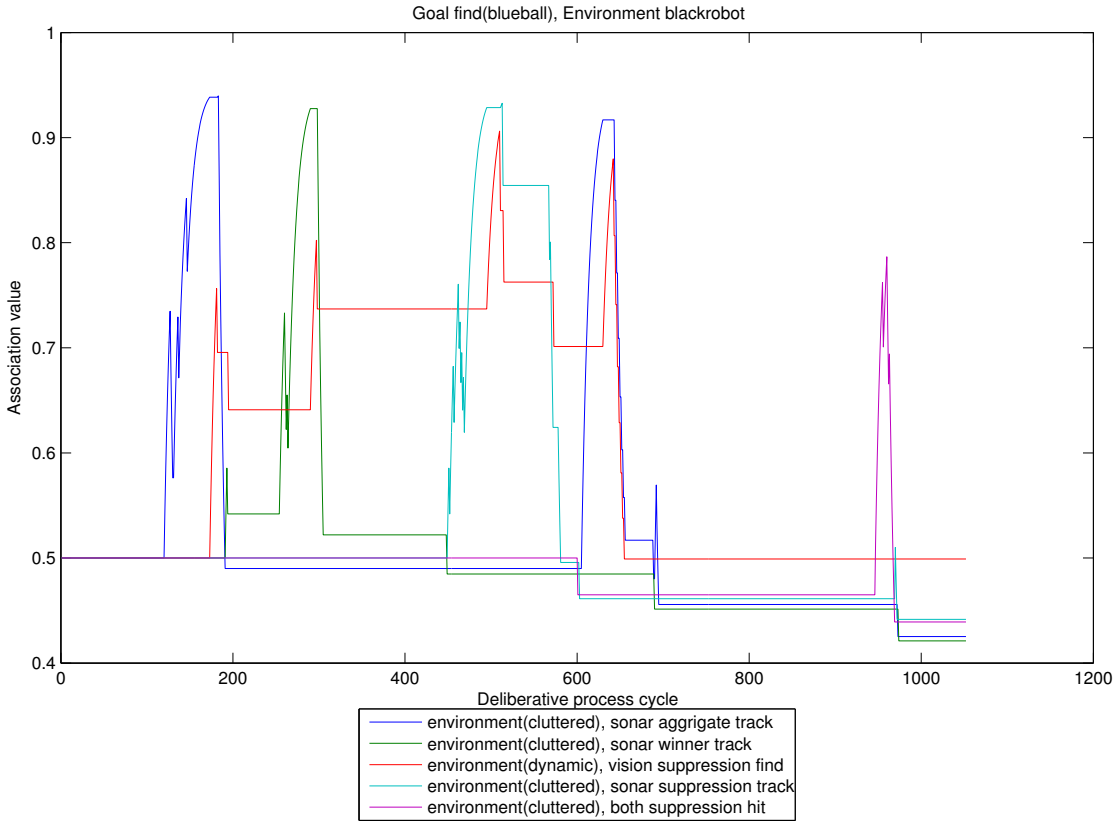


Figure 10.13: Associations in the *find(blueball)* learning experiment.

were 0.65 and 0.75. These values were chosen as the association value is considered to be the likelihood of success of that association. For example, an association with a value of 1 should have a 100% success rate, where as a value of 0 would always fail. The association values of 0.65 and 0.75 therefore represent likelihood values of 65% and 75%. These are considered reasonable likelihood values to consider an association as being an accurate reflection of a specific belief-goal-action mapping.

The second criteria relates to the amount of time the association spends above the value threshold. Five time thresholds were chosen. These were one deliberative processing cycle, 30 seconds, 1 minute, 1 minute 30 seconds and 2 minutes 30 seconds. The number of processing cycles spent above the association threshold divided by the total number of deliberative processing cycles gives a percentage of the time the association spent above its threshold. This can easily be converted into a real time value as each experiment ran for five minutes. If an association persisted longer than the persistence threshold it was recorded.

Time above threshold	**Associations above value threshold**					
	0.65			*0.75*		
	correct	*incorrect*	*ratio*	*correct*	*incorrect*	*ratio*
One processing cycle	210	159	1.32	181	106	1.71
30 sec	135	36	3.75	114	22	5.18
1 min	106	22	4.81	98	18	5.44
1 min 30 sec	91	16	5.69	81	8	10.13
2 min 30 sec	68	3	22.6	52	0	N/A

Table 10.2: Total number of correct and incorrect associations found.

Table 10.2 shows the number of correct and incorrect associations found during the first learning experiment. It also shows the ratio of

correct to incorrect associations as shown in equation 10.1.

$$\frac{correct\ associations}{incorrect\ associations} = ratio \tag{10.1}$$

Any ratio value above 1 indicates that robo-CAMAL has successfully found more correct associations than incorrect ones. The higher this ratio, the more accurate the learning mechanism.

The results show that overall robo-CAMAL correctly identifies the appropriate association to achieve its goal more often than not. However, if the shortest time constraint is chosen, robo-CAMAL's performance is only slightly better than a random process. As the ratio for this constraint is only just above 1, robo-CAMAL is identifying the correct association only slightly more than 50% of the time.

If an association is required to persist for more than 30 seconds, then the accuracy of the learning mechanism increases almost three fold. It is clear from the results that the longer the association is required to persist, the greater robo-CAMAL's learning accuracy.

This increased accuracy comes at a price. It is also clear that the longer the association is required to persist, the fewer the number of associations to be found. This can be seen in table 10.3, which shows the average number of associations found in a ten minute period. The total time of all the experiments was 810 minutes. This means the total number of accurate associations found divided by 81 gives the average number of associations found over a 10 minute period.

It is clear that increasing the time an association is required to persist for increases the time it takes for robo-CAMAL to learn the correct associations. For example, even though an association persistence of 2 minutes and 30 seconds provides a very accurate learning threshold, it takes robo-CAMAL over 10 minutes to learn a new association. This behaviour is expected. The longer an association is required to persist, the less time is available to try alternative associations.

Time above threshold	**Correct associations found per 10 min**	
	Value threshold 0.65	*Value threshold 0.75*
One cycle	2.59	2.23
30 sec	1.67	1.40
1 min	1.31	1.21
1 min 30 sec	1.12	1.00
2 min 30 sec	0.84	0.64

Table 10.3: Total number of correct associations found in 10 minutes.

From table 10.2 and 10.3 it can be seen that increasing the association threshold value has the same effect on robo-CAMAL's learning accuracy and speed, as increasing the associations persistence threshold.

These results can also inform us about goal based learning. Table 10.4 to 10.6 show the learning results for the specific goals used in the learning experiments. Each table shows the correct/incorrect association ratio, and the average number of associations found in 10 minutes.

Association persistence	**Association found rate and ratio**			
	0.65		*0.75*	
	ratio	*rate*	*ratio*	*rate*
One cycle	0.96	2.59	1.27	2.41
30 sec	3.11	2.07	4.55	1.85
1 min	3.90	1.59	4.22	1.42
1 min 30 sec	4.88	1.44	11.00	1.22

Table 10.4: Find learning experiment results.

From tables 10.4 to 10.6 it is clear that robo-CAMAL is able to learn the correct association to hit objects far more accurately than the other goals. For each value and persistence threshold, the ratio is significantly higher than for the other goal based learning experiments. This is due to the nature of the hit macro-behaviour.

The hit macro-behaviour actively tries to achieve its goal. In

Association persistence	**Association found rate and ratio**			
	0.65		*0.75*	
	ratio	*rate*	*ratio*	*rate*
One cycle	1.91	3.03	2.48	2.48
30 sec	5.18	2.11	6.29	1.63
1 min	6.86	1.78	9.75	1.44
1 min 30 sec	9.75	1.44	36.00	1.33

Table 10.5: Hit learning experiment results.

Association persistence	**Association found rate and ratio**			
	0.65		*0.75*	
	ratio	*rate*	*ratio*	*rate*
One cycle	1.35	2.15	1.53	1.82
30 sec	3.57	0.93	5.00	0.74
1 min	4.25	0.63	3.75	0.56
1 min 30 sec	3.25	0.48	3.00	0.44

Table 10.6: Track learning experiment results.

the case of the find macro-behaviour, robo-CAMAL simply wanders around its environment. It passively observes the objects within its environment and does not attempt to go and look for them. Therefore if robo-CAMAL finds the *blueball*, it will not continue to look at it. This means it will lose the *blueball* after it has moved on and 25 deliberative processing cycles have passed. This will then cause the goal *find(blueball)* to fail.

Although the track macro-behaviour also actively attempts to follow an object, the line between being close to, and hitting the object is a fine one. For example, robo-CAMAL might be close to the *redrobot*. The *redrobot* might unexpectedly turn and hit robo-CAMAL. This would then cause the association to fail the *track(redrobot)* goal. This effect causes robo-CAMAL to fail more often when trying to track an object than any other goal. This can be seen in the amount of time robo-CAMAL takes to learn the track behaviour. Robo-CAMAL takes twice as long to learn the correct association to track an object. In

some cases it takes over 20 minutes to learn the correct association.

Figure 10.14 shows this. Here the goal is to *track(blackrobot)* in an environment containing only a *blackrobot*. The correct association is found and succeeds for some time. Then the association value plummets as robo-CAMAL hits the *blackrobot* and fails its goal. This can be seen at points **a** and **b** in figure 10.14. This behaviour also tells us that the current BDI and domain model interaction is very sensitive to the pre-programmed definitions. It also implies that under certain circumstances it is somewhat brittle.

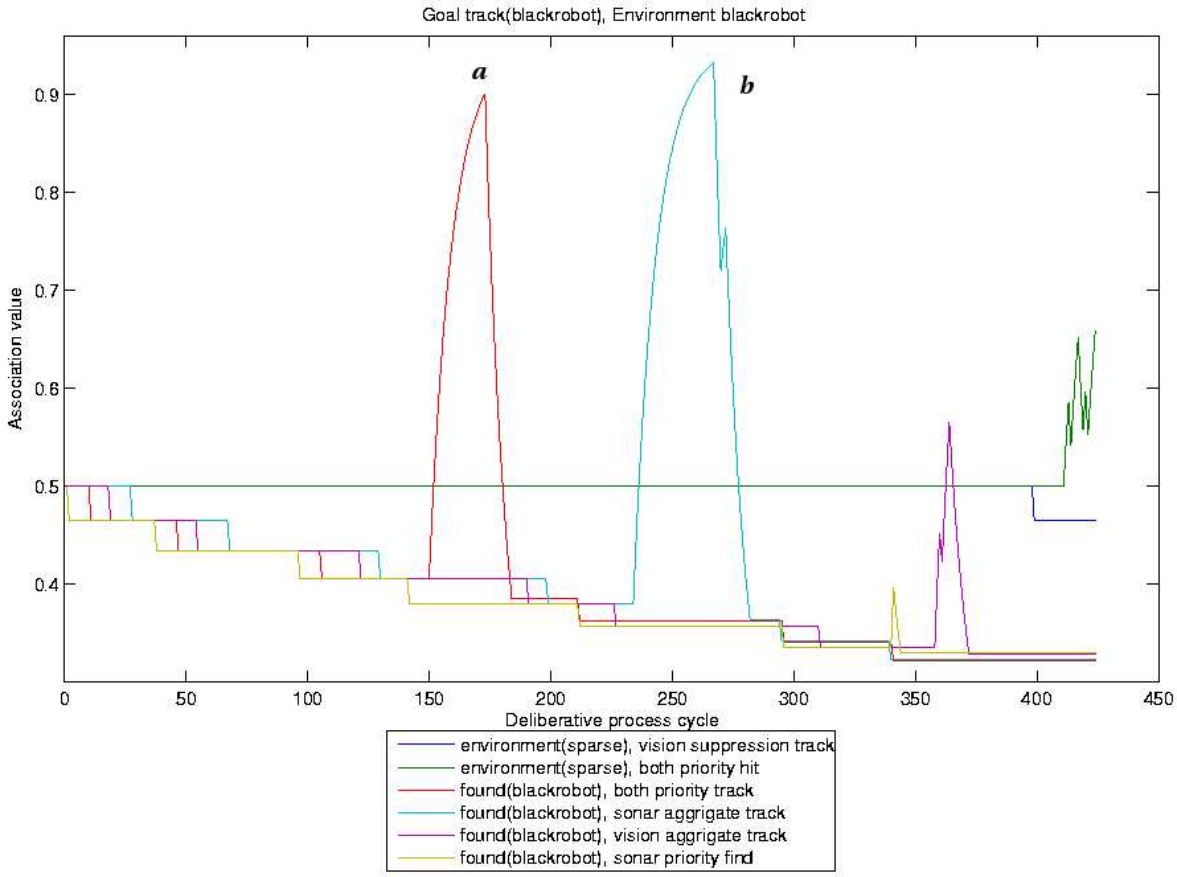


Figure 10.14: Associations in the *track(redrobot)* learning experiment.

The same analysis can be performed for the specific object based goals. Table 10.7 to 10.9 show the learning results for the specific object based goals used in the learning experiments. Each table shows

the correct/incorrect association ratio, and the average number of associations found in 10 minutes.

Association persistence	**Association found rate and ratio**			
	0.65		*0.75*	
	ratio	*rate*	*ratio*	*rate*
One cycle	2.03	2.26	2.50	2.03
30 sec	5.44	1.81	6.43	1.67
1 min	7.17	1.59	10.50	1.56
1 min 30 sec	10.25	1.52	38.00	1.41

Table 10.7: *redrobot* learning experiment results.

Association persistence	**Association found rate and ratio**			
	0.65		*0.75*	
	ratio	*rate*	*ratio*	*rate*
One cycle	1.50	2.78	1.91	2.48
30 sec	4.17	1.85	7.00	1.56
1 min	6.67	1.48	6.40	1.19
1 min 30 sec	5.33	1.19	10.33	1.15

Table 10.8: *blackrobot* learning experiment results.

Association persistence	**Association found rate and ratio**			
	0.65		*0.75*	
	ratio	*rate*	*ratio*	*rate*
One cycle	0.90	2.63	1.20	2.19
30 sec	2.60	1.44	3.00	1.00
1 min	2.50	0.93	2.25	0.67
1 min 30 sec	3.17	0.70	3.00	0.44

Table 10.9: *blueball* learning experiment results.

Tables 10.7 and 10.8 show that robo-CAMAL learns the correct associations for *redrobot* based goals with slightly more accuracy, and marginally faster, than for *blackrobot* based goals. This is because robo-CAMAL finds it easier to identify the *redrobot* than the *blackrobot*, because the two corresponding colour profiles differ greatly.

From table 10.9, it can clearly be seen that robo-CAMAL has difficulty learning the correct associations for *blueball* based goals. This is due to the difficulty robo-CAMAL has in identifying the *blueball*. Firstly the *blueball* is smaller than the other objects in the environment. This means it produces a smaller response in the vision system. The *blueball* also produces a number of black pixel responses in the vision system. This means that robo-CAMAL may correctly identify the *blueball*, then misclassify it as it attempts to achieve its goal.

For example, robo-CAMAL may have the goal *hit(blueball)*. Once the *blueball* has been identified, robo-CAMAL will attempt to run into it. As robo-CAMAL manoeuvres into position to strike the *blueball*, the angle, and therefore its response in the vision system, changes. From a different angle the *blueball* may look pre-dominantly black and robo-CAMAL will incorrectly identify it as the *blackrobot*. The association chosen to hit the *blueball* will then fail. This result clearly highlights the difficulties in using real sensors in real environments, or the inadequacies of the simple vision system used within the robo-CAMAL architecture.

This can be seen in figure 10.15. Here the goal is to *find(blueball)* in an environment containing a *blueball*. Robo-CAMAL uses the track association to try and find the *blueball*. It successfully identifies the *blueball* and turns towards it. This changes the angle from which robo-CAMAL views the *blueball*. As it can no longer identify the object, the association fails. A second example of this can be found in figure 10.13.

10.3.2 Second Learning Experiment

The Second learning experiment was designed to investigate robo-CAMAL's fine learning ability. That is robo-CAMAL's ability to differentiate between actions that achieve the same goal. For this

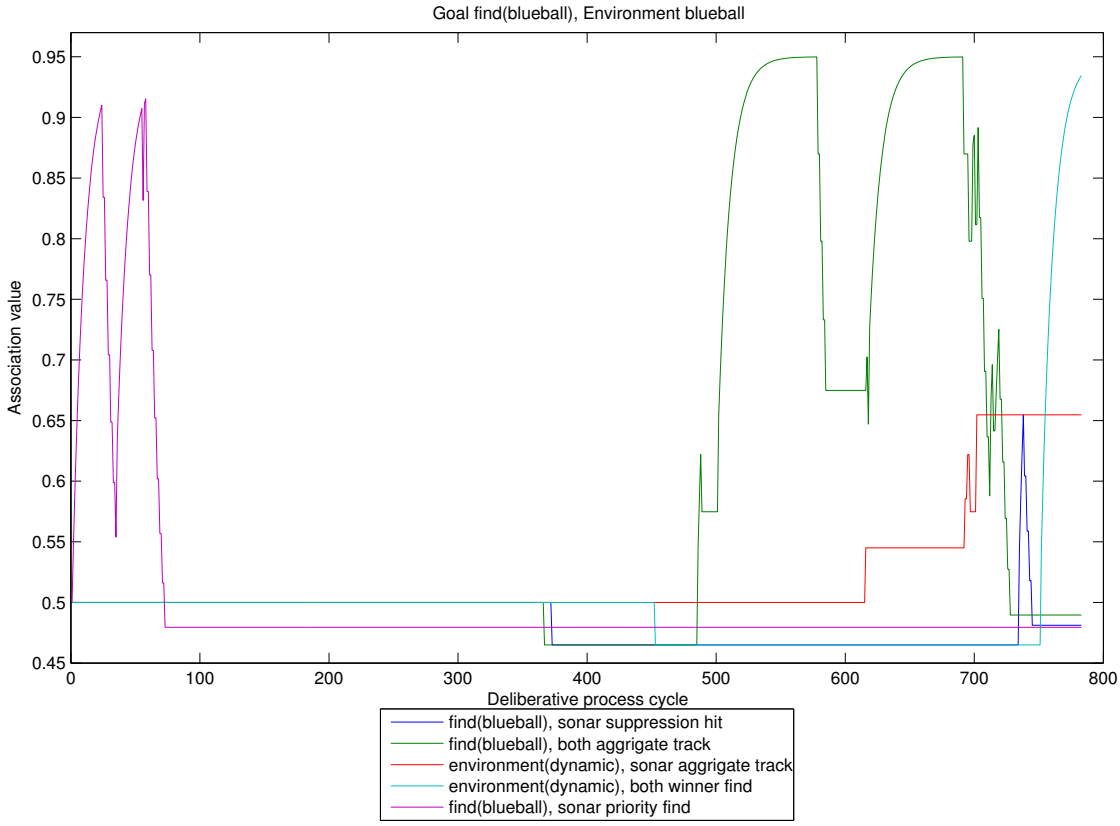


Figure 10.15: Associations in the *find(blueball)* learning experiment.

experiment the *avoid(object)* goal was chosen. This was to allow the result to be compared with data obtained from the *avoid(object)* control experiment (see section 10.2.2).

Robo-CAMAL was pre-programmed with the goal *avoid(object)*. It was pre-programmed with the correct environment beliefs and the associations with which to achieve its goal. There are twelve reactive architectures designed to achieve the goal, and two possible environment beliefs for each experiment. The possible environment beliefs are either dynamic or static, and sparse or cluttered. This gives 24 associations for each experiment.

Robo-CAMAL was run three times for three minutes in each possible environment. Each environment contained up to four objects. The environment could also be either static or dynamic. This provided a total of nine possible environments.

The value of each association was recorded at each deliberative processing cycle. An association was considered to be an accurate belief-desire-intention mapping if its value rose above a threshold of 0.75 for a total time of one minute during an experimental run. These values were chosen based on the results of the first learning experiment. This set up allows robo-CAMAL to learn a reasonable number of correct associations to the required accuracy. The results can be seen in table 10.10.

In terms of overall results, table 10.10 shows that in general the priority and aggregate behaviour combination methods perform the best. It also seems to indicate that the reactive architectures that use the sonar sensors exclusively, out perform those that use vision and the combination of vision plus sonar. These results are surprising as the control experiments indicated that the priority combination method using just the sonar sensor, did not perform as well as the other possible architectures.

Reactive architecture		**Success count**
Sensor	*Arbitration*	
Vision only	Priority	0
	Aggregate	14
	Winner	2
	Suppression	3
Sonar only	Priority	15
	Aggregate	5
	Winner	2
	Suppression	0
Sonar and Vision	Priority	4
	Aggregate	12
	Winner	3
	Suppression	0

Table 10.10: Success count for each reactive architecture.

The reason for this is the nature of robo-CAMAL's learning mechanism, and the way in which the associations are pre-programmed. Robo-CAMAL learns in an opportunistic way. If an association is successful, it continues to use that association. This means that the first association to succeed will be used until it fails. The first association used by robo-CAMAL in all the experiments always contained the intention *architecture(sonar, priority, avoid).* This can be seen in figures 10.16 to 10.18.

The results show that robo-CAMAL initially uses the association containing the intention *architecture(sonar, priority, avoid).* This is because it is the first on the list. If this association is successful, robo-CAMAL will continue to use it to the exclusion of all other associations. Only when robo-CAMAL fails enough times for the association value to fall, will it try an alternative association. This effect skews the results in favour of associations containing the intention *architecture(sonar, priority, avoid).*

Table 10.11 shows the results for the behaviour combination methods.

Again, the results contradict the control experiments. Where the

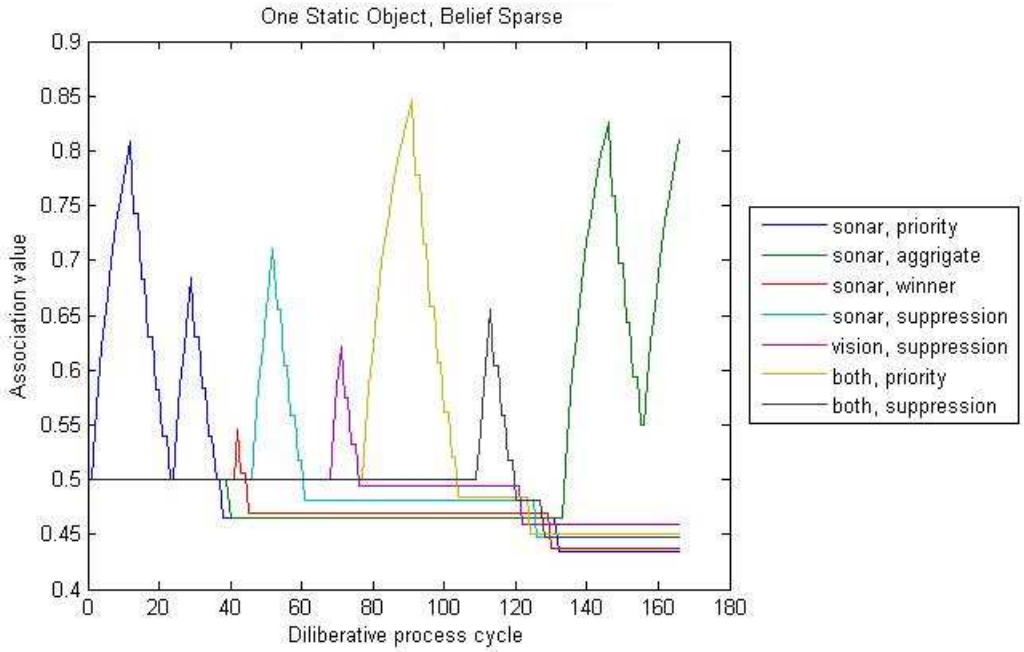


Figure 10.16: Environment 1 static object, association belief *environment(sparse)*.

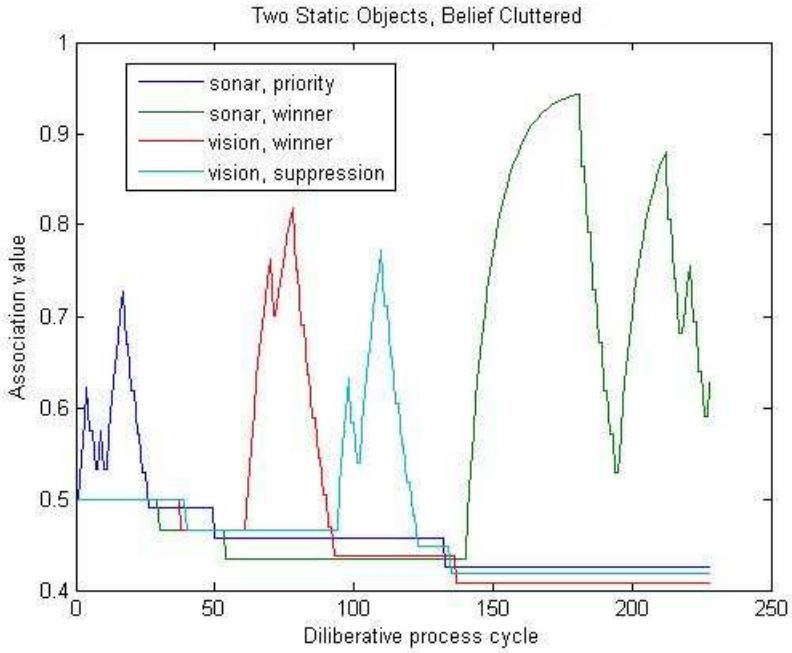


Figure 10.17: Environment 2 static objects, association belief *environment(cluttered)*.

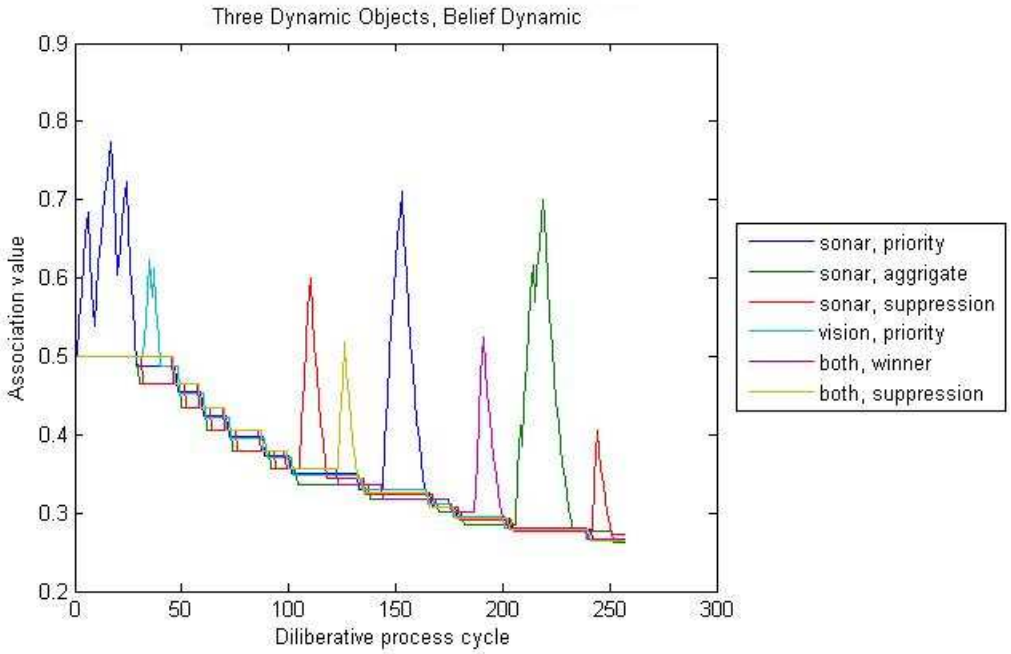


Figure 10.18: Environment 3 dynamic objects, association belief *environment(dynamic)*.

Arbitration method	**Success count**
Priority	19
Aggregate	31
Winner	7
Suppression	3

Table 10.11: Success count for each arbitration method.

control experiments show the best combination methods are aggregate and winner, this experiment shows that priority and aggregate perform the best. This result is again due to the nature of robo-CAMAL's learning mechanism.

10.4 Adaptation Experiments

The adaptation results were designed to determine if robo-CAMAL has the ability to modify its goals to reflect the changes in its environment.

The robo-CAMAL architecture was instantiated with the three

goals *hit(blueball)*, *hit(redrobot)*, and *hit(blackrobot)*. The correct associations were also pre-programmed into the architecture. RoboCAMAL was then allowed to run for three minutes in a variable environment. The environment contained any combination of the three possible objects *blueball*, *redrobot*, and *blackrobot*. The object combination was changed at intervals of one minute.

At each deliberative cycle, the agent's internal deliberative state was recorded. This included the agent's current beliefs, the association values, and the goal importance values. In addition, the number of actual collisions with the objects present was recorded. Some of the results of this experiment can be seen in figures 10.19 to 10.24.

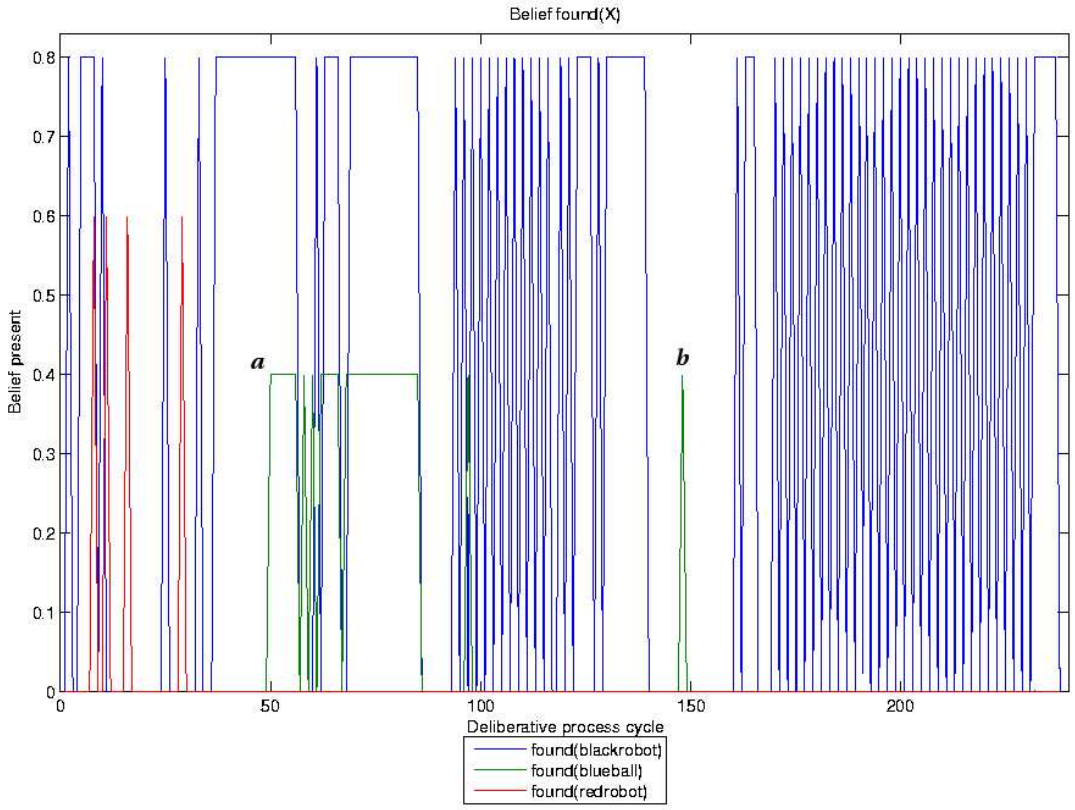


Figure 10.19: Periods when the belief *found(X)* was present.

Figure 10.19 shows the various *found(X)* beliefs present during one of the experimental runs. Each line represents one of the possible *found* beliefs. If a belief is present, it was given an arbitrary value used

for display purposes only, this value has no effect on the architecture. If the *found(X)* belief is not present, the value is zero. Figure 10.19 shows that initially robo-CAMAL holds the beliefs *found(redrobot)* and *found(blackrobot)* at various times. Then at point **a**, the belief *found(blueball)* is present. It is around this point that the *found(redrobot)* belief is no longer present. At point **b** the belief *found(blueball)* is no longer held.

This coincides with the way the objects were varied within the environment. At first the *redrobot* and the *blackball* were present in the environment. After one minute the *redrobot* was replaced with the *blueball*. After the second minute the *blueball* was removed. This mirrors figure 10.19 in that the *redrobot* was found at the start of the run, the *blueball* was found during the middle of the run, and the *blackrobot* was present thought the whole of the run.

One point to note is the spacing of the *found* beliefs. The total number of deliberative processing cycles for this three minute experiment is around 250. As the environment was changed after each minute, it would be expected that the beliefs would alter to reflect that change at around 80 and 160 deliberative processing cycles. This however is not the case. The *found(X)* belief alters its profile at around 50 and 150 deliberative processing cycles. This is due to the way in which the deliberative and reactive levels of the architecture interact.

The deliberative component sets the reactive component to run for a number of reactive cycles. The reactivate level returns control when an event occurs, or when it completes the required number of reactive cycles. This means that when there are few events occurring in the agent's local vicinity, the deliberative component is not as active. Therefore at the deliberative level, the number of processing cycles per minute is dependent on the number of events that occur within that minute.

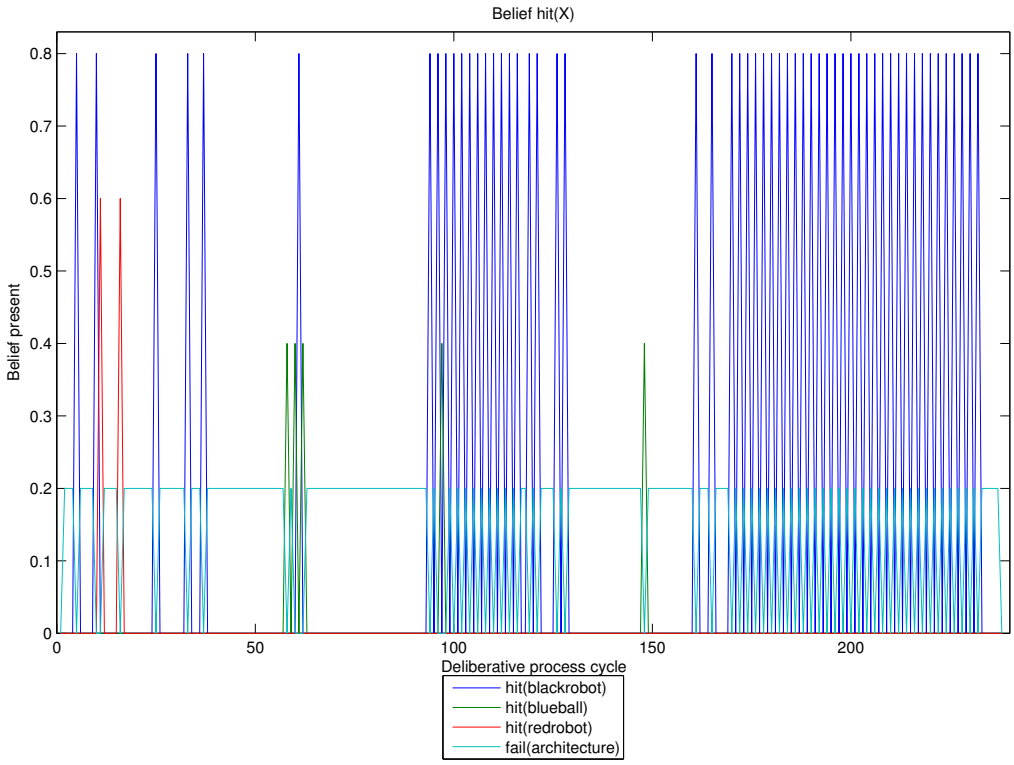


Figure 10.20: Periods when the belief *hit(X)* was present.

This can be seen in figure 10.20. This shows when robo-CAMAL believes that it has hit an object. This figure shows little activity for the first 100 processing cycles. After this point, the agent believes that it keeps hitting objects. In the first 100 processing cycles the agent fails its goals more often than it succeeds. In reality, during the first minute the agent hit the *redrobot* once and the *blackrobot* twice. In the second minute the agent hit the *blueball* twice and the *blackrobot* three times. In the final minute, robo-CAMAL hit the *blackrobot* twice.

The reason the agent appears to believe that it has hit the various objects more often then the actual hit count is due to the collision process. During a single collision event the object remains close to robo-CAMAL. The reactive level continues to register a *hit* event which is communicated to the deliberative level. This means the *hit* event is registered a number of times in quick succession. This means that

control is passed between the two levels in quick succession. Hence the variability with which the deliberative component registers the flow of time. The more events registered by the reactive level, the greater the number of deliberative processing cycles per minute. The addition of a touch sensor would rectify this.

Figure 10.21 shows the importance value for each goal over time. Initially robo-CAMAL achieves the hit *blackrobot* and *redrobot* goals. This can be seen at points **a** and **b**. At these points the importance value for each goal jumps to a value of 0.52. Section 8.6 states that once a goal has been achieved, its importance value is set to 0.5. The step increase for the goal importance value is 0.02. The goal importance value does not get recorded until after it has been incremented. This means that when the importance value jumps to, or remains at 0.52, the relevant goal has been achieved. In addition, if a goal fails, its importance value is reduced to 0.1. Therefore, in figure 10.21 if the goal importance value drops to 0.12 that goal has failed. This failure can be seen at points **c**, **d**, and **e** where each of the three goals fail. Point **f** shows the success of the goal *hit(blueball)*. This corresponds well with figure 10.20 as the *redrobot* was removed and the *blueball* added.

An important point to note can be seen at point **g**. Here the importance value of the goal *hit(blueball)* increases beyond 0.7. One question is why has robo-CAMAL not attempted to achieve this goal when its importance value is so high in comparison to the others? The answer can be found by examining the association values. These can be seen in figures 10.22 to 10.24.

Section 8.6 states that the motivator is chosen based on not only the goal importance, but the likelihood of successfully achieving the goal (i.e. the association with the highest value and the corresponding goal and current belief). Figures 10.22 and 10.23 clearly show that

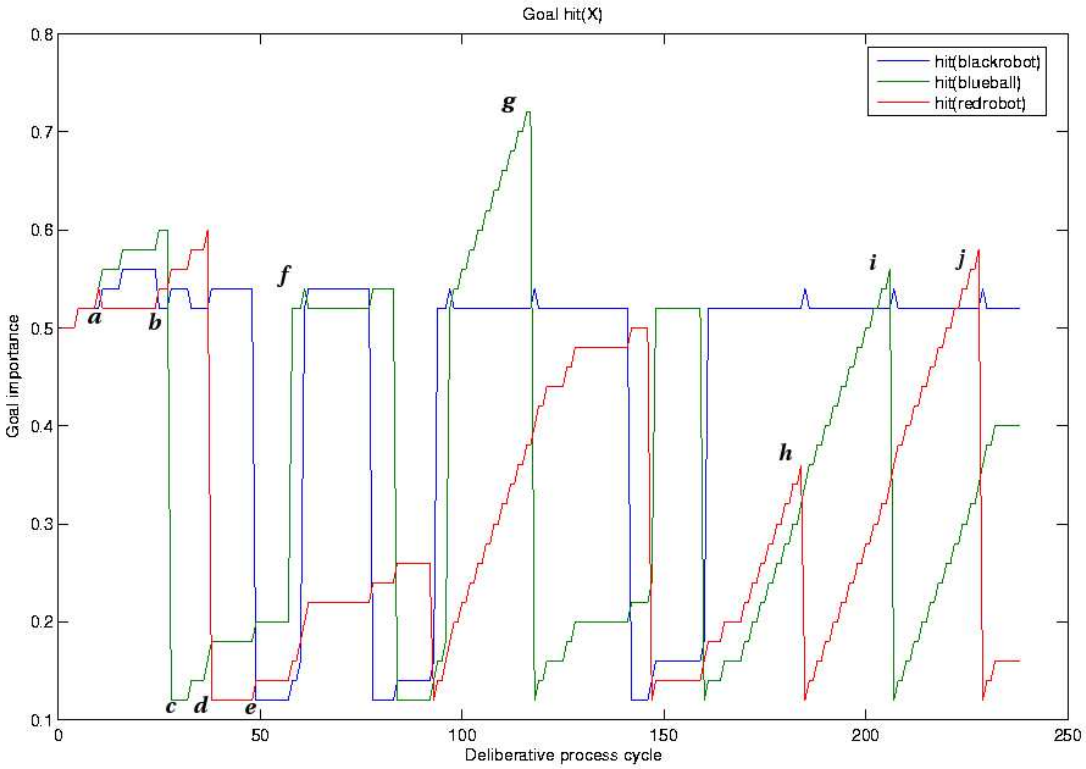


Figure 10.21: Goal importance value.

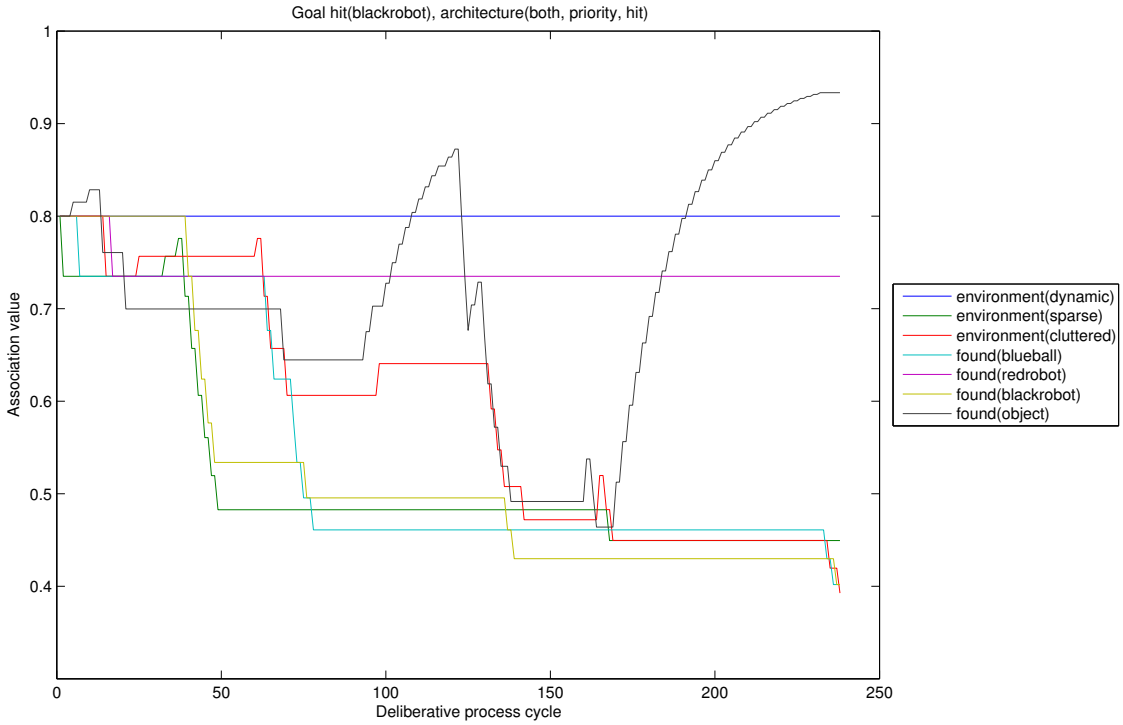


Figure 10.22: *hit(blackrobot)* association value.

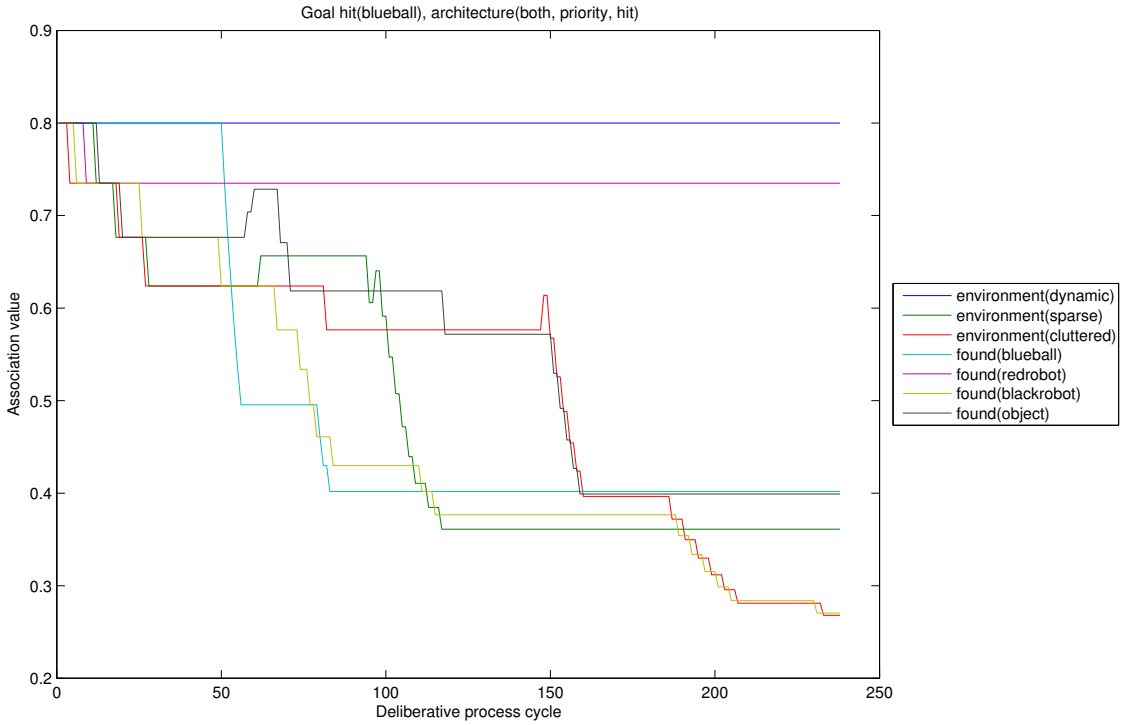


Figure 10.23: *hit(blueball)* association value.

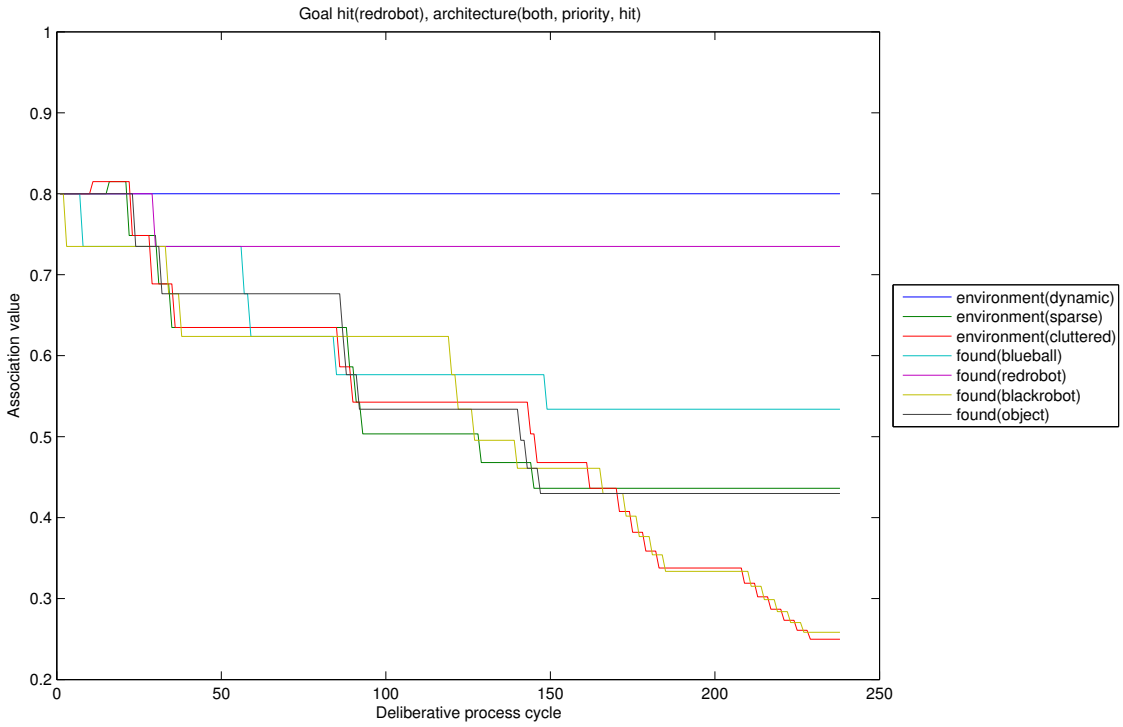


Figure 10.24: *hit(redrobot)* association value.

the association value related to the goal *hit(blackrobot)* is significantly higher than the one related to the goal *hit(blueball)*. This means that when combining the two values, the motivation to hit the *blackrobot* is greater than the one to hit the *blueball*. Therefore robo-CAMAL continues to attempt the goal *hit(blackrobot)*.

This result is as expected. Even though robo-CAMAL believes, correctly, that the *blueball* is present in the environment, it also believes that it is more likely to achieve the goal *hit(blackrobot)*. This was the reason for including the association value in the mechanism to decide robo-CAMAL's current motivation.

The same reasoning can be applied to robo-CAMAL's behaviour after 150 processing cycles. It is clear from 10.21 that at points **h**, **i**, and **j**, robo-CAMAL attempts the other goals. At these points the goal importance value becomes so great that it outbalances the low

association values. This result is again expected and was the reason for incrementing the goal importance value. This ensures that robo-CAMAL periodically attempts a previously failed goal to see if it has become achievable.

These result show that robo-CAMAL has the ability to adapt to a variable environment, and attempt the goals it believes achievable at the right time. However, not every result shows this kind of excellent performance. This can be seen in figures 10.25 to 10.30.

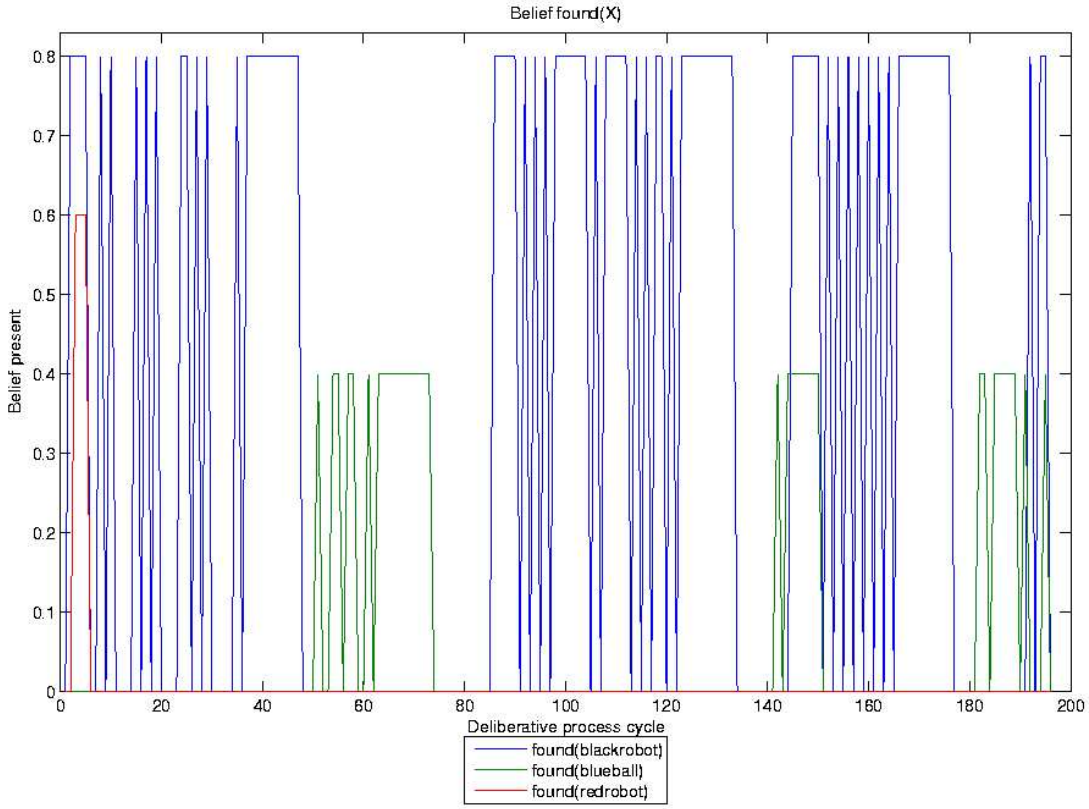


Figure 10.25: Periods when the belief *found(X)* was present.

Figures 10.27 to 10.30 show the goal importance values and the various *hit(object)* associations for another experimental run. These results seem to show a similar behaviour to the experimental run shown in figures 10.21 to 10.24. As robo-CAMAL continues to believe it has hit the *blackrobot*, the appropriate association value remains high so the belief that a *blackrobot* is present is reinforced. The problem

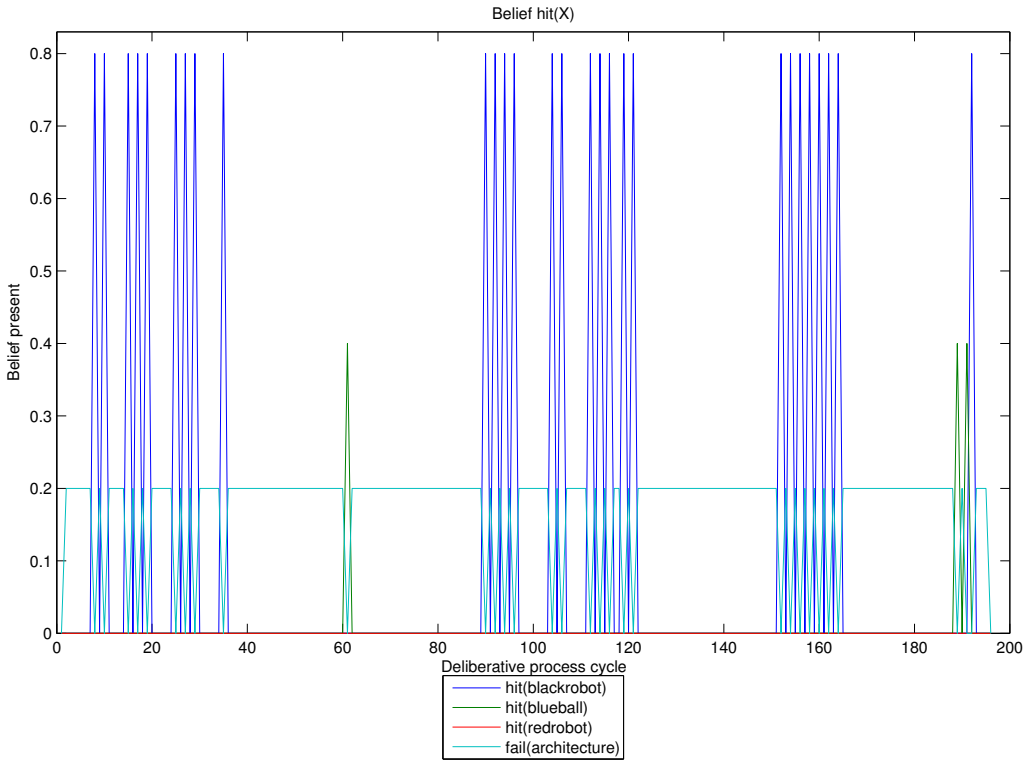


Figure 10.26: Periods when the belief *hit(X)* was present.

however is highlighted when the agent's beliefs are examined with reference to the actual objects present. These beliefs are shown in figures 10.25 and 10.26.

In this experiment the objects present were the *redrobot* and the *blackrobot* in the first minute, the *blueball* in the second and third minute. However, it can clearly be seen that at point **a** in figure 10.27 that robo-CAMAL believes it has hit the *blackrobot*, even though it is not present at that time.

This result shows two things. The first is that robo-CAMAL has difficulty distinguishing between the *blueball* and the *blackrobot*. This reinforces the findings shown in section 10.2.1. The second is more fundamental to robo-CAMAL's architecture as a whole.

Figures 10.27 to 10.30 show that robo-CAMAL is successfully adapting its goals in a variable environment. However, figures 10.25 and

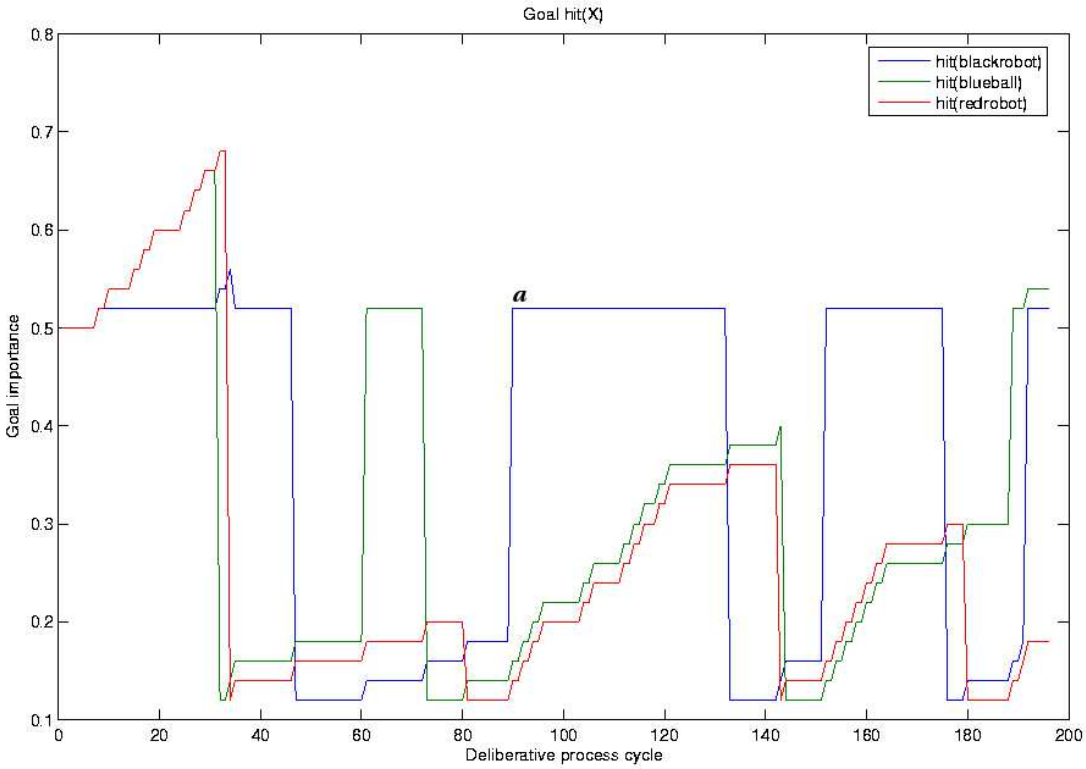


Figure 10.27: Goal importance value.

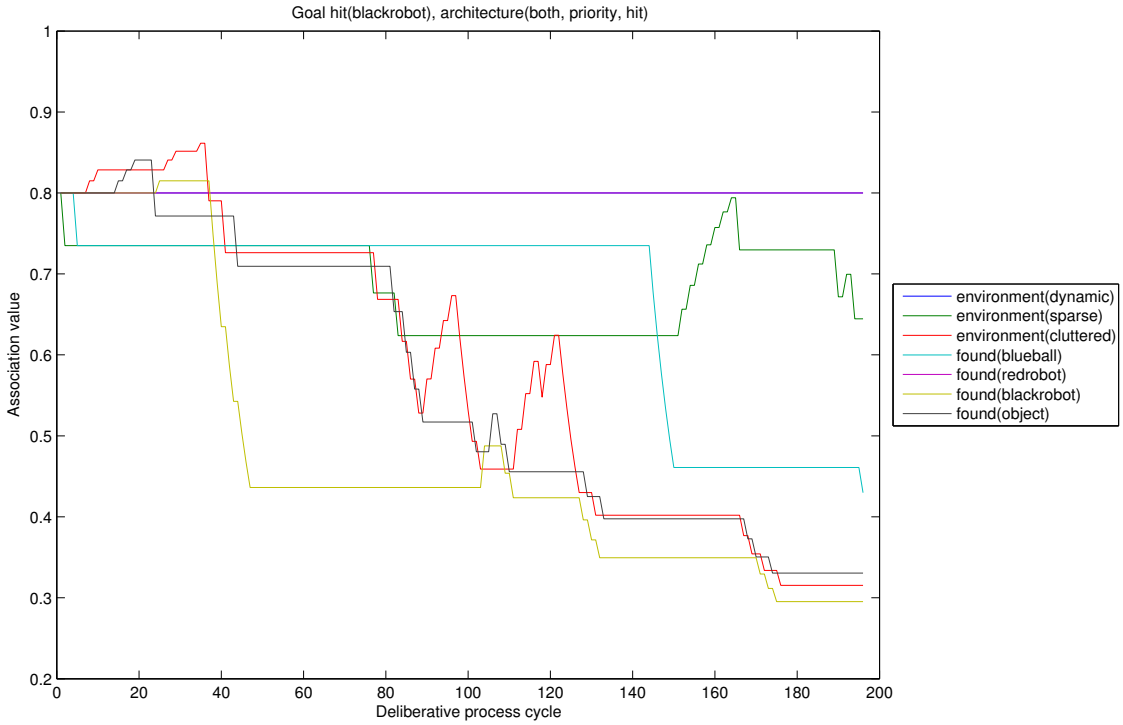


Figure 10.28: *hit(blackrobot)* association value.

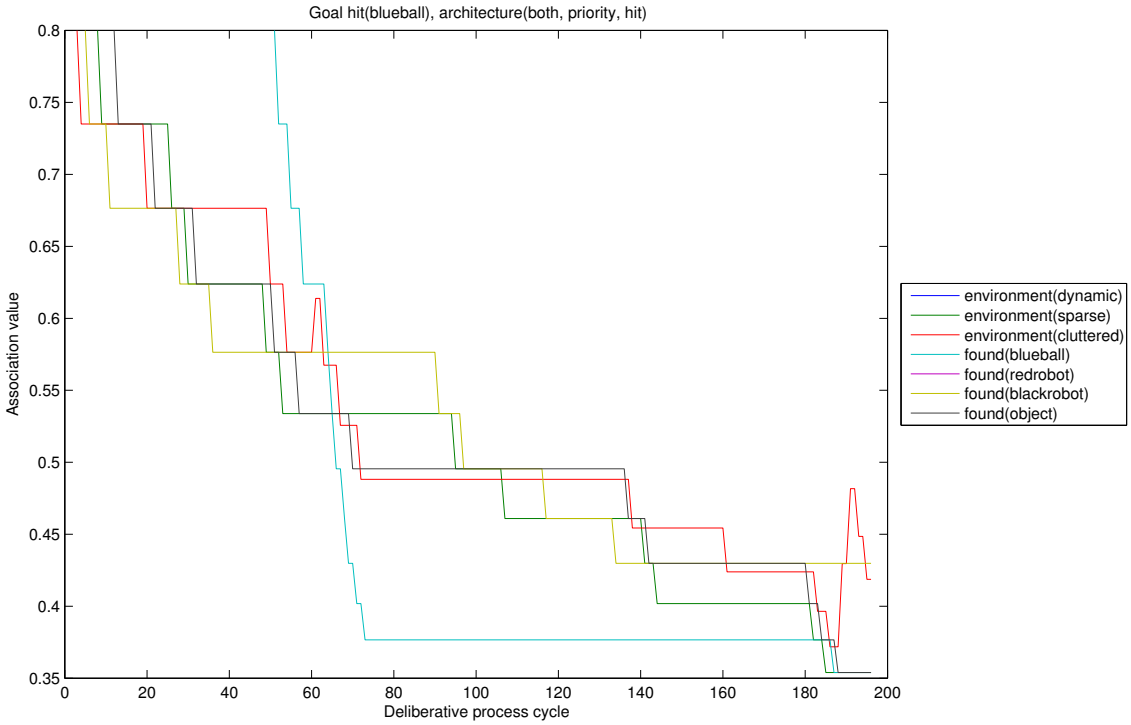


Figure 10.29: *hit(blueball)* association value.

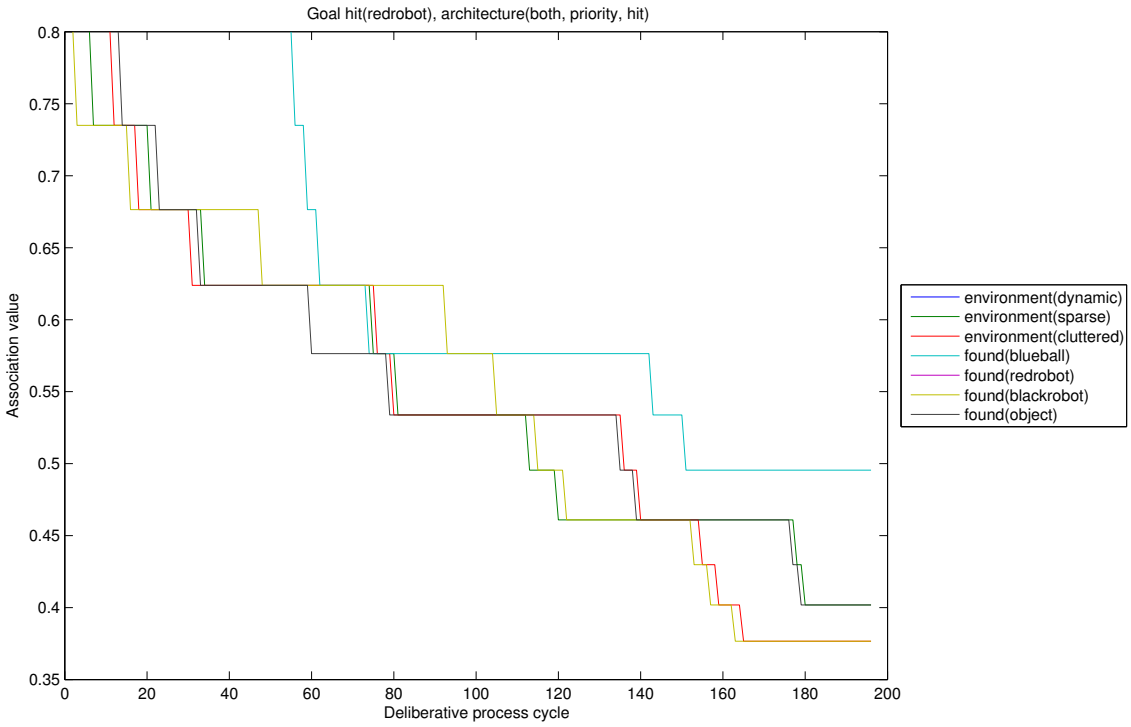


Figure 10.30: *hit(redrobot)* association value.

10.26 show that it is attempting to adapt to the wrong environment. This shows that robo-CAMAL has difficulty recovering from a false belief.

In figures 10.19 to 10.24, robo-CAMAL holds the correct beliefs about its environment. As it adapts correctly to this variable environment it achieves its goals. This in turn reinforces its original belief about its environment. In figures 10.25 to 10.30, robo-CAMAL holds incorrect beliefs about its environment. As robo-CAMAL adapts to the wrong environment, and believes it has successfully achieved its goals, these incorrect beliefs become reinforced.

Table 10.12 shows the number of times robo-CAMAL held the correct and incorrect beliefs during the adaptation experiments.

A false belief was registered if the agent believed that it had hit an object that was not present in the environment at that time. Only

Object	Correct hit belief	Incorrect hit belief
all	27	8
redrobot	5	1
blackrobot	10	5
blueball	12	2

Table 10.12: Correct and incorrect hit belief.

the first hit was recorded as it is this belief that triggers the cycle of robo-CAMAL's false belief reinforcement.

These results show that in general robo-CAMAL holds the correct beliefs, and therefore is adapting correctly to its environment. However, it often mistakenly believes it has hit the *blackrobot* on a number of occasions.

This table does not however give the full picture of the experiment. Of the six experiments performed, two showed that robo-CAMAL failed to adapt correctly to the actual environment. On only two occasions did robo-CAMAL hold the correct beliefs for the duration of the experiment.

The adaptation results can only be considered a partial success. Although robo-CAMAL can successfully adapt to its environment if it is maintaining the correct beliefs, it has great difficulty in recovering from incorrect beliefs. This result demonstrates just how important an agent's anchoring mechanism is to its ability to adapt to its environment. If the agent can not correctly identify, or miss-classifies, the object of its goals, then it will be unable to adapt to changes within its environment.

10.5 Conclusion

This chapter described the various experiments performed with robo-CAMAL, and their results. It started by describing the control experiments. These were designed to ensure that robo-CAMAL operated

as designed, and to provide performance data with which to compare the other experiments.

The various control experiments showed that at the deliberative level the association value, generation, and choice mechanisms worked as expected. It also showed that the belief and goal mechanisms worked as expected. This included the belief creation and removal process, and the goal importance and threat mechanism.

At the reactive level the control experiments showed that the various macro and micro behaviours worked as designed. It also showed that the vision system could differentiate between the various objects in the environment. However, the experiments also showed that on occasion the *blueball* gets mistaken for the *blackrobot*.

In addition the control experiments provided data with which to investigate robo-CAMAL's learning capability. This showed that in general the architecture making use of the vision system and the aggregate and winner combination methods were the best at avoiding objects. It also showed the architecture that makes use of sonar and the suppression combination method performed poorly.

The second set of experiments investigated the learning capability of robo-CAMAL. The first experiment demonstrated that robo-CAMAL is capable of learning general behaviours. That is, with no understanding about the nature of its actions, robo-CAMAL can successfully learn which action to perform in order to achieve a specific goal. This is accomplished by performing and observing the effect of its actions on the environment.

The second learning experiment was designed to determine if robo-CAMAL can optimise its behaviour. That is, given a list of actions that attempt to achieve the same thing, can robo-CAMAL learn which is the best. This experiment showed that robo-CAMAL was unable to learn the optimal action with which to achieve its goal. This is due

to the opportunistic way in which robo-CAMAL learns.

The final experiments were designed to test robo-CAMAL's ability to adapt to a variable environment. This experiment showed limited success. Robo-CAMAL successfully adapted to its environment when it held the correct beliefs. However, when the agent formed the incorrect beliefs it failed to correctly modify its goals to reflect the actual environment. Instead it modified its goals to coincide with its incorrect beliefs.

In general the results showed that robo-CAMAL performed as expected. However, robo-CAMAL failed in two ways. The first was due to the opportunistic learning mechanism implemented. This meant that robo-CAMAL was unable to optimise its behaviour. The second failure was due to the limitations of robo-CAMAL's anchoring mechanism. In some instances, robo-CAMAL mistook the *blueball* for the *blackrobot*. When this occurred the agent failed to modify its goals to reflect the actual environment.

Chapter 11

Analysis and Discussion

11.1 Introduction

This chapter will analyse the results shown in chapter 10, and describe their impact on the robo-CAMAL architecture. It will start by looking at robo-CAMAL's anchoring mechanism. It will relate this mechanism to various other proposed theories regarding the anchoring of events and objects.

Section 11.3 will look at robo-CAMAL's learning mechanism. This will look at how robo-CAMAL's opportunistic method of learning effected its performance. It will then compare robo-CAMAL's learning performance with an alternative system that makes use of the same learning algorithm.

Section 11.4 will relate how robo-CAMAL uses the adaptation mechanisms described in section 8.6. It will then show why this model on its own shows only limited success within robo-CAMAL.

The next section discusses the nature of robo-CAMAL's motivation. Section 9.3 described how motivation is grounded through its interactions with its environment. It also discussed the criteria for determining if a motivation is purposeful and deliberative. Section 11.5 will therefore judge robo-CAMAL's motivation in relation to these criteria.

Finally section 11.6 will describe some possible additions to the robo-CAMAL architecture to improve its overall performance and functionality. This will include a more sophisticated and variable domain model, as well as a meta cognitive layer.

11.2 Robo-CAMAL's Anchoring Mechanism

The key component that allows robo-CAMAL to operate within its environment is its anchoring mechanism. Section 1.2.1 states that anchoring is the process of creating and maintaining a link between an internal symbol and a perceptual response due to an object or event (Coradeschi and Saffiotti, 2003). Within robo-CAMAL this process is achieved through the domain model.

The domain model is a set of statements that allow specific characteristics of the agent's environment to be instantiated within the agent's architecture. Within robo-CAMAL the domain model is present throughout the architecture. For example at the reactive level the colour profiles are considered part of the domain model. At the deliberative level the specific beliefs the agent can hold are a component of the domain model (see appendix B for more details on which components are part of the domain model). As all the symbols are instantiated a-priori, there is no symbol grounding performed within robo-CAMAL.

11.2.1 Anchoring Objects with Robo-CAMAL

The anchoring mechanism within robo-CAMAL is reviewed here, and explained by describing the process with which an object is linked to the appropriate symbol. Initially an object produces a response in the agent's vision system. The vision system runs a Sobel operator over the image to produce an edge image. The edge image is used to detect

the location of the object.

The leading edge of the object is determined by placing an edge node in each radial segment surrounding the agent. It is placed at the pixel that is closest to the agent, and that exceeds a predefined threshold value. The radial segment size, the edge pixel threshold, and the assumption that the closest edge node is the objects leading edge are all part of the domain model. This process detects the location of an area of interest to the agent.

The area of interest (AOI) to the agent is taken from the edge node closest to the agent. This AOI can then be extracted for further analysis. This extraction process can be seen in figure 5.15 (see section 5.6 for more details). The size of the AOI is determined by the domain model.

Once the AOI extraction has taken place, the RGB value of every pixel is compared to the pre-defined RGB profile of the various objects. If a pixel value is contained within a particular profile, then the profile pixel count is incremented by one. If the pixel count of a particular profile exceeds a pre-defined threshold, then the appropriate profile flag is activated. The various object profiles, and the pixel count threshold for each profile, are a part of the domain model.

At every reactive processing cycle the agent's perceptual module checks the profile flags. The perceptual module then constructs the appropriate statements based on the state of the profile flags, and the proximity of the AOI. This statement construction is based on a set of predefined rules. For example if the *redrobot* flag is checked, and the AOI is closer than a pre-defined threshold value, then part of the constructed statement contains the event and object values *hit* and *redrobot*. The rules that govern the construction of the statement are pre-defined, and are a component of the domain model. Although these values make sense to a human reader, they are still just values

to robo-CAMAL at this point. This means that one more step is required before an object can be considered anchored. This final step is achieved within the deliberative level.

The deliberative level receives the constructed statement and uses its content, and a number of pre-defined symbols, to generate beliefs that reflect the state of the environment. For example, if the received statement contains the values *hit* and *redrobot*, then the beliefs *hit(redrobot)* and *found(redrobot)* are constructed. The belief *hit(redrobot)* is removed after one deliberative processing cycle. The belief *found(redrobot)* is removed after 25 deliberative processing cycles. These pre-defined event and object symbols are a component of the domain model. Once the appropriate beliefs have been constructed at the deliberative level, the object can be considered anchored.

Another example of how the anchoring of an object can be achieved was given in section 7.3.4. This made use of fuzzy logic to identify an object (Coradeschi and Saffiotti, 1999). In essence, each object has a specific set of features. The degree of matching between the perceptual systems response to an object and the feature set gives a measure of the likelihood that an object belongs to that feature set. In this sense robo-CAMAL's anchoring mechanism is similar to other proposed anchoring mechanisms. Only the specific details of the object identification really differ.

One thing the robo-CAMAL architecture highlights, and in particular the adaptation experiment, is the importance of the initial identification. This means that choosing the correct features with which to identify an object, and extracting those features from the perceptual system, is crucial to the anchoring process. The adaptation results in section 10.4 clearly show that the hardest part of the anchoring process is this initial object identification. For example, robo-CAMAL only used colour to identify objects. However, colour is

subject to lighting conditions, with shadows and bright spots affecting the objects colour profile significantly. For this reason robo-CAMAL was not always successful in correctly identifying an object.

The results clearly show that if robo-CAMAL correctly identifies the objects in its environment, it can adapt its goals and form the correct beliefs. This demonstrates that robo-CAMAL can successfully anchor objects to predefined symbols. However, if robo-CAMAL incorrectly classifies an object, it attempts to achieve inappropriate goals and forms the wrong beliefs. This shows that if a deliberative agent can not successfully anchor objects to the appropriate symbols, it can not function correctly in that environment.

This demonstrates how reliant the deliberative component is on the information provided by the perceptual system. In robo-CAMAL the fault of unsuccessful anchoring lies not with the deliberative level, but with the initial identification process. This means the features used to identify an object are crucial. This also includes the way in which those features are extracted. Both these elements are a key component of an agent's domain model.

11.2.2 Anchoring Events with Robo-CAMAL

In addition to anchoring objects, robo-CAMAL also has the ability to anchor events. This is done via a two step process. Initially robo-CAMAL's environment is in state S_1. The agent performs an action A_1 with known consequences. This changes the environment to state S_2. As the agent can predict the consequences of the action, if it observes the environment changing from S_1 to S_2, it knows that event E_1 occurred. For example, S_1 = *environment(sparse)* and A_1 = *find(redrobot)*. If after A_1 the environment is observed to be S_2 = *near(redrobot)* then the event E_1 = *found(redrobot)* occurred. In this way the event *found* can said to be anchored.

Another system that has the ability to anchor events and actions makes use of situation/event calculus to observe a robotic finger that strikes a ball (Steels and Baillie, 2003). Section 2.6 stated that event calculus is a formalism for representing events and their effects (Kowalski and Sergot, 1986). The event calculus contains three types of elements, events or actions, fluents, and time points. A fluent is a value that has the capacity to change over time. These elements can be reasoned about using various statements.

In this system the finger has three moving components. Each component can be represented by the three vectors, k_a, k_b, and k_c. The ball can be represented by vector k_d.

Initially the camera observes the environment with the finger stationary in state $S_0 = (k_a, k_b, k_c, k_d)$. When the camera observes the finger starting to move the environment state becomes $S_1 = (k'_a, k'_b, k'_c, k_d)$. This transition corresponds to the action $start\_move\_finger$. That is $S_1 = do(start\_move\_finger, S_0)$.

At some time the finger strikes the ball. This gives rise to the state $S_2 = (k'_a, k'_b, k'_c, k'_d)$. This corresponds to the event hit_ball, i.e. $S_2 = do(hit\_ball, S_1)$. This clearly shows that the system anchors actions and events by observing the initial and new state of the environment.

The way robo-CAMAL anchors events is completely different to the event calculus method. However, both systems are required to observe the initial and subsequent state of the environment arising from an event in order to anchor that event.

11.3 Robo-CAMAL's Learning Mechanism

If an agent is to achieve its goal, it needs to learn, plan, and execute the appropriate set of actions to do so. Execution deals with the instigation of an action, and observing the results on the environment

(Martínez and Borrajo, 2000). Learning is an agent's ability to use past experience and percepts to improve its performance in the future (Zimmerman and Kambhampati, 2003). Planning is the process of generating representations of future actions, prior to the use of such actions, to control behaviour (Tate, 1999). That is, an agent couples a set of known actions in a specific order in such a way that it achieves its goal.

Robo-CAMAL makes no use of a planning mechanism (This statement relates to robo-CAMAL only. The CAMAL architecture does include a planning mechanism. See section 11.6.2 for more details). It does, however, make use of execution and learning mechanisms in order to achieve its goals.

In terms of robo-CAMAL's learning ability, it uses its anchoring mechanism to identify the objectives of its goal, and the association model to understand the consequences of its actions. As mentioned in section 8.2, an association is a coupling between a belief, a desire, an intention, and a magnitude value (i.e. the association value).

At a high level view of the learning mechanism, robo-CAMAL goes through a number of phases. These are:

- observe the environment (using the anchoring mechanism).
- initiate an association (using the affect model)
- perform the action described by the association (using the domain model details of the link between the intention symbol and the reactive control system)
- observe the environment once the action has been performed (using the anchoring mechanism)
- feed the consequences of the action on the environment into the association value (using the association model)

If the observed state of the environment after the action conforms to the required desire state, then the association mechanism increases the association value. However, if the environmental state after the action is not the required goal state, the association value is decreased. This process ensures that successful associations will develop higher association values, whilst unsuccessful associations will develop lower values.

11.3.1 Reinforcement Learning

Robo-CAMAL makes use of a reinforcement mechanism. It uses the association model to choose an action. It then executes that action and observes the the results on the environment. These observations get fed back into the association model and the cycle continues. There are a number of agents that use a similar approach. One such system makes use of a Q-learning algorithm (as described in section 8.5.1) to enable a robot to hit a ball into a goal (Asada et al., 1994).

This system's algorithm was trained within a computer simulation. Two experiments were conducted. The first simulated a two video frame (1/15 sec) delay. This was done to simulate the physical agent's image processing routine. The second experiment simulated no delay. Each training phase was run for 1000 processing steps. Once the algorithm was trained the simulated agent was able to score a goal with a success rate of 60% for the video delay policy, and 70% for the no delay policy. This result clearly shows that a delayed response in the vision system of an agent has a noticeable effect on its learning ability.

The algorithm trained using the no delay policy was then transferred to a physical agent attempting to hit a red ball into a goal. The shooting rate of the physical agent was 40%, which is significantly lower than the simulated results. This clearly shows the difficulties of

operating in a physical environment as opposed to a simulated one.

It is hard to fully compare this system with robo-CAMAL for two reasons. The first is that robo-CAMAL's actions were specifically designed to achieve its goal, where as the system described above had to learn how to combine basic actions (the equivalent of robo-CAMAL's micro-behaviours) to achieve its goal. In addition, the goal of hitting a ball into a goal is more complex than simply hitting an object. However, the control experiments for robo-CAMAL show that the *hit(object)* has a success rate of 80% (see figures 10.1 to 10.3), and the first learning experiment showed that robo-CAMAL is between 5 and 7 times more likely to learn the correct macro-behaviour to hit an object than the incorrect macro-behaviour (see table 10.5). This success comes despite robo-CAMAL having to operate in a physical environment, and cope with the omni-directionals video systems processing delay.

It was mentioned above that all robo-CAMAL's macro-behaviours are a pre-programmed combination of its micro-behaviours. However, this does not mean that robo-CAMAL is unable to learn how to combine micro-behaviours. In principle the learning of micro-behaviours, and how they can be combined, can be achieved using the existing mechanism. All that would be required is a modification of the domain model.

The initial step would be to set a hierarchy of goals. The lower goal set would contain basic goals such as *move(forward)*, *avoid(objectright)*, *avoid(objectleft)* etc. The higher set would contain more complex goals such as *hit(object)* etc. In addition, symbols linked to the micro-behaviours would be entered into the domain model. For example, spatial symbols such as *right*, *front* etc. may be necessary to determine the type of motion the agent has performed. Finally the perceptual systems would need to be enabled to determine if the goal state

has been achieved for simple goals, e.g. routines that would allow the perceptual system to determine forward motion. Once this has been achieved then robo-CAMAL could learn the effect of its micro-behaviours in the same way as it does for its macro-behaviours.

The second step would be to learn to link different micro-behaviours together to achieve more complex goals. Figure 11.1 shows a suggested goal hierarchy. In this hierarchy level 1 goals would represent basic functionality such as movement. Level 2 goals represent more complex behaviour based goals. A simple distinction could be that level 1 goals are considered reflexes, and level 2 goals are considered desires. In this hierarchy a level 2 goal could be any combination of the level 1 goals.

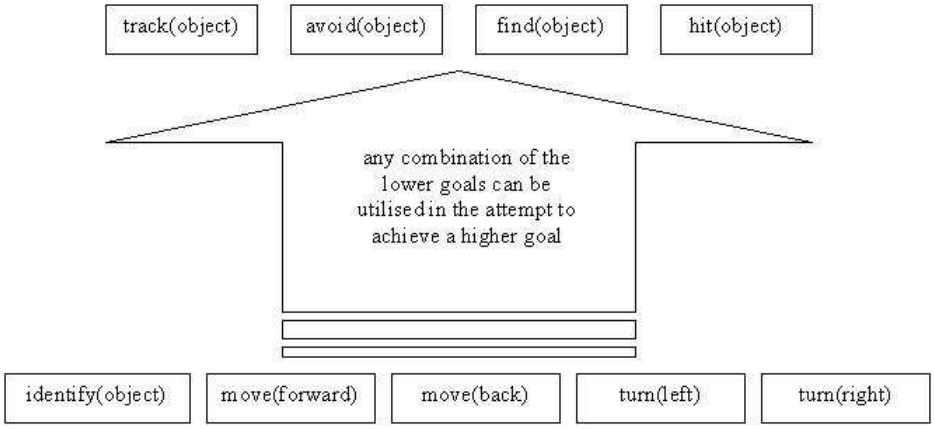


Figure 11.1: Goal hierarchy.

Currently robo-CAMAL makes use of symbols that relate to macro-behaviours. Robo-CAMAL then attempts to determine the most likely symbol to achieve its goal. The alternative approach is to use symbols that relate to the individual micro-behaviours. The agent then tries different combinations of micro-behaviours in order to achieve a goal. This process would, however, take considerably more time as there are a greater number possible combinations of micro-behaviours to test. This is because the possible number of micro-behaviours for each macro-behaviour is variable.

Section 10.3.2 mentioned that robo-CAMAL's opportunistic learn-

ing mechanism can have a detrimental effect on its learning capability. This refers to the agent's ability to choose the best action from a list of similar actions. The agent does not consider future success, nor does it attempt all associations in equal proportion. Instead, if it finds an association that works, it sticks with it. This means that robo-CAMAL shows a bias for the first successful association. As the first association generated is also the first attempted, robo-CAMAL will show an overall bias as it will try this association more often than the rest.

One proposed solution might be to randomise the association order. This would only hide the problem, not solve it. All this would do is shift the bias from the first generated association, to the first randomly generated association. The only way to overcome this deficiency would be to attempt each association for a set period of time. Once the time period has elapsed, the results are recorded and the next association is attempted.

11.4 Adaptation with robo-CAMAL

For an agent to operate successfully it must be able to adapt to its environment. Section 8.6 described a number of different ways in which a biological agent can adapt. These include learning, evolutionary, physiological, and sensory adaptation (McFarland, 1991).

Robo-CAMAL adapts to its environment in two ways. The first is that it learns the most appropriate associations for a given environmental situation and desire. The second is similar to the physiological mechanism employed by biological agents. That is, the internal state of an agent reflects its environment and its drives.

For example the hunger drive in a dog may be considered analogous to the goal importance level in robo-CAMAL. Also the pres-

ence of food may cause a dog to eat regardless of whether it is hungry. Likewise in robo-CAMAL, a high association value and the belief *near(redrobot)* causes it to hit the *redrobot* despite a low goal importance.

Parallels may exist between the physiological and robo-CAMAL's adaptation mechanisms. However, the distributed model of affect that represents the agent's internal state, is a gross oversimplification of similar systems found in biological agents.

One point to note is that robo-CAMAL's adaptation techniques are not linked together into one process. That is, robo-CAMAL can not switch between learning the appropriate actions, and then utilising them in a variable environment. The agent requires its learning and free roam attitudes to be set, off line, by a human user. The full process requires the user to set the agent's learning attitude with a single goal. Once the associations for various goals have been learned, they can then be used in a free roam attitude in which the agent has several goals set by the human user.

11.4.1 The Domain Model and Adaptation

Once robo-CAMAL has learned the correct associations, it has to use them to adapt in a variable environment. For robo-CAMAL to adapt in a free environment, it needs to achieve three things. It must observe the environment and form the correct beliefs, it must choose the appropriate action based on those beliefs and its internal state, and it must then execute that action.

Robo-CAMAL's anchoring mechanism deals with the formation of correct environmental beliefs. The BDI schema and the affect model are then responsible for choosing the appropriate action. Finally the reactive level is responsible for interpreting the control messages from the deliberative level, and executing that action.

It is clear from the results presented in section 10.4 that robo-CAMAL is not always successful in adapting to its environment. The control experiments in section 10.2.4 and 10.2.3 show that the deliberative component is operating as expected. This is backed up by the adaptation experiment which shows that robo-CAMAL is adapting to the environment that it believes it is in.

The execution of an action is a simple reactive process i.e. the reactive level de-constructs the deliberative control message according to a set of predefined rules. These rules are defined by the domain model. The only place in which the adaptation process falls down is in the forming of correct beliefs.

In the first learning experiment, robo-CAMAL was pre-programmed with the correct beliefs a-priori. In addition, the objects within the environment were never changed. With pre-programmed beliefs and a controlled environment, the difficulties of forming the correct beliefs are removed from the process of learning. This explains the success of the learning experiment, and the partial failure of the adaptation experiment.

This result also highlights the difficulties in moving an agent from a controlled to an uncontrolled environment. Though the jump is not as vast as moving from a simulated to an uncontrolled environment, it is still significant.

There are two reasons for robo-CAMAL's failure to form the correct beliefs in a variable environment. The first is its perceptual system. However, this point will always be true as sensors can always be improved. The main problem relates to the perceptual elements of the domain model. The image processing routines used were designed to be simple to increase their processing speed. As a consequence only one feature, colour, was used to identify objects. In addition, the method of object identification was very basic. This impeded

robo-CAMALs' ability to correctly identify objects in complex environments. In essence, the reason robo-CAMAL often forms incorrect beliefs is due to deficiencies in the perceptual domain model. That is, the perceptual domain model components do not capture the full nature of the agent's environment.

11.5 Grounding Motivation with Robo-CAMAL

Section 9.3 described how an agent's motivation is grounded through its interaction with its environment. It also provided a number of characteristics that can be used to determine if an agent's motivation is reactive or deliberative.

It is clear from the learning and adaptation results that robo-CAMAL's interaction with its environment provides it with the ability to choose its own goals. Robo-CAMAL uses its perceptual systems to form beliefs about its environment. Robo-CAMAL's internal state is determined by its current and previous beliefs. Robo-CAMAL then chooses its motivation based on its beliefs and its internal state. The chosen motivation triggers the appropriate action which is executed. Robo-CAMAL then observes the consequences of that action on the environment, and forms new beliefs based on those consequences. The cycle then repeats. This shows that robo-CAMAL's motivation is chosen through its continual interaction with its environment.

The nature of robo-CAMAL's motivation can be determined by using the indices of motivation model (Epstein, 1982). The first feature is the ability of an agent to achieve its goals using alternative strategies. This ability is inherent in the association model. The way in which robo-CAMAL learns allows it to develop different strategies to achieve its goal based on its beliefs. For example, the agent can have the associations

association(environment(cluttered), avoid(X), intention(1..n), values(1..n))
association(environment(sparse), avoid(X), intention(1..n), values(1..n))

Here the same goal is the focus of the two associations, but each has a unique belief. The agent has the ability to learn the most likely intention to achieve its goal. This means that robo-CAMAL has the ability to learn different intentions, or strategies, that achieve the same goal under different circumstances.

The second characteristic is the formation of *expectancies* relating to the goal object. This is reflected in robo-CAMAL by the possible object beliefs. For example, the belief *hit(redrobot)* is a cognitive representation of the state of a goal object after an action has been performed.

The final characteristic is the presence of an *affective* response to the goal object. In robo-CAMAL this is represented by the distributed model of affect. This includes the confidance the agent has in a belief regarding the goal object, the association values of the associations relating to the goal object, and the importance and threat values of goals involving the object.

From these arguments it can be claimed that robo-CAMAL's motivation is grounded and deliberative. The only possible doubt is robo-CAMAL's formation of *expectancies*. It could be argued that as the agent requires these *expectancies* to be pre-programmed via the domain model, its motivation is not fully deliberative. However, even if aspects of the domain model are pre-programmed, that does not make them non-deliberative. If the deliberative component sets the reactive architecture to hit the *blueball*, it expects the declarative *hit(blueball)* belief to be formed.

However, it is clear from the way in which robo-CAMAL adheres

to the *individuation* and *affective* response criteria, that its behaviour is far more deliberative than reactive.

11.6 Improving Robo-CAMAL

This section will describe the two main ways which could improve robo-CAMAL's ability to learn and adapt. The first suggestion relates to the improvement of the current anchoring mechanism employed. This can be done by improving the agent's object and event detection, and making use of a variable domain model.

The addition of a variable domain model introduces additional values that will fluctuate to reflect the agent's internal state. The fluctuation of these values must be controlled and monitored in some way. For this reason the second suggested improvement is a meta-cognitive level to monitor the internal state of the agent, and to modify the domain model accordingly.

11.6.1 Improving the Domain Model

The first major improvement that can be made to the perceptual component of the domain model is by improving the criteria for distinguishing an event. This can be achieved with better sensors and more accurate event descriptions.

The first key improvement looks at robo-CAMAL's collision detection mechanism. If an object is detected within a predefined proximity threshold, a collision event is recorded. This assumption allows the possibility that robo-CAMAL misclassify an object within this distance as being hit, when in fact no physical contact has been made.

This misclassification can easily be solved by the addition of pressure sensors or whiskers. The addition of such sensors means that a collision is only recorded at the point of contact. The one possible

drawback is the potential difficulty of ensuring that an object detected via the touch sensor corresponds to the object identified by the vision system. This would need to be done by comparing the location of the activated touch sensor to the area in the image of the identified object.

The second issue surrounding collision detection is slightly more ambiguous. This involves the registering of a single collision event multiple times. Figure 10.22 in section 10.4 shows that robo-CAMAL believes it has hit the *blackrobot* on a number of occasions. However, in reality the number of collisions with the *blackrobot* was only 7, far less than robo-CAMAL registered.

This is due to the fact that collisions take a finite amount of time, or that robo-CAMAL may trap the object in a corner. As the object is always within the collision distance threshold, the belief *hit(object)* is continually registered. Touch sensors would negate this problem to a large degree.

What should the definition of a collision be? For example, consider the *track(object)* goal. If the object is in close proximity to the agent, the belief *near(object)* is registered. As long as the agent is close to the object the *near(object)* will continue to be generated, which is exactly the requirement of the *track(object)* goal. This definition could also be implemented at the deliberative level in the belief formation. If Time = t_n and *hit(object,* t_n*)* and *hit(object,* t_{n-1}*)* then *touch(object,* t_n*)*

However, the *hit(object)* goal could mean one of two things.

- Continuous contact with an object
- Single contact with an object

The first definition is what is currently implemented within robo-CAMAL. That is, contact is continually registered as a hit event.

The second definition would require some additional processing at the reactive level. This would be simple to implement as a finite state machine with the form

IF $newPressure$ = off **AND** $previousPressure$ = on
THEN REGISTER *hit(object)*)

A temporal element may also be introduced. This would record the number of reactive processing cycles the pressure variable is on. All these possible rules provide a number of possible definitions for what constitutes a collision. It would also be possible to implement a number of possible definitions as separate events. For example *touch(object)* and *hit(object)* events could be added to the domain model if both the above definitions were implemented.

However, such additions are more of a design decision to expand, rather than improve, the architecture. This is because the definition of what constitutes a collision is imposed on the agent by an external observer in much the same way behaviour can be imposed on a reactive system.

To improve robo-CAMAL's ability to identify objects, the most obvious way would be to improve its vision system. Currently the agent makes use of a simple colour analysis mechanism. However, section 11.4.1 highlights the limitations of this method.

Robo-CAMAL uses the RGB value of an object to determine its colour. The colour of an object is highly dependent on the context in which it is viewed (Brainard, 1999). For example, due to lighting effects the *blueball* is often in shadow and produces a colour profile similar to that of the *blackrobot*. This often leads robo-CAMAL to misclassify the *blueball* as a *blackrobot*. This means that it is important to recognise the possible variations in an object's brightness (Gilchrist et al., 1999).

Robo-CAMAL currently has no way to modify the colour profile of an object in response to the lighting conditions. This shows that one of the major drawbacks of the architecture is the rigidity of the perceptual domain model. If robo-CAMAL could make use of a number of different object profiles to identify an object, it could learn the most appropriate one for a given environment in much the same way as it learns the correct macro-behaviour.

One method to achieve this would be to hard code multiple colour profiles for each lighting condition. This, however, would prove impractical as there are so many different possible lighting conditions under which an object could be viewed. A second method would be the use of a perspective camera and the use of shape to augment colour. However, this would require additional computational processing and possibly increase the vision system lag.

An alternative method would be to allow robo-CAMAL to learn how to differentiate between different colours for itself. For example, the use of a neural network could allow robo-CAMAL to identify colours under a number of different lighting conditions.

A neural network designed to identify colour could work as follows. The neural net has three input neurons, each relating to one of the possible RGB values of the colour. The neural net makes use of a number of output neurons whose outputs can be combined to produce a multi-dimensional vector. In between the input and output neurons there are a number of hidden neurons.

Once the neural net has been trained, it produces a number of different dimensional vectors representing each colour and lighting condition possible. This means that an object would produce a number of output vectors that can be grouped into an object set. Such a system has been shown to have a 97% success rate at recognising different colours (Stoksik et al., 1991).

The way this could be integrated into robo-CAMAL is as follows. The neural net would be trained on a number of objects under different lighting conditions. This training would produce a number of sets containing the output vector of the neural net. Each set would become part of the domain model.

When robo-CAMAL's attitude is set as free roam, the vision system would produce an output neural vector in response to an object. This vector could be compared with the object sets deduced during the neural net's training. This mechanism would also allow a more rigorous confidence value to be associated with the belief.

Currently robo-CAMAL only makes use of three confidence ratings with the logical preference of perceived > deduced > assumed. The use of fuzzy logic would be able to associate a confidence in the belief as a percentage. For example, the degree of matching between output vector A and fuzzy set B is 0.9, giving a 90% likelihood that the object in the field of vision is B.

In addition, the case where two sets overlap can be taken into consideration. The set for a *blueball* in poor lighting conditions would be very similar to the *blackrobot* set. The reactive level would then trigger two beliefs of high confidence, i.e. *found(blueball)* and *found(blackrobot)*. It could then be left to the deliberative level to decide which belief is the most likely based on previous beliefs, in addition to the two confidence levels.

Another aspect of this addition is that robo-CAMAL would be able to learn to identify new objects that have been introduced into its environment. Currently if robo-CAMAL encounters an object that does not conform to its predefined colour profiles, it simply classifies it as a generic object and moves on.

However, if robo-CAMAL had the ability to switch attitudes, when it encountered a new object it could switch to a learning attitude. The

agent could then run the neural training routine to produce a new set that represents the new object. This set could then be added to the domain model for future reference. From this point of view the agents perceptual systems have the ability to alter the agents attitudes i.e. if the agent perceives an unknown object, and the deliberative system knows it can learn using its perceptual system, it may switch attitudes from free roam to object learning. It also suggests that specific abilities of an agents perceptual system, e.g. the ability of a neural network to identify objects based on colour, can be used to drive cognitive systems that take advantage of those abilities, e.g. the ability to learn about and identify new objects. This means that the nature of an agents perceptual system can drive the design of its cognitive systems (Barsalou, 1999).

The ability to learn new object profiles would also require robo-CAMAL to generate brand new beliefs and goals. For example, if the agent found a new object, and added the appropriate object profile to its domain model, it would also need to generate new beliefs and goals relating to the object. The generation of such beliefs is already implemented within robo-CAMAL. The reactive level would simply register the same beliefs as before, with the new object as the focus of the belief. Similarly, if a new object symbol is encountered at the deliberative level, it would be trivial to generate new goals by simply taking existing ones, and making the new object symbol the focus. As robo-CAMAL already has the ability to generate new associations, learning how to interact with the new object should not prove too difficult for the agent.

This sub-section has shown that by providing a more detailed and variable domain model, the robo-CAMAL architecture would be significantly enhanced. However, adding such functionality would require far more effort on the part of an external controller to manage the

agent's attitudes. For example, every time the agent encountered a new object, the external controller would have to swap the free roam attitude to a learn object attitude, then a learn association attitude, then finally back to the free roam attitude. For this reason the next sub-section details a possible solution that could be used to manage robo-CAMAL's attitudes.

11.6.2 Meta-Cognitive Layer

Currently robo-CAMAL requires an external control source to manage its attitudes. That is, for robo-CAMAL to switch between a learning and free roam attitude, it must be taken off line, reconfigured, and restarted. It is clear that robo-CAMAL would be better equipped to adapt to its environment if it could manage its attitudes without an external influence. For this reason the final suggested improvement to the robo-CAMAL architecture is the addition of a meta-cognitive layer.

Meta-cognition can be defined as *knowledge that takes as its object, or regulates any aspect of, any cognitive endeavour* (Flavell, 1978). This statement details two types of process. The first is an agent's knowledge about its own cognitive resources, and how they relate to its environment. The second activity regards the agent's ability to self regulate its cognitive mechanisms in an ongoing attempt to solve problems (Baker and Brown, 1984).

This description states that a meta-cognitive layer should have knowledge about how its cognitive processes relate to its environment, and with this knowledge be able to manage and control such processes.

The full CAMAL architecture consists of a meta-cognitive component (Venkatamuni, 2008). However, this was not implemented as a part of the robo-CAMAL project. This section suggests how such a layer may be integrated, and function within the robo-CAMAL archi-

tecture.

The first point regards the agent's knowledge about its own cognitive processes, and their relationship with the environment. Any knowledge the agent has about its self would be instantiated as a domain model component. Any variation in the domain model would be managed by the meta-cognitive layer. This is where the agent's ability to manage its own cognitive processes becomes important.

The first and most obvious cognitive process that requires management is the agent's attitudes. One example could be to change the agent's goal set. If the environment was cluttered and the agent continually failed to avoid objects, it could swap its goal set from *avoid(object)* to *hit(object)*. This is the simplest mechanism to employ. A more complex task would be to enable the agent to switch between a free roam and a learning attitude.

In a free roam attitude robo-CAMAL has a set of goals and a number of known good associations. In a learning attitude the agent has a single goal and a confined belief set that reflects its environment. For robo-CAMAL to have the ability to switch between the two attitudes, several additional mechanisms and affect values must be introduced.

The first and most obvious mechanism is the ability to recognise when to switch attitudes. This could be done by monitoring the association values. If one falls below a given threshold, it could mean that the association is no longer valid.

However, an association's failure may not be due to the fact that it is no longer valid. It may also be that the focus of the association's goal is no longer present in the environment. Therefore, an association should be relearned if it continues to fail whilst beliefs about the object of its focus persist e.g. if the *found(redrobot)* belief is constantly formed over a period of time, and the association used to achieve the *hit(redrobot)* goal continues to fail.

It can also be seen from section 10.3 that the association values fluctuate a great deal within a five minute period. This could adversely affect the agent's ability to switch attitudes. With such fluctuations robo-CAMAL could be switching between its learning and free roam attitude very rapidly.

One solution would be to have two association values. This would mean a learning and a known good value. The learning association value would vary as currently implemented. The known good association value would show a much smaller variation due to the agent's success or failure. This would mean that known good associations would persist for a greater period of time. Only continued failure over time would trigger the learning attitude.

The second important aspect in switching to a learning attitude is recognising that the environment is appropriate to learn from. That is, the environment is consistent so that the agent's beliefs can be constrained at the reactive level. This involves better object recognition and has been described previously in this chapter.

At the deliberative level, the agent must recognise when the environment is unchanging. This can be done by monitoring the belief set over a period of time. If no beliefs about a new object are introduced, or beliefs about a current object do not disappear, over a prolonged period of time, then the environment can be considered consistent.

A point to note is the situation where the environment changes whilst the agent is in a learning attitude. In this situation the agent has two choices. It can either exit the learning attitude, or risk learning in a variable environment. This decision is a balance between the importance of the goal the agent is trying to learn to achieve, and the success rate of associations learned in a variable environment.

The difficulty here is enabling robo-CAMAL to determine the reliability of its own learning mechanism. This would require the agent

to monitor the length of time an association learned in a variable environment took to decay. If this decay was rapid, robo-CAMAL could deduce that learning in a variable environment showed poor results. An additional affect value would be required to detail the number of processing cycles a known good association took to decay.

A second use for a meta-cognitive layer would be to enable robo-CAMAL to learn how to plan. Section 11.3.1 mentioned that this is something the agent is currently unable to do. However, this is a deficiency with robo-CAMAL, not the CAMAL architecture.

The CAMAL architecture allows for intentions to be strung together in a single association. For example, if the ultimate end goal is to *hit(X)*, and the current belief is *lost(X)*, the associations

associations(lost(X), find(X), lookFor(X))
associations(found(X), follow(X), stayCloseTo(X))
associations(near(X), hit(X), runInTo(X))

can be combined to given

associations(lost(X), hit(X), lookFor(X) + stayCloseTo(X) + runInTo(X))

There is no reason this could not be implemented within robo-CAMAL. This would simply require the agent to string a set of intentions together, and use the learning mechanism to decide which intention set, or plan, is the most successful. All the meta-cognitive layer would have to do is ensure that each intention in the set is triggered in the correct order.

11.7 Summary

This chapter took the results presented in chapter 10, and discussed their implications for the robo-CAMAL architecture. It also used

these results to compare robo-CAMAL with a number of similar systems, and its overall relation to some of the fields within A.I. research.

It initially looked at robo-CAMAL's anchoring mechanism. It described how the domain model is used to control the anchoring mechanism. It also looked at the anchoring mechanism employed by other systems and highlighted their similarities. This included the anchoring of events as well as objects.

The chapter then went on to look at the learning results. It highlighted the importance of learning in order to achieve goals, and gave a basic overview of robo-CAMAL's learning mechanism. It then compared the agent's learning mechanism to a system that makes use of a similar algorithm. This comparison highlighted the effect of learning in a physical environment. It showed that the image processing required to interpret the environment can adversely affect an agent's ability to learn.

It also stated that transferring an agent trained in a simulated environment to a physical one has a detrimental affect on its performance. However, robo-CAMAL showed an excellent success rate for learning in a physical environment with an imperfect vision system.

Section 11.4 discussed how robo-CAMAL adapts to its environment. It also showed the effect an inadequate perceptual domain model has on the agent's ability to adapt. It noted that robo-CAMAL adapts correctly to an environment. However, as the agent will sometimes form false beliefs about its environment, due to deficiencies in the domain model, it will adapt to the wrong environment.

The next section went on to discuss the nature of motivation in robo-CAMAL. It stated that motivation in robo-CAMAL is grounded through its interaction with the environment. It also showed, by using the indices of motivation, that motivation in robo-CAMAL can be considered as deliberative.

The chapter finally went on to suggest various improvements to robo-CAMAL. The first suggestion involved improving the agent's domain model. This included a better colour detection model using a neural network or the extending of the visual feature set beyond colour (e.g. shape)., and enabling the domain model to vary. The second suggestion looked at the inclusion of a meta-cognitive layer in the architecture. This layer would manage the variable domain model, and the agent's attitudes.

Chapter 12

Summary

12.1 Introduction

This thesis set out to demonstrate a cognitive agent that can do two things.

- Learn how to achieve pre-determined goals given no understanding of the consequences of its actions.
- Adapt its goals to reflect changes in a variable environment.

This was done in order to investigate the anchoring problem. This is the problem of linking perceptual data about objects and events, to symbolic representations of those objects and events. For an agent to achieve the goals of the thesis, it must have the ability to anchor. To this end, the robo-CAMAL architecture was developed.

This thesis can be divided up into three parts.

- The previous research that influenced the design of the robo-CAMAL architecture.
- The design of the robo-CAMAL architecture.
- The performance of the robo-CAMAL architecture.

This final chapter will summarise the three elements of this thesis, and draw the final conclusions about the architecture developed here.

12.2 Background Material

The robo-CAMAL architecture was designed from the reductionalist view point, and the theory of mind of as a control system. This means that the robo-CAMAL architecture can be decomposed into separate functional components, and was designed to control the actions of a mobile robot.

The architecture makes use of a number of different systems that have been developed over the years. At a more general level, robo-CAMAL makes use of a hybrid, reactive/deliberative architecture. It uses this set up in order to make use of a reactive system's ability to respond and act quickly in a rapidly changing environment. It can also use the deliberative system's ability to reason about events within its environment, and solve problems.

At a more specific level, robo-CAMAL's design was based on the CogAff architecture. This makes use of a three row, three column structure including reactive, deliberative and meta-cognitive rows, with a perceptual, cognitive and action execution columns.

In order to decide what to do next, robo-CAMAL makes use of the belief desire intention (BDI) architecture found in the CRIBB model. In addition, robo-CAMAL makes use of the distributed model of affect found in a-CRIBB to aid its decision making process.

The driving force behind robo-CAMAL's actions are its motivational constructs. It makes use of a motivational blackboard to prioritise its goals. The agent's motivations reside on the blackboard, along with their associated affect values. The affect values are used to order the agents goals in terms of their priority. These affect values are then modified by various knowledge sources, and are then used within the BDI schema to direct the agent's next action.

12.3 The Robo-CAMAL Architecture

The robo-CAMAL architecture consists of a number of components that work together to produce goal directed behaviour. A basic cycle operates as follows. The perceptual system observes the environment. If an event is perceived, then an event message is constructed and passed to the deliberative level. The deliberative level constructs a number of beliefs based on the received control message. The affect values are then modified based on the new information contained within the new beliefs. The BDI schema then chooses which action to perform next based on the affect values attached to the agent's beliefs and its motivators. A control message is then passed back to the reactive level that indicates which reactive architecture to activate. The reactive component then activates the relevant architecture, which in turn alters the environment. The change in the environment is registered by the perceptual system which constructs a new event message, and the cycle continues.

There are a number of components that allow the agent to perform these tasks. One is the domain model. This model is used to instantiate all the domain specific properties the agent has. This includes such things as the possible beliefs the agent can hold, the goals it can have, or the actions it can perform. It is the domain model that controls the anchoring process. The control messages are constructed based on rules defined by the domain model. The domain model also defined the various trigger thresholds. If an affect value crosses a trigger threshold, a control process is altered to produce a different behaviour. This can either change the agent's behaviour directly as in the case of a reactive system, or indirectly as in the alteration of a specific goal's affect values. In essence, the domain model is used to enable the architecture's generic components to operate in a specific

environment.

The driving force behind robo-CAMAL's motivation is its motivational constructs. Motivational control states come in a number of forms. Instincts and drives are found at the reactive level. These are pre-programmed responses to sensory input. For example the reactive behaviour

IF $(rightSensorValue < 50mm)$ **THEN** $(turn\ left)$

Goals come in two types, quantitative and qualitative. Quantitative goals are present at the reactive level and refer to groupings of a number of reactive systems that achieve a specific task. For example, the avoid object on the right alone is a drive, where as the combination of a number of avoid drives produces an *avoid(objects)* quantitative goal.

Qualitative goals are present in robo-CAMAL at the deliberative level and take the form *hit(blueball)*. However it is more appropriate to think of these goals as the end states associated with desires.

Desires within robo-CAMAL describe the specific goal, the belief required for the goals success, and the desires importance. Desires are present at the deliberative level within robo-CAMAL, and take the form *goal(Desire, Success Condition, Goal Importance, Threat Value)*.

Intentions are strategies and plans that are used to achieve desires. Intentions are found within robo-CAMAL at the deliberative level and take the form of predicates detailing the various possible reactive architectures.

Attitudes are pre-dispositions to respond in certain ways to certain perceptual or internal triggers. Attitudes are present within robo-CAMAL, but are pre-programmed prior to run time. An attitude in robo-CAMAL refers to the pre-defined goal set. Different attitudes or goal sets must be changed by the user off-line.

The component at the centre of the BDI schema is the association. An association consists of a belief, a desire, an intention, and an affect value that represents the likelihood that the intention will achieve its desire based on the belief.

The BDI schema initially chooses all the associations that contain the agent's current beliefs. It then chooses a single association based on the affect model. The intention is then extracted from this association, and used to construct the relevant control message sent to the reactive level.

Associations allow robo-CAMAL to learn how to achieve its goals and adapt in a variable environment. When learning how to achieve its goals, robo-CAMAL selects an association and extracts the intention. If the intention successfully achieves it desire, the association value is increased. This means that successful associations will develop higher association values over time.

The final association chosen is the one holding the relevant belief, and the highest association value. This means that robo-CAMAL's learning mechanism is opportunistic, and continually looks for the quick win that will achieve its goal immediately. This is a form of reinforcement learning in which the agent determines the success of its actions based on the feedback it receives from its environment. In this scenario the agent has a single goal in an unchanging environment.

Robo-CAMAL adapts in much the same way as it uses the learning mechanism. The main differences are:

- The agent has more than one goal.
- The agent's environment is variable.
- The agent has gone through the learning process to find the known good associations relating to its goals.

In this situation the importance value of a specific goal is taken into consideration when choosing the relevant association.

The affect values vary over time, in response to events in the environment, and on previous performance. The affect values therefore reflect changes in the agent's environment and its internal state. The associations are chosen based on the state of the affect model at that time. This allows robo-CAMAL to choose its goals based on changes in the environment and its internal state, and on its previous experience. It is the interaction of the BDI schema, the distributed model of affect, the association model, and the motivational control mechanisms that allow robo-CAMAL to adapt within its environment. The domain model is used to enable robo-CAMAL to operate in a specific environment, and is the only mechanism in the architecture that is not generic.

12.4 Experiments and Results

The experiments performed fall into three main categories. Control, learning, and adaptation experiments. The control experiments showed that the robo-CAMAL architecture performed as expected. They also produced control data with which to compare the learning experiments.

Two experiments were designed to analyse robo-CAMAL's learning mechanism. The first looked at the agent's ability to learn general behaviours. For this experiment the agent was pre-programmed with the correct environmental beliefs, and instantiated with a single goal. The agent then had to generate a number of associations and learn which one was most likely to achieve its goal.

Robo-CAMAL learns this through trial and error. Initially it tries the intention contained within a particular association. If this inten-

tion succeeds, then the association value is increased. If it fails then the association value is decreased. This means that over time, associations with the most appropriate intention will develop the highest association values.

The second learning experiment investigated robo-CAMAL's ability to optimise its performance. For example, can robo-CAMAL learn the best intention to achieve a specific goal? The agent is initially preprogrammed with the correct environmental beliefs, and instantiated with a single goal. It also has a number of correct associations preprogrammed. The agent then tries to learn which association shows the best performance.

However, the results for this experiment showed that robo-CAMAL did not have the ability to learn the optimal intention with which to achieve its goal. This was due to the opportunistic nature of the agent's learning mechanism. Robo-CAMAL initially attempts the first association in the list. If this is successful, then the agent continues to use this association until it fails. This means that robo-CAMAL will favour the first successful association.

The adaptation experiments were designed to investigate robo-CAMAL's ability to modify its goal in response to changes in its environment. For this experiment the agent was instantiated with a number of goals, and the correct associations. The agent was then allowed to roam within its environment and choose its own goals. The environment was periodically altered, with objects being removed and added at various intervals.

The adaptation experiments showed mixed results. They indicated that the deliberative component adapted in a consistent way with changes in its beliefs. However, if the agent's beliefs were incorrect, it attempted to adapt to the perceived environment, and not the actual one. This highlights two things,

- Moving from a controlled to a free environment has a detrimental effect on an agent's performance.
- An agent's anchoring mechanism is crucial in allowing an agent to adapt in a variable environment.

In the case of robo-CAMAL, although the anchoring mechanism was sufficient to allow it to learn in a controlled environment, it was insufficient to provide a set of beliefs that consistently mirrored a variable environment.

12.5 Conclusion

This thesis set out to demonstrate a cognitive agent that could learn, and adapt in a physical environment. This was done to investigate issues surrounding the anchoring problem. If a cognitive agent that makes use of symbolic processing can learn and adapt in a physical environment, then it must also have the ability to link sensory data to the symbols with which it reasons.

To this end the robo-CAMAL architecture was developed. Robo-CAMAL demonstrated the ability to learn which actions to use in order to achieve its goal. However, it was unable to fine tune its performance, and learn the best action given a number of actions that achieve the same thing. This was due to the opportunistic nature of the learning mechanism implemented.

The learning experiments demonstrate that robo-CAMAL has the ability to anchor events and object to pre-defined symbols. It uses the domain model to recognise an object such as a *blueball*, and an event such as *hit(blueball)*.

The adaptation experiment showed that robo-CAMAL can adapt to its perceived environment, but that the domain model was insufficient to consistently build the correct beliefs.

The agent's performance can be improved in two main ways. The first involves a more sophisticated domain model, for example, the use of a neural network to learn about new objects in its environment.

The second improvement involves the addition of a meta-cognitive layer to manage the agent's goals and attitudes, as well as a variable domain model, for example changing the agent's attitude from a free roam to a learning mode.

In conclusion, robo-CAMAL has demonstrated an ability to anchor symbols to perceptual data with the use of a domain model. This anchoring mechanism performs very well in a controlled environment, but struggles to cope in an uncontrolled environment. This failure is due to a lack of sophistication in the domain model. This highlights just how important the anchoring process is to a deliberative agent if it is to function in its environment

References

Activemedia. *Amigobot technical manual*. Activemedia Robotics, 2000.

Activemedia. *ARIA reference manual*. Activemedia Robotics, 2003.

G Adorni, S Cagnoni, M Carletti, and M Mordonini. Omnidirectional vision algorithms in robotics. In *Atti dell'VIII Convegno AI*IA*, 2002.

P Agre and D Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, 1987.

E Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.

R Arkin. Integrating behavioural, perceptual and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

R Arkin. *Behaviour-Based Robotics*. MIT Press, 1998.

M Asada, S Noda, S Tawaratsumida, and K Hosoda. Perposive behavior acquisition on a real robot by a vision-based reinforcement learning. In *Proceedings of MLC-COLT (Machine Learning Conference and Computer Learning Theory) Workshop on Robot Learning*, pages 1–9, 1994.

L Baker and A Brown. Metacognitive skills and reading. In P Pearson, editor, *Handbook of Reading Research*. Lawrence Erlbaum Associates, 1984.

L Barsalou. Perceptual symbol systems. *Behavioral and Brain Sciences*, 22:577–660, 1999.

K Bartsch and H Wellman. Young children's attribution of action to beliefs and desires. *Child Development*, 60:946–964, 1989.

R Beck. *Motivation: Theories and Principles*. Pearson Education, 4 edition, 2000.

G Bourgne. Affect-based multi agent architecture (for a 5-aside football simulation). Master's thesis, School of Computer Science, University of Hull, 2003.

B Brainard. Colour vision. In R Wilson and F Keil, editors, *The MIT Encyclopaedia of the Cognitive Sciences*. MIT Press, 1999.

V Braitenberg. *Vehicles, Experiments in Synthetic Psychology*. MIT Press, 1984.

M Bratman. *Intentions, Plans and Practical Reason*. Harvord University Press, 1987.

R Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, 1986.

R Brooks. Intelligence without reason. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 1, pages 569–595, 1991.

R Brooks. From earwigs to humans. *Robotics and Autonomous Systems*, 20:291–304, 1997.

B Buchanan and E Shortcliff. *Rule based expert systems: The. MYCIN experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

N Carlson, G Martin, and W Buskist. *Psychology.* Allyn and Bacon, 2003.

A Chella, M Frixione, and S Gaglio. Anchoring symbols to conceptual spaces: the case of dynamic scenarios. *Robotics and Autonomous Systems*, 43(2):175–188, 2003.

W Clancey. *Situated Cognition.* Cambridge University Press, 1997.

A Clark. Embodiement and the philosophy of mind. In A O'Hear, editor, *Current Issues in Philosophy of Mind: Royal Institute of Philosophy Supplement 43.* Cambridge University Press, 1998.

S Coradeschi and A Saffiotti. Anchoring symbols to vision data by fuzzy logic. In A Hunter and S Parsons, editors, *Qualitative and Quantitative Approaches to Reasoning with Uncertainty.* Springer, 1999.

S Coradeschi and A Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43:85–96, 2003.

D Corkill. Blackboard systems. *AI Expert*, 6(9):40–47, 1991.

I Craig. *The Cassandra architecture: distributed control in a blackboard system.* Halsted Press, 1989.

D Davis. Reactive and motivational agents. In J Muller, M Wooldridge, and N Jennings, editors, *Intelligent Agents III: Agent Theories, Architectures and Languages.* Springer-Verlag, 1996.

D Davis. Control states and complete agent architectures. *Computational Intelligence*, 17(3):621–650, 2001.

D Davis. Computational architectures for intelligence and motivation. In *The 17th IEEE International Symposium on Intelligent Control*, 2002.

D Davis. Why do anything? emotion, affect and the fitness function underlying behaviour and thought. In *Proceedings of the AISB 2004 Symposium on Affective Computing*, 2004.

D Davis. Linking perception and action through motivation and affect. *Journal of Experimental and Theoretical Artificial Intelligence*, 2007.

D Dennett. Cognitive wheels: The frame problem of a.i. In C Hookway, editor, *Minds, Machines and Evolution: Philosophical Studies*. Cambridge University Press, 1984a.

D Dennett. *Elbow Room*. MIT Press, 1984b.

D Dennett. Cognitive wheels. In Z Pylyshyn, editor, *Robot's Dilemma: The Frame Problem in Artificial Intelligence*. Greenwood Publishing, 1987.

A Epstein. Instinct and motivation as explanations for complex behavior. In D Pfaff, editor, *The Physiological Mechanisms of Motivation*. Springer, 1982.

M. Fiala and A. Basu. Robot navigation using panoramic tracking. *Pattern Recognition*, 37(11):2195–2215, 2004.

M Fishbein and I Ajzen. Attitudes towards objects as predictors of singla and multiple behavioral criteria. *Psychology Review*, 81:59–74, 1974.

J Flavell. Metacognitive development. In J Scandura and C Brainerd, editors, *Structural/Process Theories of Complex Human Behavior*. Sijthoff and Noordhoff, 1978.

J Fodor. Methodological solipsism considered as a research strategy in cognitive psychology. *Behavioral and Brain Sciences*, 3:63–109, 1980.

J Fodor. *Modularity of Mind: An Essay on Faculty Psychology*. MIT Press, 1983.

J Fodor. Precis of the modularity of mind. *Behavioral and Brain Sciences*, 8:1–42, 1985.

S Franklin. *Artificial Minds*. MIT Press, 1995.

J Gaspar, E Grossmann, and J Santos-Victor. Interactive reconstruction from an omnidirectional image. In *In Proceedings of the 9th International Symposium on Intelligent Robotic Systems*, 2001.

M Genesereth and N Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

M Georgeff, B Pell, M Pollack, M Tambe, and M Wooldridge. The belief-desire-intention model of agency. In J Müller, M Singh, and A Rao, editors, *Intelligent Agents V: Agent Theories, Architectures and Languages*, pages 1–10. Springer-Verlag, 1999.

M Georgeff and A Rao. The semantics of intention maintenance for rational agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 1, pages 704–710. Morgan Kaufmann, 1995.

J Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum, 1986.

A Gilchrist, C Kossyfidis, F Bonato, T Agostini, J Cataliotti, X Li, B Spehar, V Annan, and E Economou. An anchoring theory of lightness perception. *Pychological Review*, 106(4):795–834, 1999.

S Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

J Haugeland. An overview of the frame problem. In Z Pylyshyn, editor, *Robot's Dilemma: The Frame Problem in Artificial Intelligence*. Greenwood Publishing, 1987.

B Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(12):329–365, 1995.

K Holyoak. Introduction:psychology. In R Wilson and F Keil, editors, *The MIT Encyclopaedia of the Cognitive Sciences*. MIT Press, 1999.

D. Hougen, J. Fischer, M. Gini, and J. Slagle. Fast connectionist learning for trailer backing using a real robot. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1917–1922, 1996.

C Hull. *Principles of Behavior: An Introduction to Behavior Theory*. Oxford University Press, 1943.

R Kowalski and M Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

G Lakoff and M Johnson. *Philosophy in the flesh: The embodied mind and its challenge to western thought*. Basic Books, 1999.

D Langer, J Rosenblattb, and M Heberte. A behavior-based system for off-road navigation. *IEEE Journal of Robotics and Automation*, 10(6):776–782, 1994.

J Larsson and B Hayes-Roth. Guardian:an intelligent autonomous agent for medical monitoring and diagnosis. *IEEE Intelligent Systems*, 13(1):58–64, 1998.

S Lewis. *Computational Models of Emotion and Effect*. PhD thesis, School of Computer Science, University of Hull, 2004.

D Lloyd. *Simple Minds*. MIT Press, 1989.

E Locke. Motivation through conscious goal setting. *Applied and Preventive Psychology*, 5:117–124, 1996.

S Mahadevan and J Connell. Automatic programming of behavior-based robots using reinforcement learning. In *National Conference on Artificial Intelligence*, pages 768–773, 1991.

L. Marques, U. Nunes, and A. Almeida. Olfaction-based mobile robot navigation. *Thin Solid Films*, 418(1):51–58, 2002.

R. Martínez and D. Borrajo. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems*, 29(1):47–78, 2000.

H Maturana and F Varela. *The Tree of Knowledge*. Shambhala, 1992.

B McBride. A timeline of panoramic cameras. on line, 2005. URL `http://www.panoramicphoto.com/timeline.htm`.

J McCarthy and P Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B Meltzer and D Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, 1969.

D McFarland. What it means for robot behaviour to be adaptive. In J Meyer and S Wilson, editors, *From animals to animats: Proceedings of the Third International Conference on Simulation of Adaptive Behaviour*. MIT Press, 1991.

D McFarland. *Animal Behaviour*. Addison Wesley Longman Ltd, 1993.

D McFarland and T Bösser. *Intelligent Behavior in Animals and Robots*. MIT Press, 1993.

M Minsky. *The Society of Mind*. Simon and Schuster, 1986.

T Mitchell. *Machine Learning*. McGraw-Hill, 1997.

R Murphy, K Hughes, A Marzilli, and E Noll. Integrating explicit planning with reactive control of mobile robots using trulla. *Robotics and Autonomous Systems*, 27(4):225–245, 1999.

A Newell. Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R Nickerson, editor, *Attention and Performance VIII*. Lawrence Eribaum Asscciates, 1980.

A Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

A Newell and H Simon. *Human Problem Solving*. Prentice-Hall, 1972.

A Newell and H Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

N Nilsson. Shakey the robot. Technical Report 323, Stanford Research Institute AI Centre, 1984.

A Norman. Twelve issues for cognitive science. *Cognitive Science*, 4: 1–32, 1980.

H Nunes. Investigation of motivation in agents using the simulation of 5-aside football. Master's thesis, School of Computer Science, University of Hull, 2001.

J Okamoto and V Grassi. Visual servo control of a mobile robot using omnidirectional vision. In *Proceedings of the 8th Mechatronics Forum International Conference*, pages 413–422, 2002.

R Pfeifer. Cognition - perspectives from autonomous agents. *Robotics and Autonomous Systems*, 15:47–70, 1995.

R Pfeifer and C Scheier. *Understanding Intelligence*. MIT Press, 1999.

R Picard. *Affective Computing*. MIT Press, 1998.

E Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1934.

Z Pylyshyn. *Robot's Dilemma: The Frame Problem in Artificial Intelligence*. Greenwood Publishing, 1987.

W Ramsey, S Stich, and J Garan. Connectionism, eliminativism, and the future of folk psychology. In D Cole, H Fetzer, and T Rankin, editors, *Philosophy, Mind, and Cognitive Inquiry*. Kluwer Academic Publishers, 1990.

P Rosenbloom, J Lairdand, and A Newell. *The Soar Papers: Research on Integrated Intelligence*. MIT Press, 1993.

S Rosenschein and L Kaelbling. A situated view of representation and control. In P Agre and S Rosenschein, editors, *Computational Theories of Interaction and Agency*. MIT Press, 1996.

D Rumelhart and J McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

S Russell and P Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

T Savage. The grounding of motivation in artificial animals: Indices of motivational behavior. *Cognitive Systems Research*, 4:23–55, 2003.

J Searle. Minds, brains and programs. *Behavioral and Brain Sciences*, 3:417–457, 1980.

M Shanahan. *Solving the Frame Problem*. MIT Press, 1997.

M Shanahan. The event calculus explained. In M Wooldridge and M Veloso, editors, *Artificial Intelligence Today*. Springer-Verlag, 1999.

M Shanahan. Reinventing shakey. In J Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.

M Shanahan and M Witkowski. High level robot control through logic. In C Castelfranchi and Y Lesperance, editors, *Intelligent Agents VII: Agent Theories, Architectures and Languages*. Springer-Verlag, 2001.

N Sharkey and J Heemskerk. The neural mind and the robot. In A Brown, editor, *Neural Network Perspectives on Cognition and Adaptive Robotics*. IOP Press, 1997.

N Sharkey and T Ziemke. Mechanistic verses phenomenal enbodiment: Can robot embodiment lead to strong ai? *Cognitive Systems Research*, 2:251–262, 2001.

H Simon. *The Sciences of the Artificial*. MIT Press, third edition, 1996.

H Simon. Production system. In R Wilson and F Keil, editors, *The MIT Encyclopaedia of the Cognitive Sciences*. MIT Press, 1999.

A Sloman. Motives, mechanisms, and emotions. In M Boden, editor, *The Philosophy or Artificial Intelligence*. Oxford University Press, 1990.

A Sloman. The mind as a control system. In C Hookway and D Peterson, editors, *Philosophy and Cognitive Science*. Cambridge University Press, 1993.

A Sloman. Cognitive architecture. In R Wilson and F Keil, editors, *The MIT Encyclopaedia of the Cognitive Sciences*. MIT Press, 1999.

A Sloman. Varieties of affect and the cogaff architecture schema. In *Proceedings Symposium on Emotion, Cognition, and Affective Computing AISB'01 Convention*, pages 39–48, 2001.

A Sloman, L Beaudoin, and I Wright. Computational modelling of motive-management processes. In N Frijda, editor, *Proceedings of the Conference of the International Society for Research in Emotions*. ISRE Publications, 1994.

B Smith. Situatedness/embeddedness. In R Wilson and F Keil, editors, *The MIT Encyclopaedia of the Cognitive Sciences*. MIT Press, 1999.

T Sogo, H Ishiguro, and M Treivedi. Real-time target localization and tracking by n-ocular stereo. In *Proceedings of the 1st International IEEE Workshop on Omnidirectional Vision*, 2000.

M Sonka, V Hlavac, and R Boyle. *Image Processing, Analysis and Machine Vision*. Chapman and Hall, 1993.

E Sontag. *Mathematical control theory: Deterministic finite dimensional systems*. Springer, New York, second edition, 1998.

C. Stachniss and W. Burgard. Mobile robot mapping and localization in non-static environments. In *In Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 1324–1329, 2005.

L Steels and J Baillie. Shared grounding of event descriptions by autonomous robots. *Robotics and Autonomous Systems*, 43(2):163–173, 2003.

A Stenz. The focussed d* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

N Stillings, S Weisler, C Chase, M Feinstein, J Garfield, and E Rissland. *Cognitive Science: An Introduction*. MIT Press, 1995.

M Stoksik, T Nguyen, and M Czernkowski. A neural net based colour recognition system. In *The Second Interational Conference on Artificial Neural Networks*, pages 86–90, 1991.

A. Stoytchev and R. Arkin. Combining deliberation, reactivity, and motivation in the context of a behavior-based robot architecture. In *In Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 290–295, 2001.

A. Stoytchev and R. Arkin. Incorporating motivation in a hybrid robot architecture. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 8(3):100–105, 2004.

P Sturm. A method for 3d reconstruction of piecewise planar objects from single panoramic images. In *In Proceedings of the 1st International IEEE Workshop on Omnidirectional Vision*, pages 119–126, 2000.

R Sutton and A Barto. *Reinforcement Learning*. MIT Press, 1998.

A Tate. Planning. In R Wilson and F Keil, editors, *The MIT Encyclopaedia of the Cognitive Sciences*. MIT Press, 1999.

E Thelen, G Schoner, C Scheier, and L Smith. The dynamics of embodiment: A field theory of infant perservative reaching. *Behavioral and Brain Sciences*, 24:1–86, 2001.

N Tinbergen. *The Study of Instinct*. Clarendon Press, 1951.

M Toda. *Man, Robot, and Society*. Martinus Nijhoff Publishing, The Hague, 1982.

W Uttal. *The New Phrenology: The Limits of Localizing Cognitive Processes in the Brain*. MIT Press, 2003.

V Venkatamuni. *A Society of Mind Approach to Cognition and Metacognition in a Cognitive Architecture*. PhD thesis, School of Computer Science, University of Hull, 2008.

S Wahl and H Spada. Children's reasoning about intentions, beliefs and behaviour. *Cognitive Science Quarterly*, 1:5–34, 2000.

C Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

D Westen. *Pyychology: Mind, Brain, and Culture*. John Wiley and Sons, Inc, 1996.

L Wittgenstein. *Philosophical Investigations*. Blackwell Publishing, 1953.

D. Wolf and S. Sukhatme. Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, 19(1): 53–65, 2005.

T Zimmerman and S Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2):73–96, 2003.

Appendix A

The Motivational Blackboard

A.1 Introduction

This appendix will cover, in detail, the motivational blackboard system used by Robo-CAMAL. Though this topic has been discussed in chapter 9, it was covered in general terms. No real detail was given. This appendix will cover in detail what the three components of the blackboard system (the blackboard, the control component, and the knowledge sources) contain. This represents the whole deliberative component which has been written in Prolog. It will also highlight, in bold script, any component that is considered part of the domain model. Components of the affect model will be highlighted in red.

A.2 The Blackboard

---

agent_time(X).

Details the number of processing cycles, X, the deliberative component has executed.

---

environment_descriptor_list([sparse, cluttered, dynamic, static]).

Details the possible states that the environment can be in.

---

object_list([object, ball, blackrobot, redrobot]).

Details the possible types of objects to be found in the environment.

---

object_predicate_list([lost, found, near, hit]).

Details the possible states that environmental objects can be in.

---

negate(environment(sparse), environment(cluttered)).
negate(environment(cluttered), environment(sparse)).
negate(environment(static), environment(moving)).
negate(environment(moving), environment(static)).
negate(found(X), lost(X)).
negate(lost(X), found(X)).
negate(near(X), lost(X)).
negate(hit(X), lost(X)).
negate(not(hit(X)), hit(X)).
negate(hit(X), not(hit(X))).

Details which beliefs conflict with one and other.

---

belief_preference(perception, assumption).
belief_preference(perception, deduction).
belief_preference(deduction, assumption).

Determines which source is preferred for two conflicting beliefs. That is X is preferred over Y for *belief_preference(X, Y).* This is a component of the affect model.

---

domain_synonym(found, near).

Details the belief found can be deduced if the belief near is present.

---

clutter_threshold(2).

Determines the number of objects needed in the environment for it to be considered cluttered.

---

*belief(**BeliefDescriptor**, BeliefSource, Agent_time).*

Represents the agent's beliefs. ***BeliefDescriptor*** details the semantic context of the belief, and it can take one of the following forms,

- *environment(**environment_descriptor_list**).*
- ***object_list(object_predicate_list).***
- *failed(**Intention**).*

Here ***Intention*** is one of the possible reactive architectures. *BeliefSource* detailes how the belief was formed, and is an affect value. It can take one of the following values, perception, deduction, assumption. *Agent_time* details at which deliberative processing cycle the belief was formed.

---

intention_list([track, hit, avoid, find]).

Details the possible actions that can be performed.

---

desire_list([avoid(collisions), hit(object), hit(ball), hit(blackrobot), hit(redrobot), avoid(object), avoid(ball), avoid(blackrobot), avoid(redrobot), find(object), find(ball), find(blackrobot), find(redrobot), track(object), track(ball), track(blackrobot), track(redrobot)]).

Details all the possible desires with regards to the ***intention_list*** and ***object_list***.

---

goal_minmax(0.1, 0.9).

Represents the minimum and maximum values the Goal_importance value can take. It is defined by the affect model.

---

goal_threat_max(10).

Represents the maximum number of times a goal can consecutively fail before the Goal_importance is reduced. It is defined by the affect model.

---

*goal(**Desire**, **Success_condition**, Goal_importance, Threat_value).*

Represents the agent's goals.***Desire*** states the desired end state of the goal, and takes the form ***desire_list***. ***The Success_condition*** is the belief required for the ***Desire*** to be true. It takes the form ***BeliefDescriptor***. *Goal_importance* is an affect value, and reflects how important the goal is to the agent. It can vary between 0.1 and 0.9. *Threat_value* represents the number of times the goal has consecutively failed. Due to the domain specific connection between the ***Desire*** and ***Success_condition***, all goals are pre-programmed.

---

architecture(sonar, priority, avoid).
architecture(sonar, aggregate, avoid).
architecture(sonar, winner, avoid).
architecture(sonar, suppression, avoid).
architecture(vision, priority, avoid).
architecture(vision, aggregate, avoid).
architecture(vision, winner, avoid).
architecture(vision, suppression, avoid).
architecture(both, priority, avoid).
architecture(both, aggregate, avoid).
architecture(both, winner, avoid).
architecture(both, suppression, avoid).
architecture(sonar, priority, find).
architecture(sonar, aggregate, find).
architecture(sonar, winner, find).
architecture(sonar, suppression, find).
architecture(vision, priority, find).
architecture(vision, aggregate, find).
architecture(vision, winner, find).
architecture(vision, suppression, find).
architecture(both, priority, find).
architecture(both, aggregate, find).
architecture(both, winner, find).
architecture(both, suppression, find).
architecture(sonar, priority, track).
architecture(sonar, aggregate, track).
architecture(sonar, winner, track).
architecture(sonar, suppression, track).
architecture(vision, priority, track).

architecture(vision, aggregate, track).
architecture(vision, winner, track).
architecture(vision, suppression, track).
architecture(both, priority, track).
architecture(both, aggregate, track).
architecture(both, winner, track).
architecture(both, suppression, track).
architecture(sonar, priority, hit).
architecture(sonar, aggregate, hit).
architecture(sonar, winner, hit).
architecture(sonar, suppression, hit).
architecture(vision, priority, hit).
architecture(vision, aggregate, hit).
architecture(vision, winner, hit).
architecture(vision, suppression, hit).
architecture(both, priority, hit).
architecture(both, aggregate, hit).
architecture(both, winner, hit).
architecture(both, suppression, hit).

Details the possible reactive architectures that can be triggered.

---

*association(**BeliefDescriptor**, **Desire**, **Architecture**, Association_value).*

Used to keep track of the possible belief, desire, intention combinations used within the BDI schema. Given the ***BeliefDescriptor***, the *Association_value* gives the likelihood that the specific reactive architecture, ***Architecture***, will achieve the **Desire**.

---

motivator(Goal, Association, ***Reactive_cycles****, Motivator_value).*

The motivator is the result of the execution of the various knowledge sources. It contains the goal the agent wishes to achieve, the association the agent has chosen to achieve that goal, the number of cycles the reactive component should run for, and the motivators importance. The *Motivator_value* is redundant, but has been retained for any possible future research using the Robo-CAMAL model.

A.3 The Control Component

*call_deliberative(****Event****,* ***Object****,* ***Sensor****,* ***Method****,* ***Task****,* ***Cycles****,* ***ObjectOut****):-*

time_update,
*construct_belief(****Event****,* ***Object****, Updates),*
belief_update(Updates),
desire_update,
choose_motivator,
*intention_update(****Sensor****,* ***Method****,* ***Task****,* ***Cycles****,* ***ObjectOut****),!.*

The *call_deliberative* function is the entry point to Robo-CAMAL's deliberative component. It is called by the reactive component in much the same way as a micro-behaviour. The call is dependent on a set of rules instantiated at the reactive level (see section 6.7).

The *call_deliberative* function takes several arguments. The first two are provided by the reactive component. The ***Event*** argument

takes the form ***intention_list***. The ***Object*** argument takes the form ***object_list***.

Once the *call_deliberative* function has been triggered, the knowledge sources are called in the order shown. The *time_update* increments the *agent_time* by one. The *construct_belief* uses the ***Event*** and ***Object*** arguments, provided by the reactive component, to construct any new beliefs. These new beliefs are placed in the *Updates* argument. The *belief_update* uses the *Updates* argument to modify the agent's current belief set accordingly. The *desire_update* uses the new belief set to determine if it has achieved its current goal. It then updates the relevant goal and association affect values accordingly. The *choose_motivator* uses the current goal and belief set to choose the most appropriate goal and association for the agent at that time. It then updates the motivator accordingly.

Finally the *intention_update* extracts the relevant arguments from the new motivator. These are the arguments that determine the specific reactive architecture to trigger. The ***Sensor*** argument determines which sensor combination is to be used. It can take one of three values which are ***sonar***, ***vision***, or ***both***. The ***Method*** argument determines which micro-behaviour combination method should be used. It can take one of four values which are ***priority***, ***aggregate***, ***winner***, or ***suppression***. The ***Task*** argument details the specific macro-behaviour to triggerd. It is of the form ***intention_list***. The ***Cycles*** argument details the number of processing cycles the reactive component is to run for. The ***ObjectOut*** argument details the specific object of interest. It is used to configure the vision system and is of the form ***object_list***. The ***call_deliberative*** function then returns control to the reactive component with the relevant arguments updated.

A.4 The Knowledge Sources

The knowledge sources are the various functions called by the *call_deliberative* routine.

---

time_update.

Increments the *agent_time* by 1.

---

*construct_belief(**Event**, **Object**, Updates),*

Uses the ***Event*** and ***Object*** arguments to generate a list of new beliefs, *Updates*, based on the robots perceptual data. There are four possible scenarios.

- Scenario 1. The incoming ***Event*** is avoid, and the agent's goal was to avoid objects. In this case the *Updates* argument is set to *[[environment(**sparse**), deduction, agent_time]]*. The belief is that the environment is ***sparse*** as no objects were encountered. The belief is *deduced* as there could be objects in the environment, the agent simply has not found them.
- Scenario 2. The incoming ***Event*** is avoid, and the agent's goal was to perform an action on an object. In this case the *Updates* argument is set to *[[environment(**sparse**), deduction, agent_time], [failed(**Intention**), perception, agent_time]]*. The first belief is generated for the same reason as scenario 1. The second belief states that the specific reactive architecture, ***Intention***, failed to achieve its goal.
- Scenario 3. The incoming ***Event*** is an event, and the agent's goal is satisfied by that event on the appropriate

object, ***Object***. In this case the *Updates* argument is set to *[[Event(**Object**) ,perception, agent_time]]*. This time the specific event has been perceived by the agent.

- Scenario 4. The incoming ***Event*** is an event, and the agent's goal is not satisfied by that event on the appropriate object, ***Object***. In this case the *Updates* argument is set to *[[Event(**Object**) ,perception, agent_time], [failed(**Intention**), perception, agent_time]]*. Again the second belief states that the specific reactive architecture, ***Intention***, failed to achieve its goal.

---

belief_update(Updates),

Uses the new beliefs generated by the *construct_belief* knowledge source to update the agent's belief set. The *belief_update* knowledge source has several tasks.

- Remove beliefs about *hit(**Object**)* that are older than one deliberative process cycle.
- Remove beliefs about *failed(**Intention**)* that are older than one deliberative process cycle.
- If two beliefs have the same ***BeliefDescriptor***, then retain the most recent. If two beliefs have the same ***BeliefDescriptor***, and were formed at the same time, then use the *belief_preference(X, X)* to retain the belief with the preferred source.
- Use the ***negate(X, Y)*** predicate to determine if two beliefs conflict. If so then retain the most recent.
- Use the ***negate(X, Y)*** predicate to determine if two beliefs conflict. If they were formed at the same time, then

use the *belief_preference(X, Y)* to retain the belief with the preferred source.

- Remove beliefs older than 25 deliberative processing cycles.
- Use the ***domain_synonym(X, Y)*** to deduce any belief if subset belief present.
- Use the ***clutter_threshold(2)*** to determine if the environment is sparse or cluttered.
- Determine if the environment is static or dynamic using the domain model assumption that **if a red or black robot is present then the environment is dynamic.**

---

desire_update,

Uses the motivator to determine if the agent has achieved its goal. Using this information it updates the relevant association and goal affect values accordingly. There are two possible scenarios

- Scenario 1. The agent has achieved its goal. The *association_value* of the association used to achieve the goal is increased. The *goal_importance* value is set to 0.5. The *goal_threat* value is set to 0. Finally the motivator is removed.
- Scenario 2. The agent has failed its goal. The *association_value* of the association used to try and achieve the goal is reduced. The *goal_importance* value is incremented by 0.02. If the *goal_threat* value is set to 10,

set the *goal_threat* to 0 and the *goal_importance* to 0.1. Finally the motivator is removed.

---

choose_motivator,

Uses the various affect values to choose the goal the agent wishes to achieve, and the association with which to try and achieve the goal.

- Place the agent's current goal set in a list.
- Remove goals that have just been achieved from the goal list.
- Find all the associations that relate to the agents current goals and belief set. If no association exists then create new ones based on the goal and belief set.
- Pair up each goal with the relevant associations.
- Rank each association based on its *association_value*, the *goal_importance*, and the time its belief was formed at.
- Use the association with the highest *rank_value* to instantiate the new motivator. The number of reactive cycles is based on the domain model. **If the motivator goal is to *find(Object)*, then the cycle number is set to 100, else the cycle number is set to 50**.

---

*intention_update(**Sensor**, **Method**, **Task**, **Cycles**, **ObjectOut**),*

Used to extract the relevant reactive configuration to be triggered, from the *motivator*.

- ***Sensor***, ***Method***, and ***Task*** arguments extracted from the *association*, ***Intention*** component of the *motivator*. Represents the specific reactive architecture to trigger.
- ***Cycles*** argument extracted directly from the *motivator*. Represents how many processing cycles the reactive component should run for.
- ***ObjectOut*** argument extracted from the *goal*, ***Desire*** component of the *motivator*. Represents the agent's object of interest and is used to configure the vision system.

Appendix B

The Domain Model

B.1 Introduction

This appendix will describe the components that make up the domain model. The domain model consists of all the elements that describe some part of the agent's physical form or its environment. Table B.1 describes the domain element name, its description, the level within the architecture it can be found at, and the nature (i.e. whether the element relates to the agent's environment or form) of the element.

B.2 Domain Model Description

Name	**Description**	**Level**	**Nature**
Proximity Thresholds	Distance threshold within which an event has occurred or an action is triggered. Found in the micro-behaviours, the reactive rule set, and the vision processing system.	reactive	environment
Leading Edge Assumption	Assumption that any node chain found represents the leading edge of an object. This is based on the agents environment and the nature of the omnidirectional vision system.	reactive	environment/ form

Name	**Description**	**Level**	**Nature**
Colour Profiles	The RGB values that describe each of the three objects present in the environment.	reactive	environment
Reactive Rule Set	Set of rules that construct a statement based on the internal state of the reactive component. This statement is then passed to the deliberative component.	reactive	environment/ form
Reactive Processing Cycles	The maximum number of the reactive processing cycles the agent should run for before constructing a deliberative statement.	deliberative	environment
Belief Descriptor	The descriptor used do describe a possible belief i.e. *found*, *hit* etc.	deliberative	environment
Belief State	Possible states the Belief Descriptor can take i.e. *redrobot*, *sparse* etc.	deliberative	environment
Belief Persistence Threshold	The number of deliberative processing cycles a belief exists for before it is removed.	deliberative	environment
Belief Update Model	Describes which beliefs are mutually exclusive (i.e. *lost(redrobot)* and *found(redrobot)*), and which can be deduced (i.e. *found(blueball)* can be deduced from *hit(blueball)*).	deliberative	environment
Goal Descriptor	Specific goal description i.e. *hit*, *avoid* etc.	deliberative	environment
Goal Object	The focus of the Goal Descriptor i.e. *redrobot*, *object* etc.	deliberative	environment
Architecture Descriptor	One of the possible combinations of micro-behaviours/sensor system/arbitration method	deliberative	form

Table B.1: Domain model components.

Appendix C

Results

C.1 Introduction

This appendix will detail the results gathered during the experiments described in chapter 10.5. It will start with the first learning experiment which involved robo-CAMAL in a controlled environment attempting to achieve a single goal. Robo-CAMAL had to generate a number of associations and learn which ones provided the most likely to achieve its goal.

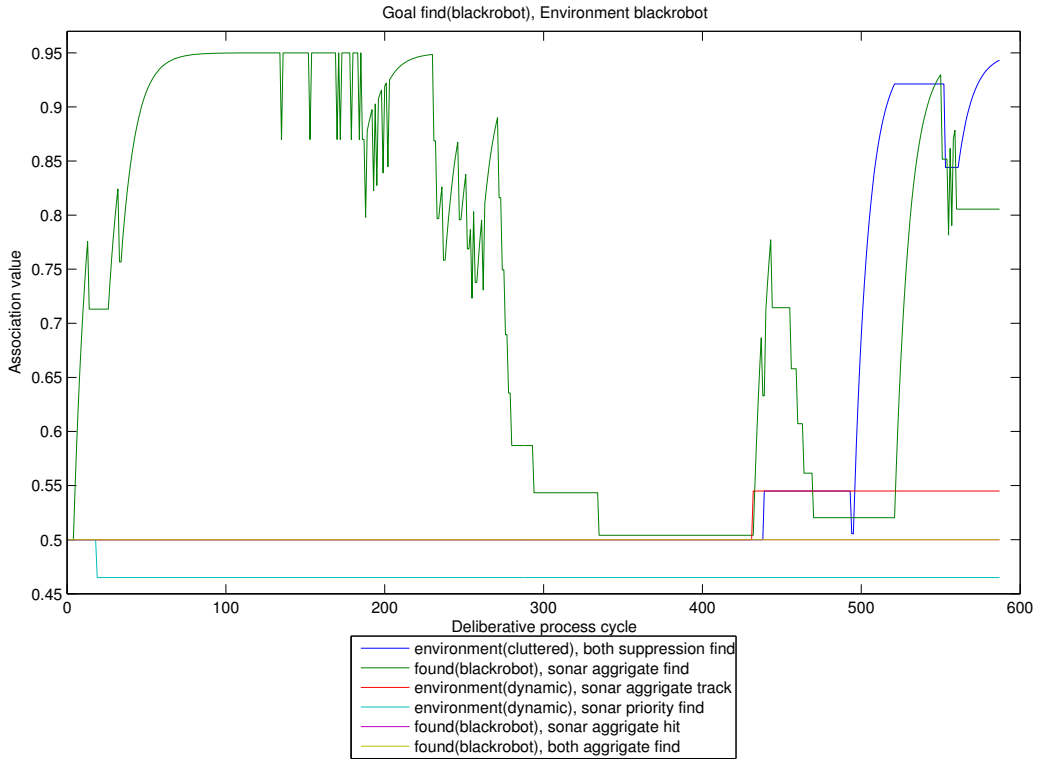
The second set of results relate to the second learning experiment. In this experiment robo-CAMAL operated in a controlled environment and attempted to achieve a single goal. Robo-CAMAL was provided a number of associations designed to achieve its goal. Robo-CAMAL had learn which ones provided the most optimal performance to achieve its goal.

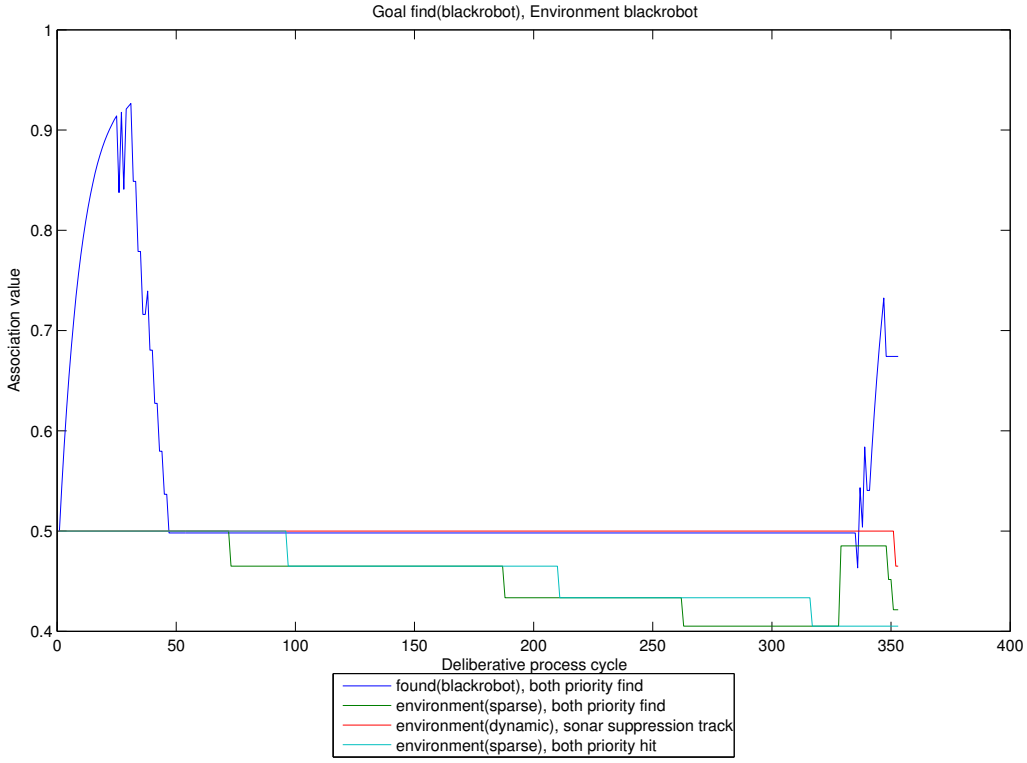
The final set of results relate to the adaptation experiments. In this experiment robo-CAMAL was placed in a changing environment with several goals and instantiated with a number of known good associations. Robo-CAMAL then had to adapt its goals to reflect the changes in its environment.

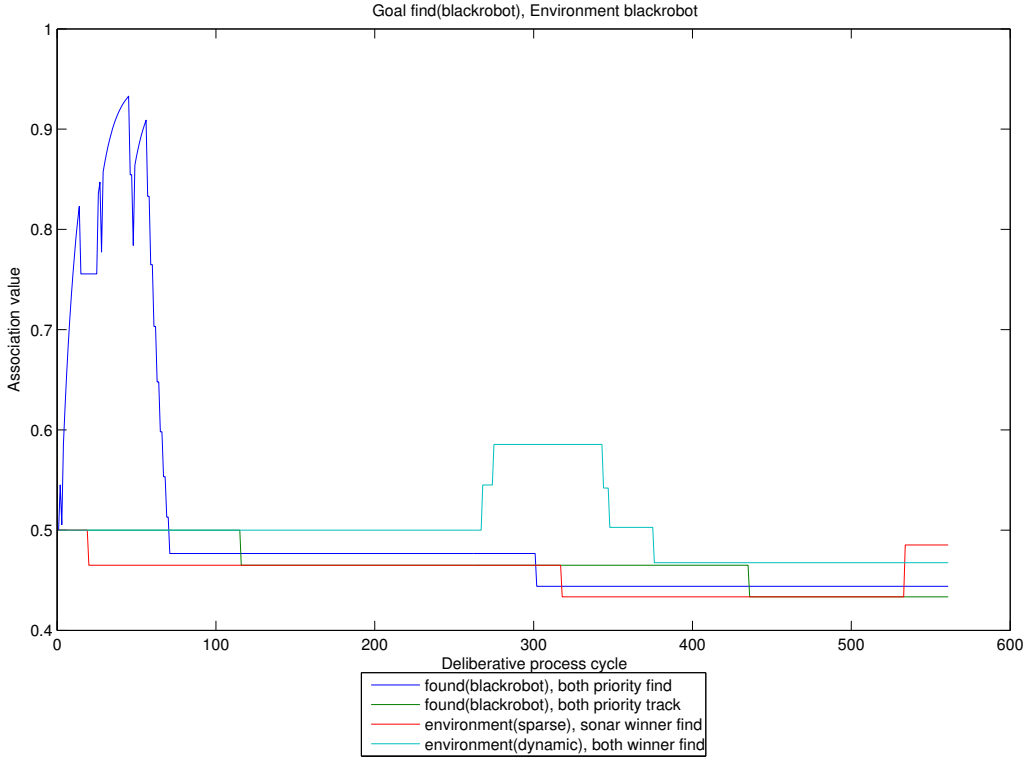
C.2 First Learning Experimental Results

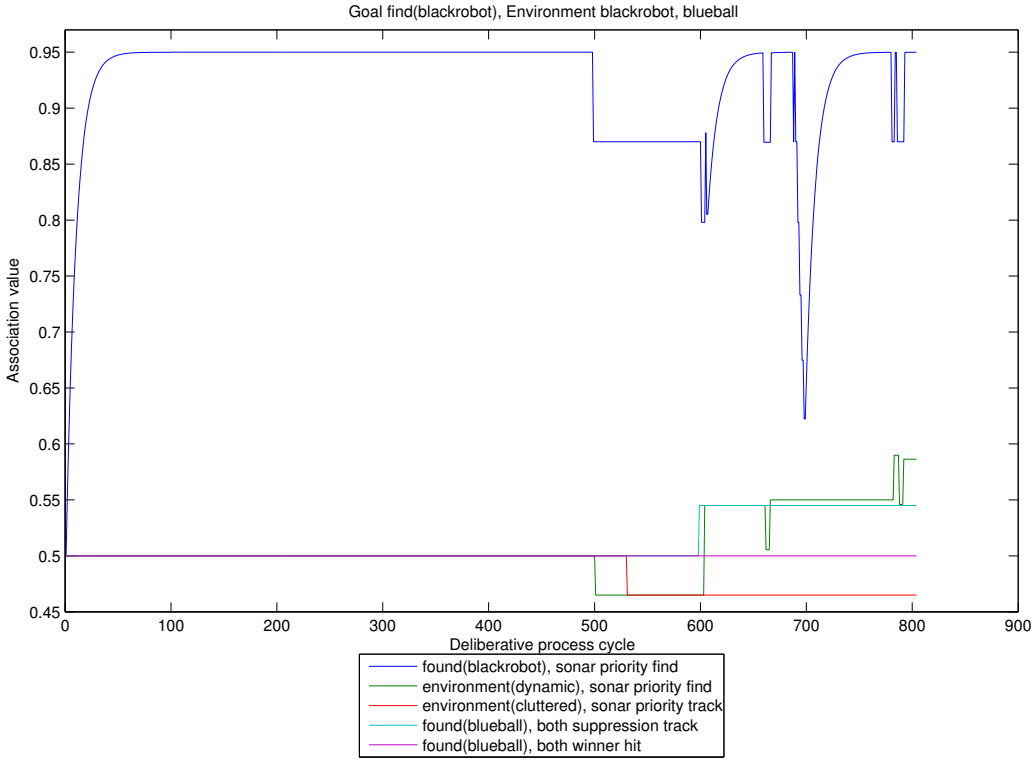
The results can be read as follows. The title of each graph represents robo-CAMAL's goal, and the objects present in the controlled environment. Each line is represents an association, as the associations goal is listed in the title, only the belief and the action are listed in the key. The y-axis represents the associations value, and the x-axis represents deliberative processing cycle the associations value was recorded at. As robo-CAMAL can generate over 50 associations in a single run, only the most relevant are displayed for clarity.

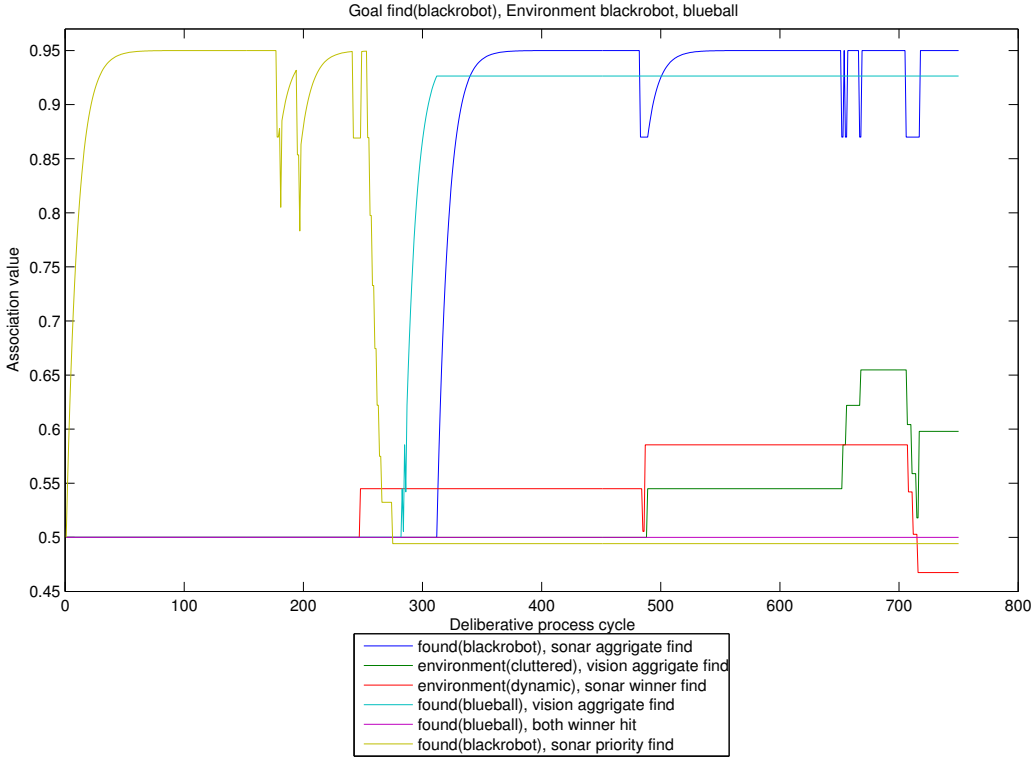
C.2.1 *find(blackrobot)*

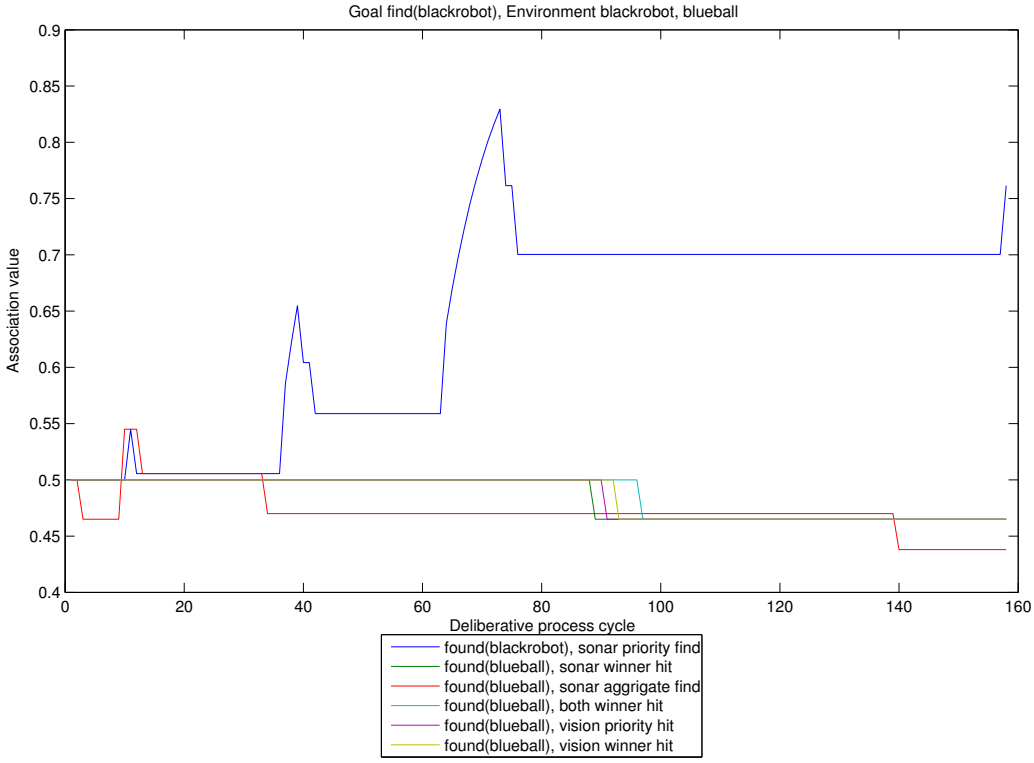


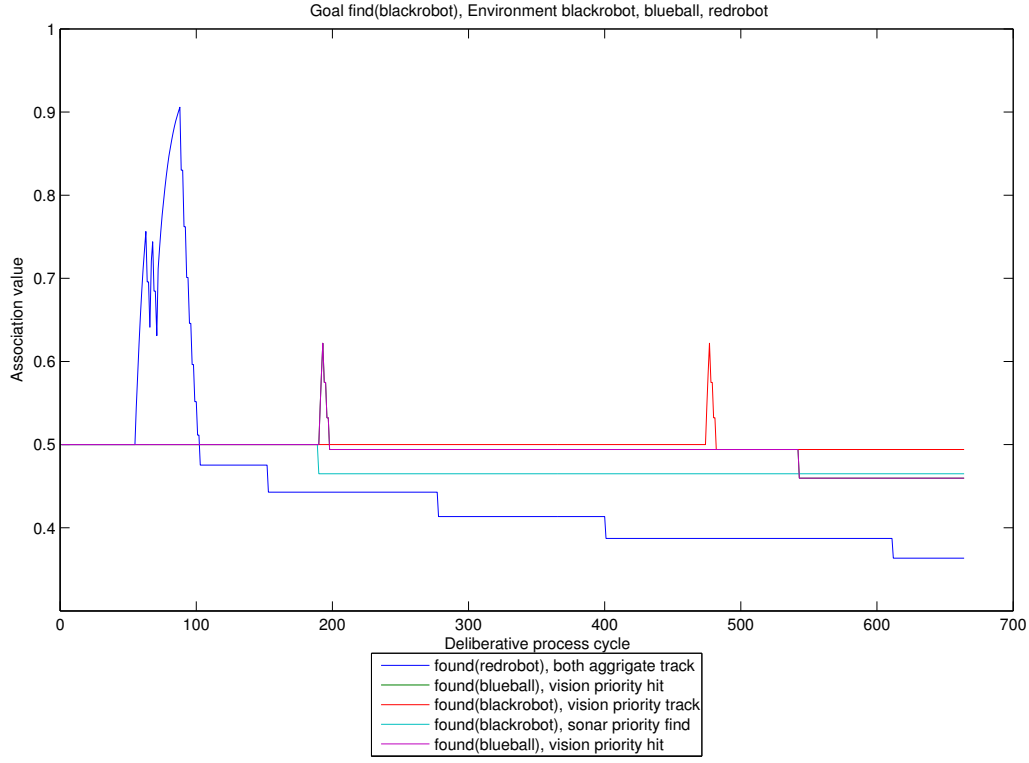

Goal find(blackrobot), Environment blackrobot
Association value
Deliberative process cycle
found(blackrobot), both priority find
environment(sparse), both priority find
environment(dynamic), sonar suppression track
environment(sparse), both priority hit

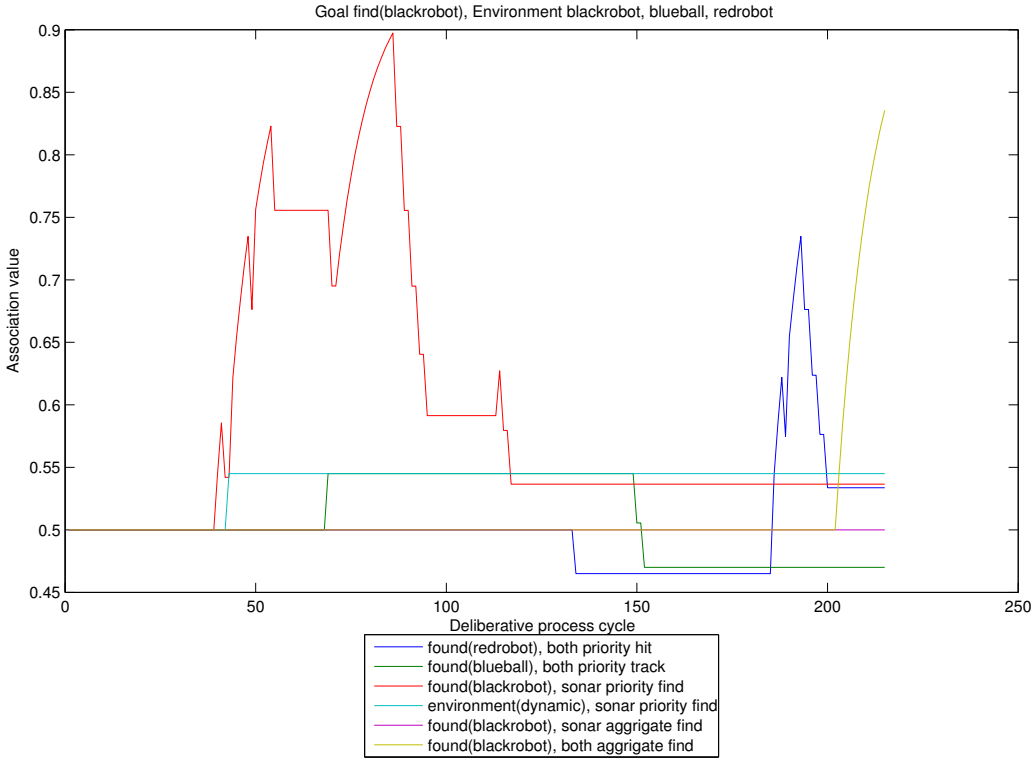

Goal find(blackrobot), Environment blackrobot
Association value
Deliberative process cycle
found(blackrobot), both priority find
found(blackrobot), both priority track
environment(sparse), sonar winner find
environment(dynamic), both winner find

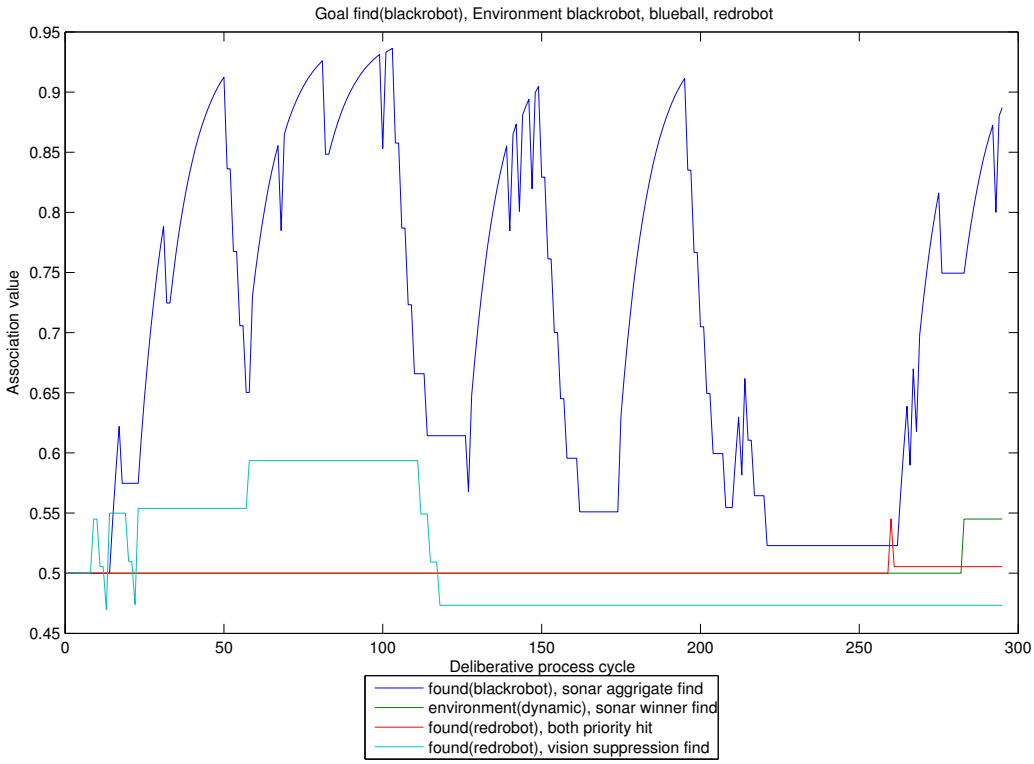

Goal find(blackrobot), Environment blackrobot, blueball
Association value
Deliberative process cycle
found(blackrobot), sonar priority find
environment(dynamic), sonar priority find
environment(cluttered), sonar priority track
found(blueball), both suppression track
found(blueball), both winner hit

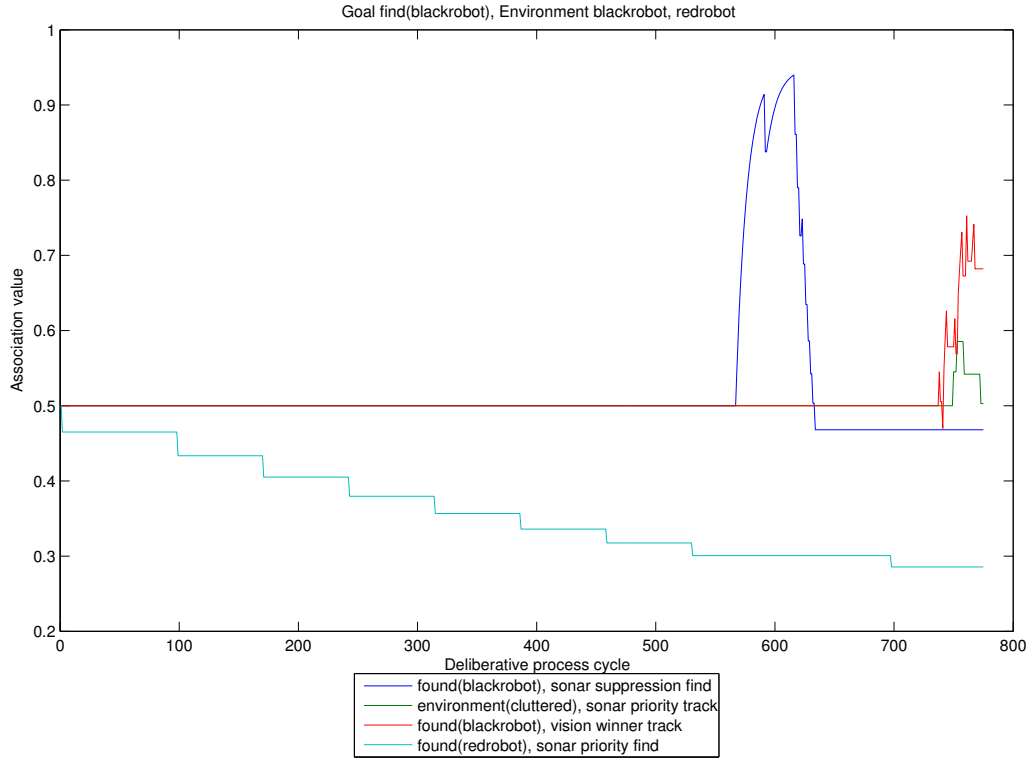

Goal find(blackrobot), Environment blackrobot, blueball
Association value
Deliberative process cycle
found(blackrobot), sonar aggrigate find
environment(cluttered), vision aggrigate find
environment(dynamic), sonar winner find
found(blueball), vision aggrigate find
found(blueball), both winner hit
found(blackrobot), sonar priority find

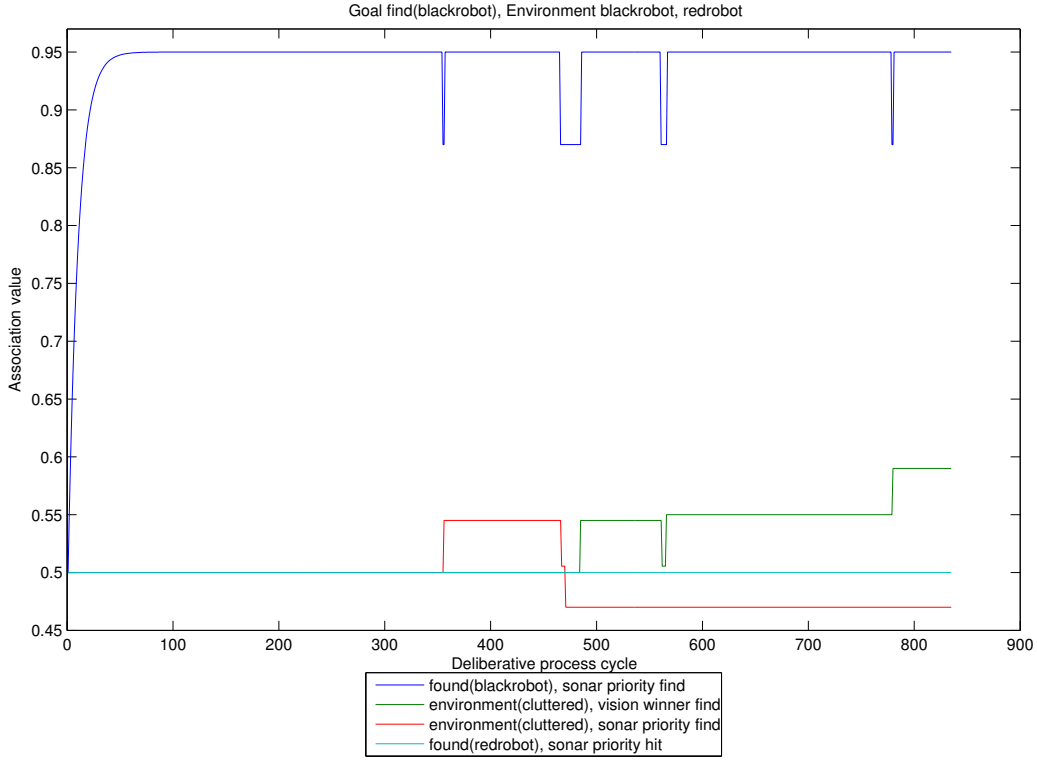

Goal find(blackrobot), Environment blackrobot, blueball
Association value
Deliberative process cycle
found(blackrobot), sonar priority find
found(blueball), sonar winner hit
found(blueball), sonar aggrigate find
found(blueball), both winner hit
found(blueball), vision priority hit
found(blueball), vision winner hit

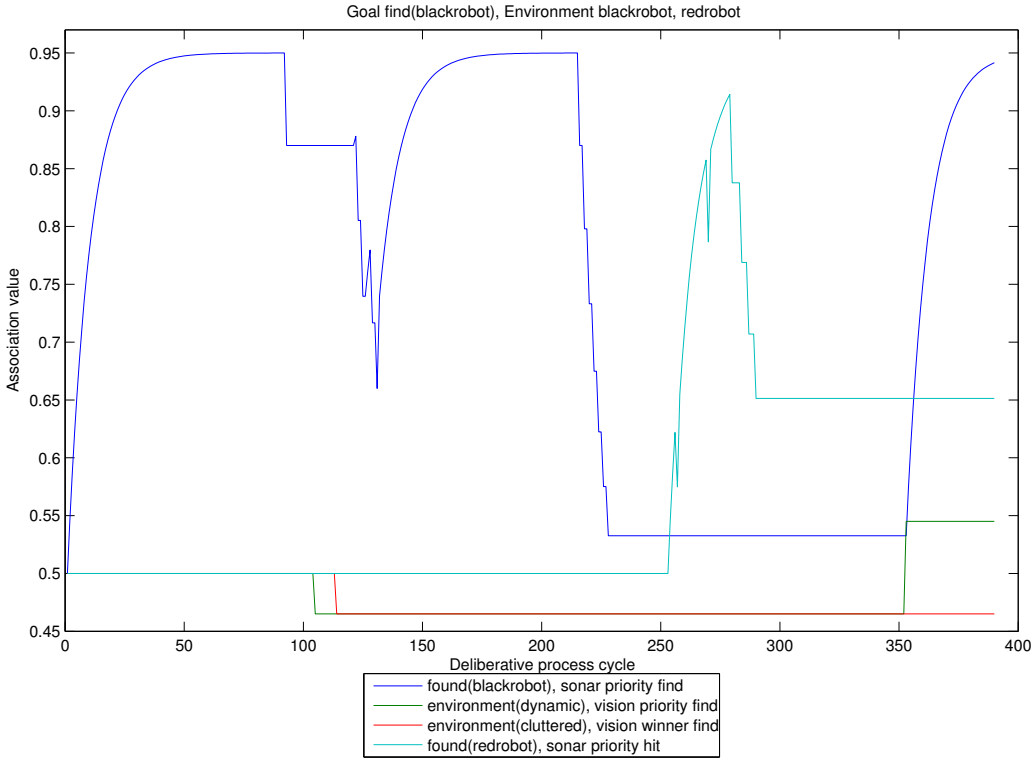

Goal find(blackrobot), Environment blackrobot, blueball, redrobot
Association value
Deliberative process cycle
found(redrobot), both aggrigate track
found(blueball), vision priority hit
found(blackrobot), vision priority track
found(blackrobot), sonar priority find
found(blueball), vision priority hit

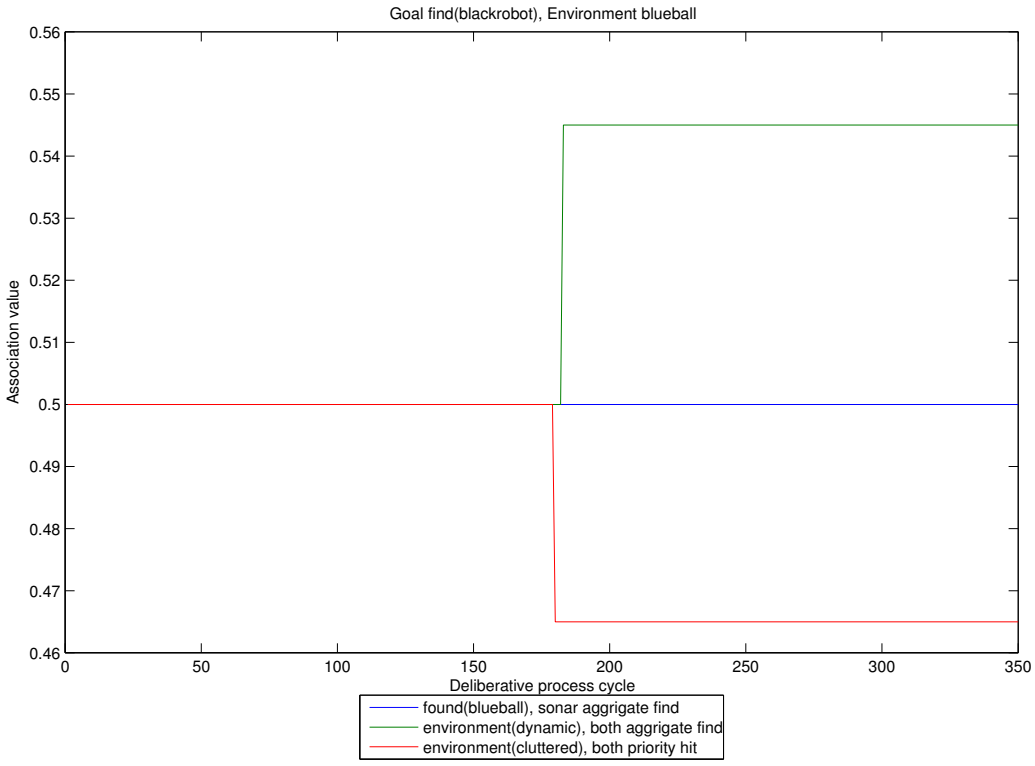

Goal find(blackrobot), Environment blackrobot, blueball, redrobot
Association value
Deliberative process cycle
found(redrobot), both priority hit
found(blueball), both priority track
found(blackrobot), sonar priority find
environment(dynamic), sonar priority find
found(blackrobot), sonar aggrigate find
found(blackrobot), both aggrigate find

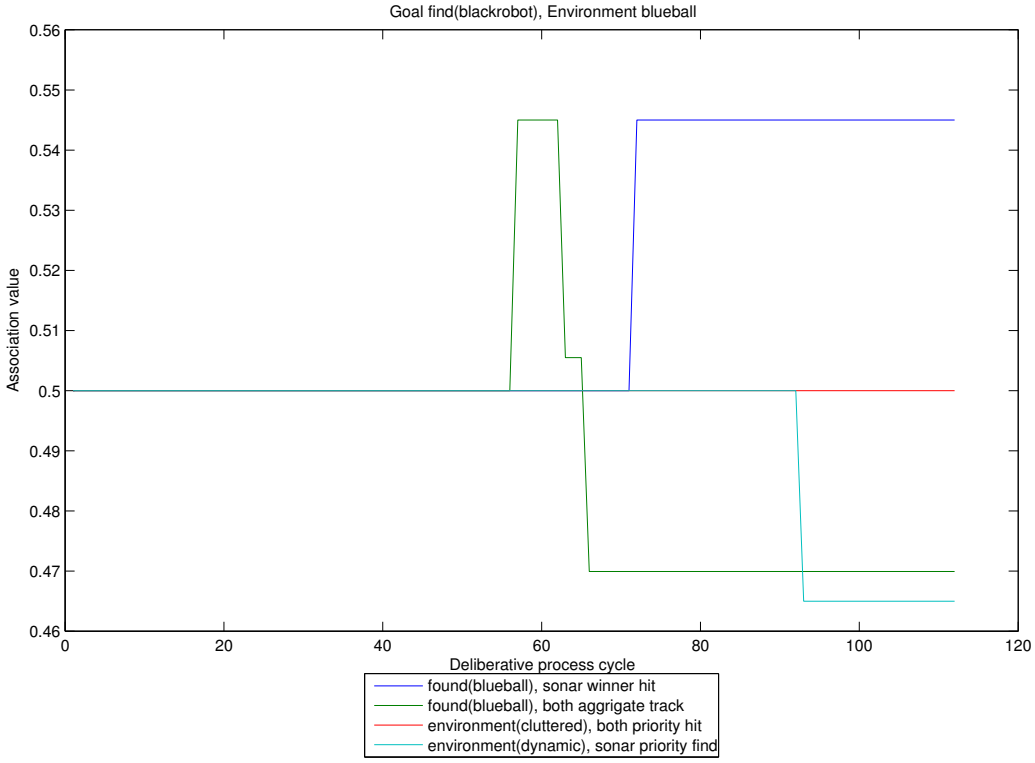

Goal find(blackrobot), Environment blackrobot, blueball, redrobot
Association value
Deliberative process cycle
found(blackrobot), sonar aggrigate find
environment(dynamic), sonar winner find
found(redrobot), both priority hit
found(redrobot), vision suppression find

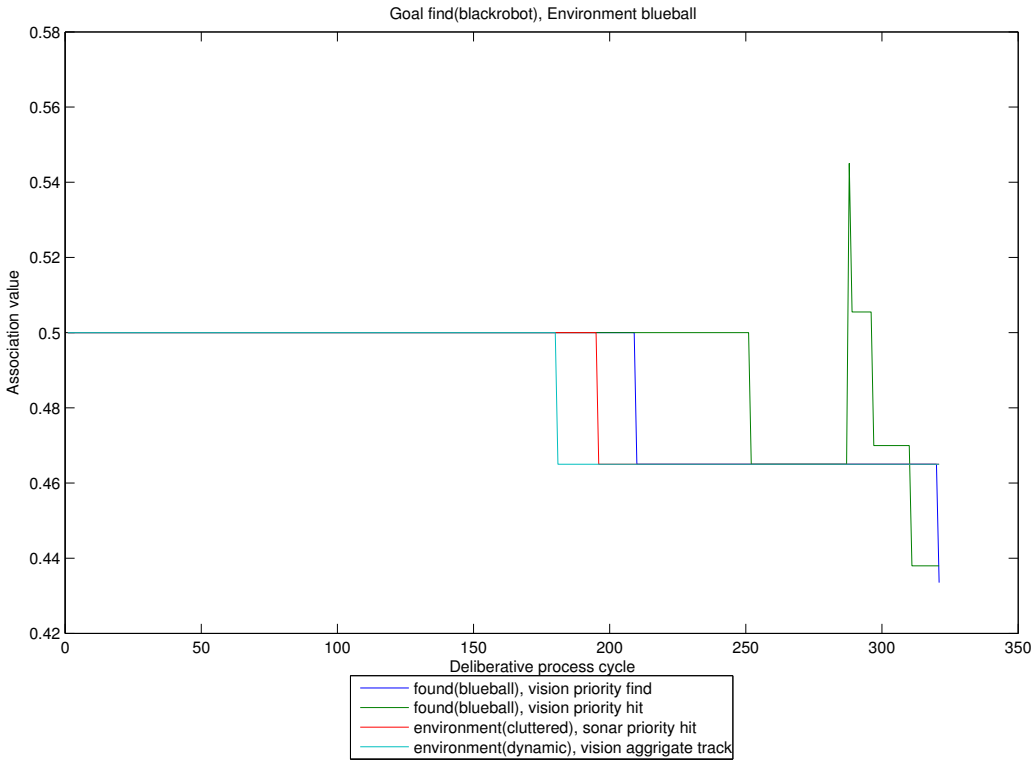

Goal find(blackrobot), Environment blackrobot, redrobot
Association value
Deliberative process cycle
found(blackrobot), sonar suppression find
environment(cluttered), sonar priority track
found(blackrobot), vision winner track
found(redrobot), sonar priority find

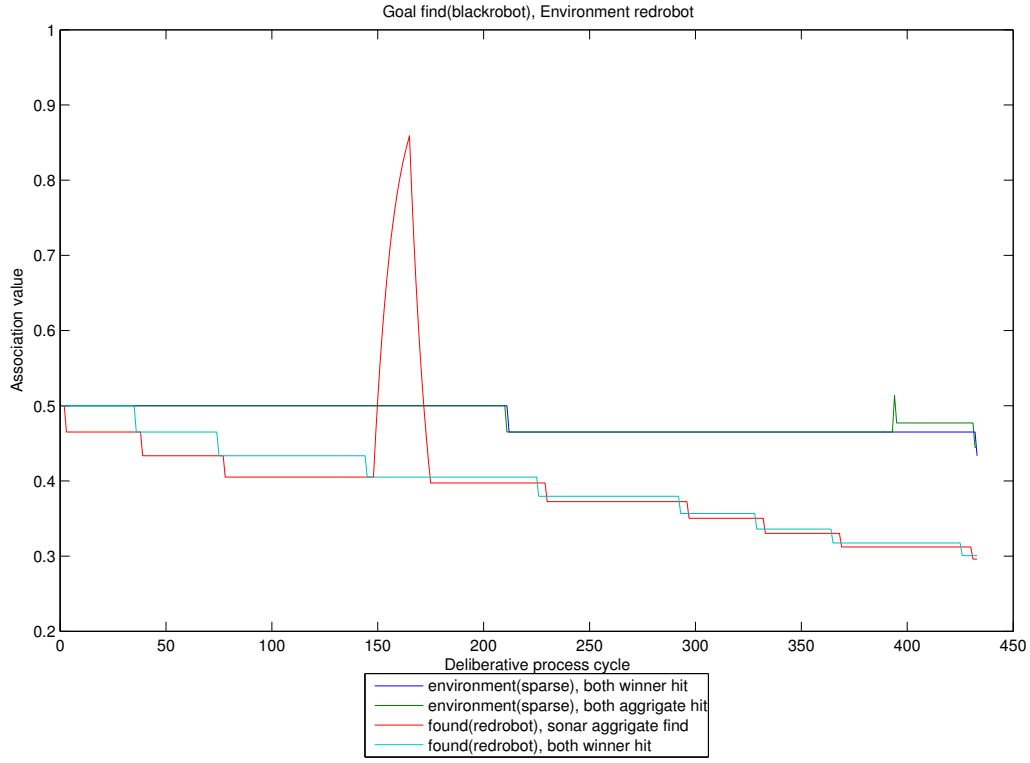

Goal find(blackrobot), Environment blackrobot, redrobot
Association value
Deliberative process cycle
found(blackrobot), sonar priority find
environment(cluttered), vision winner find
environment(cluttered), sonar priority find
found(redrobot), sonar priority hit

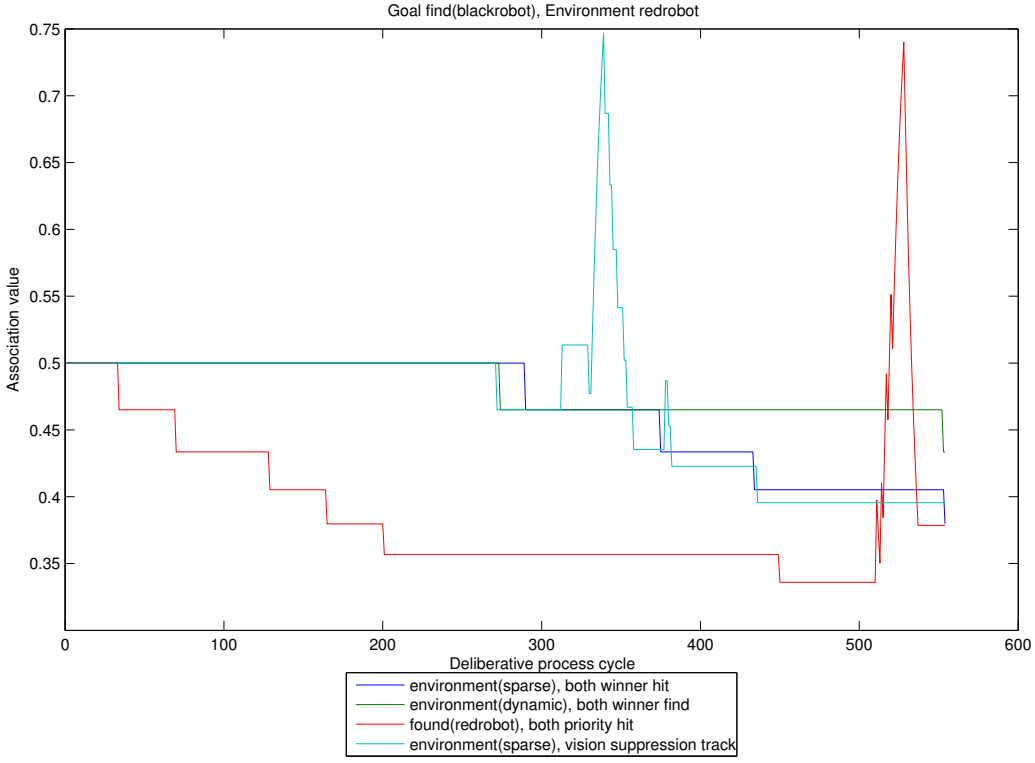

Goal find(blackrobot), Environment blackrobot, redrobot
Association value
Deliberative process cycle
found(blackrobot), sonar priority find
environment(dynamic), vision priority find
environment(cluttered), vision winner find
found(redrobot), sonar priority hit

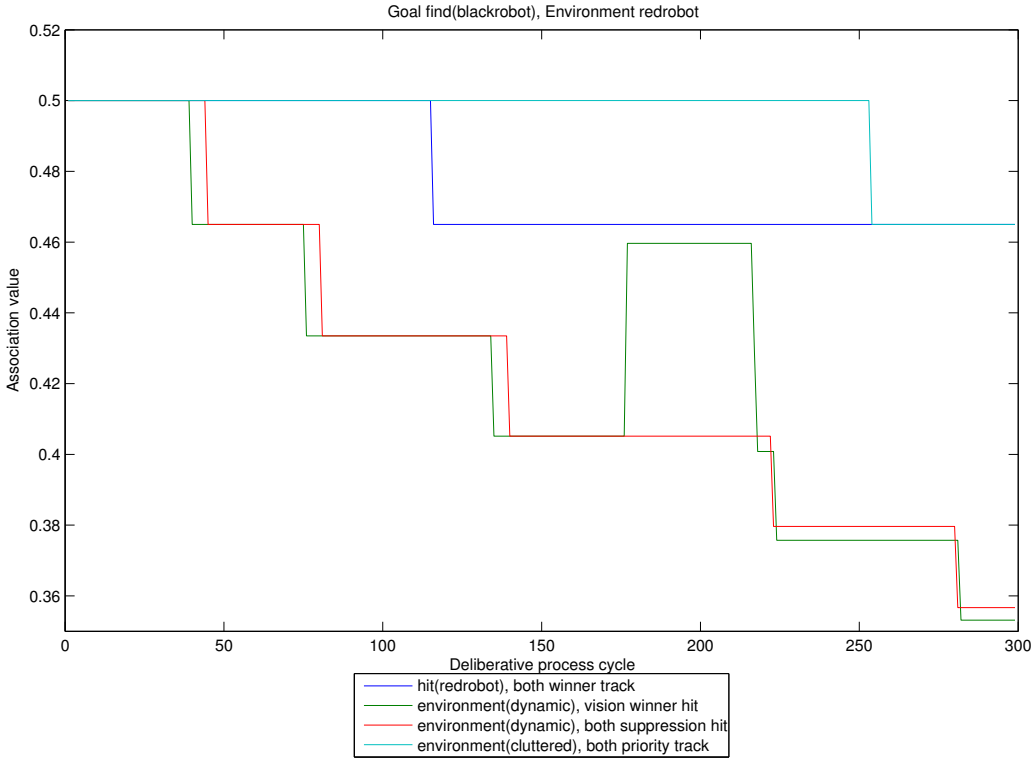

Goal find(blackrobot), Environment blueball
Association value
Deliberative process cycle
found(blueball), sonar aggrigate find
environment(dynamic), both aggrigate find
environment(cluttered), both priority hit


Goal find(blackrobot), Environment blueball
Association value
Deliberative process cycle
found(blueball), sonar winner hit
found(blueball), both aggrigate track
environment(cluttered), both priority hit
environment(dynamic), sonar priority find

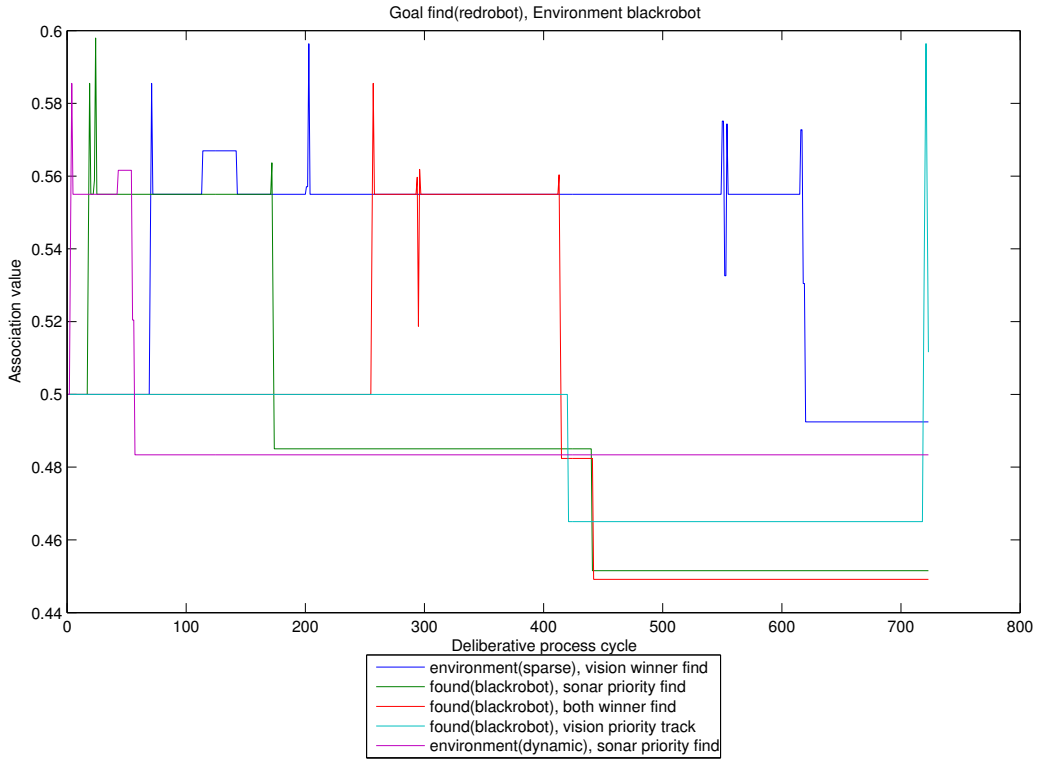

Goal find(blackrobot), Environment blueball
Association value
Deliberative process cycle
found(blueball), vision priority find
found(blueball), vision priority hit
environment(cluttered), sonar priority hit
environment(dynamic), vision aggrigate track

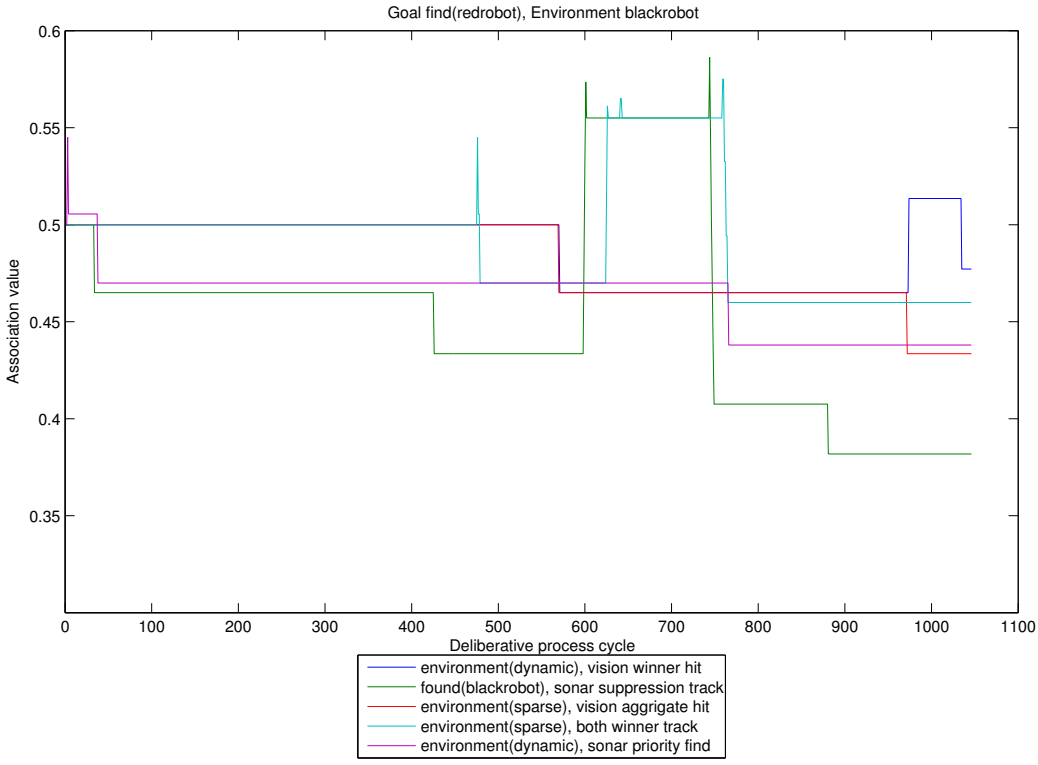

Goal find(blackrobot), Environment redrobot
Association value
Deliberative process cycle
environment(sparse), both winner hit
environment(sparse), both aggrigate hit
found(redrobot), sonar aggrigate find
found(redrobot), both winner hit

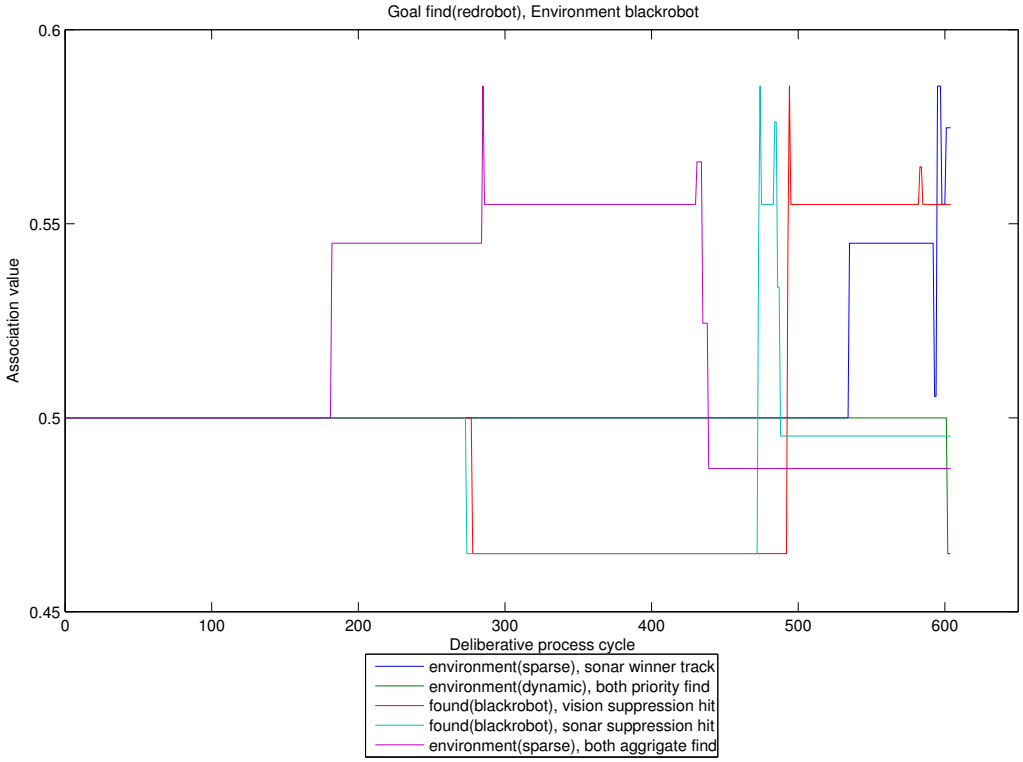

Goal find(blackrobot), Environment redrobot
Association value
Deliberative process cycle
environment(sparse), both winner hit
environment(dynamic), both winner find
found(redrobot), both priority hit
environment(sparse), vision suppression track

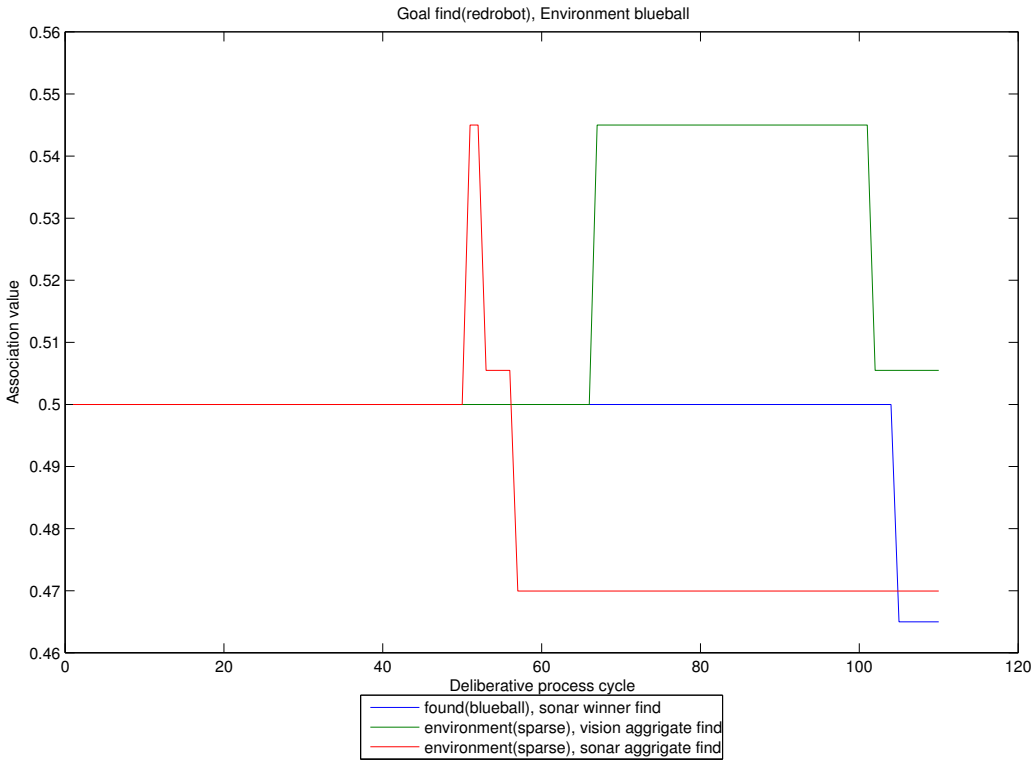

Goal find(blackrobot), Environment redrobot
Association value
Deliberative process cycle
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0
50
100
150
200
250
300
hit(redrobot), both winner track
environment(dynamic), vision winner hit
environment(dynamic), both suppression hit
environment(cluttered), both priority track

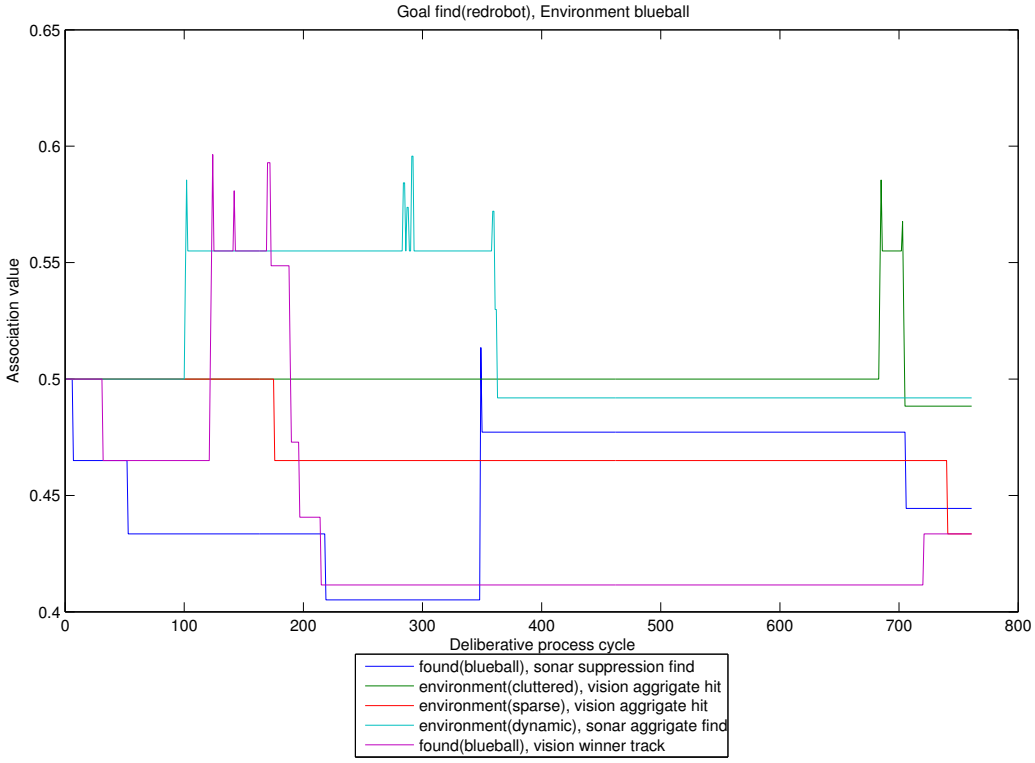
C.2.2 *find(redrobot)*

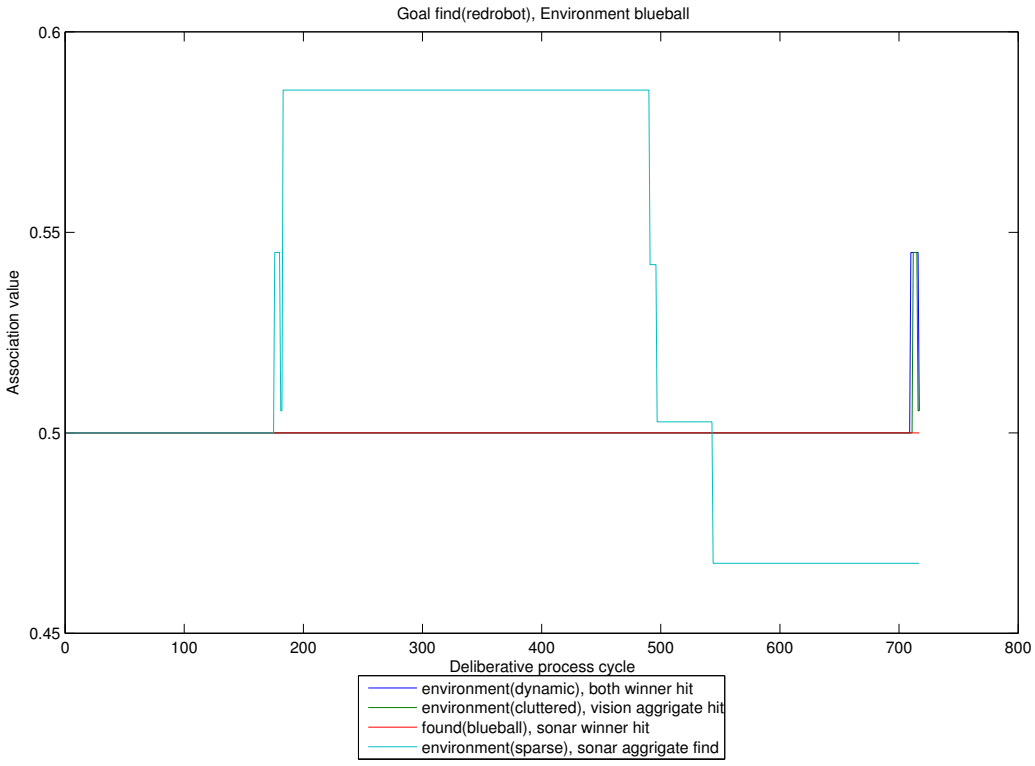


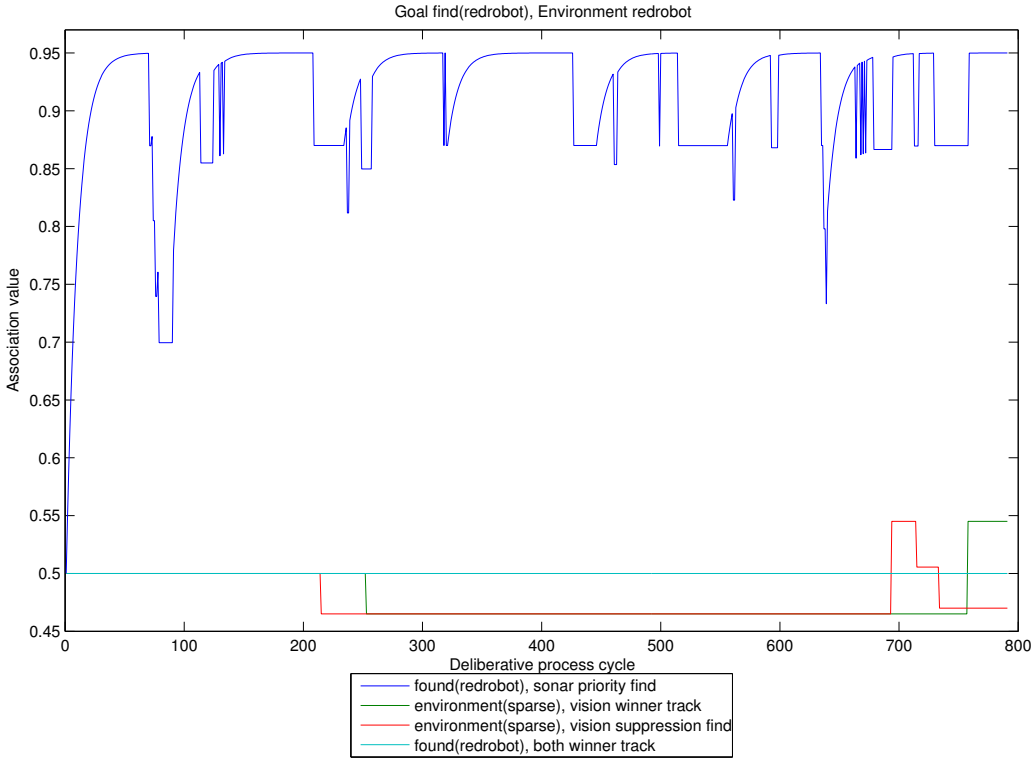


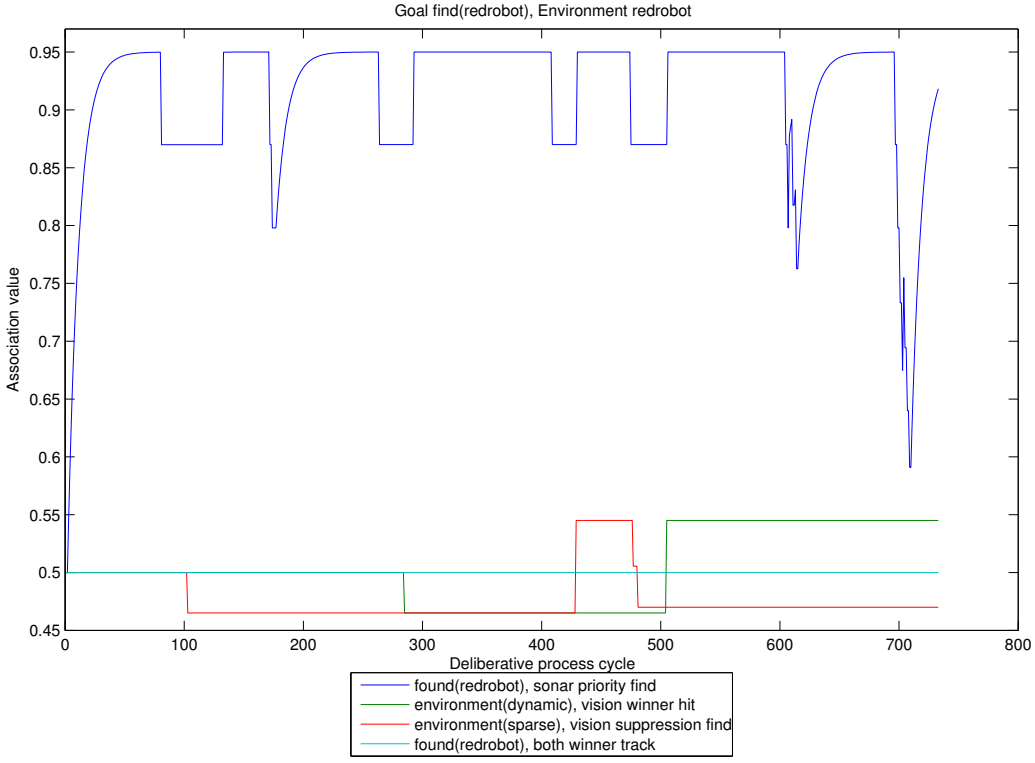

Goal find(redrobot), Environment blackrobot
Association value
Deliberative process cycle
environment(sparse), sonar winner track
environment(dynamic), both priority find
found(blackrobot), vision suppression hit
found(blackrobot), sonar suppression hit
environment(sparse), both aggrigate find

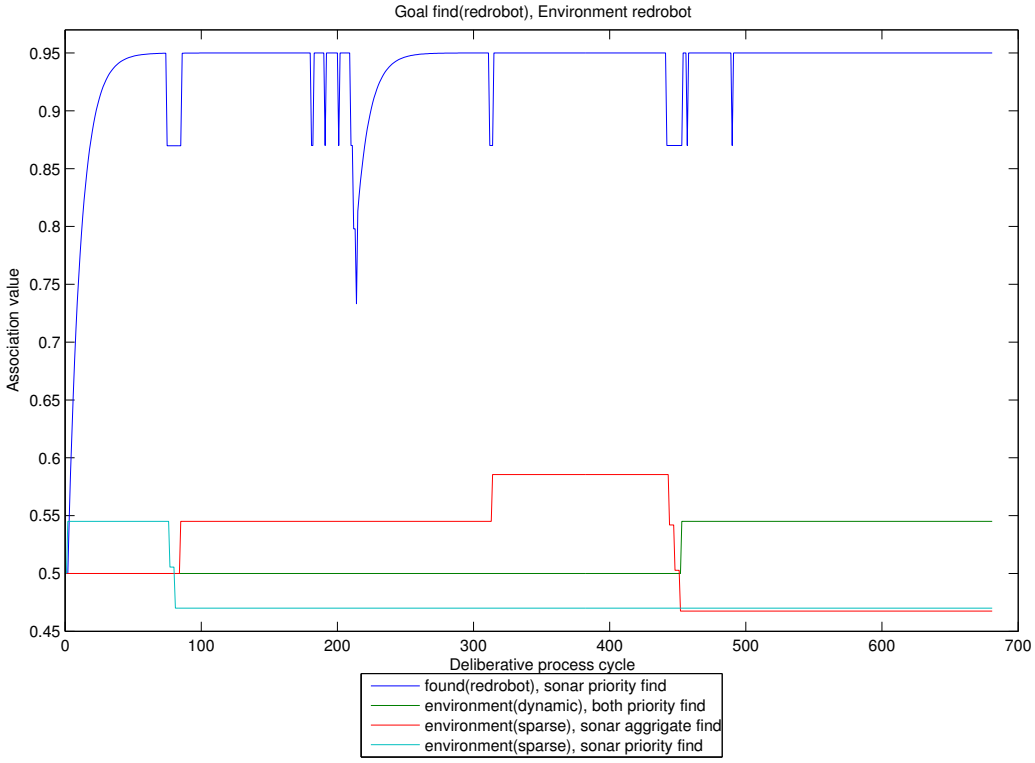

Goal find(redrobot), Environment blueball
Association value
Deliberative process cycle
found(blueball), sonar winner find
environment(sparse), vision aggrigate find
environment(sparse), sonar aggrigate find

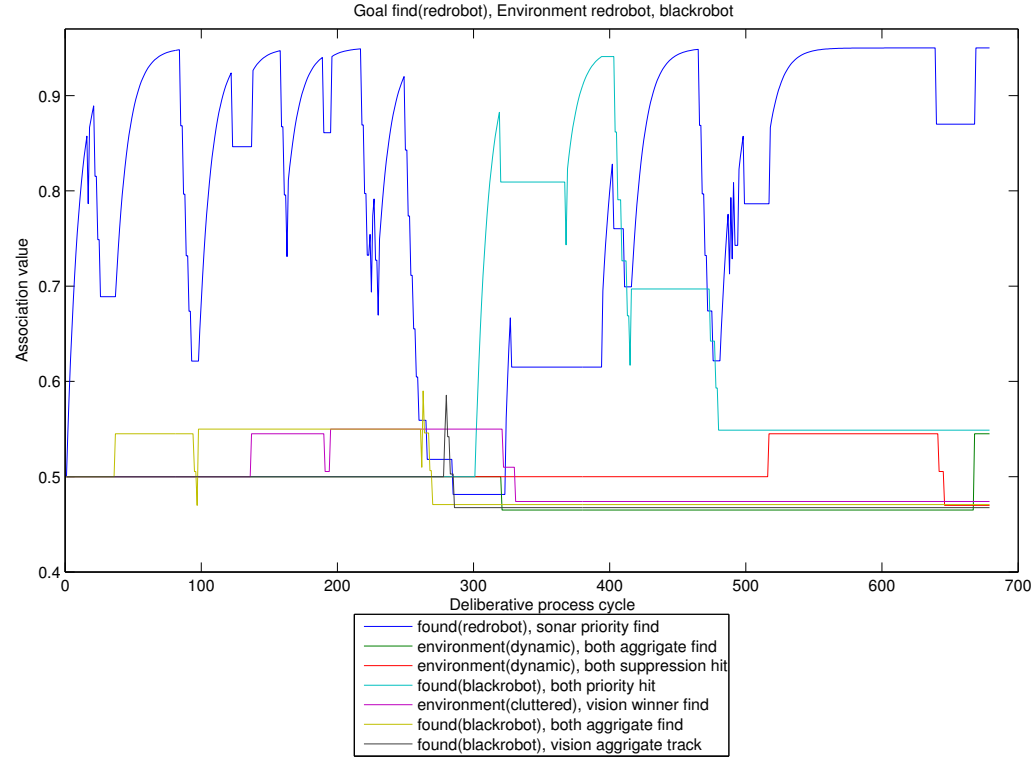

Goal find(redrobot), Environment blueball
Association value
Deliberative process cycle
found(blueball), sonar suppression find
environment(cluttered), vision aggrigate hit
environment(sparse), vision aggrigate hit
environment(dynamic), sonar aggrigate find
found(blueball), vision winner track

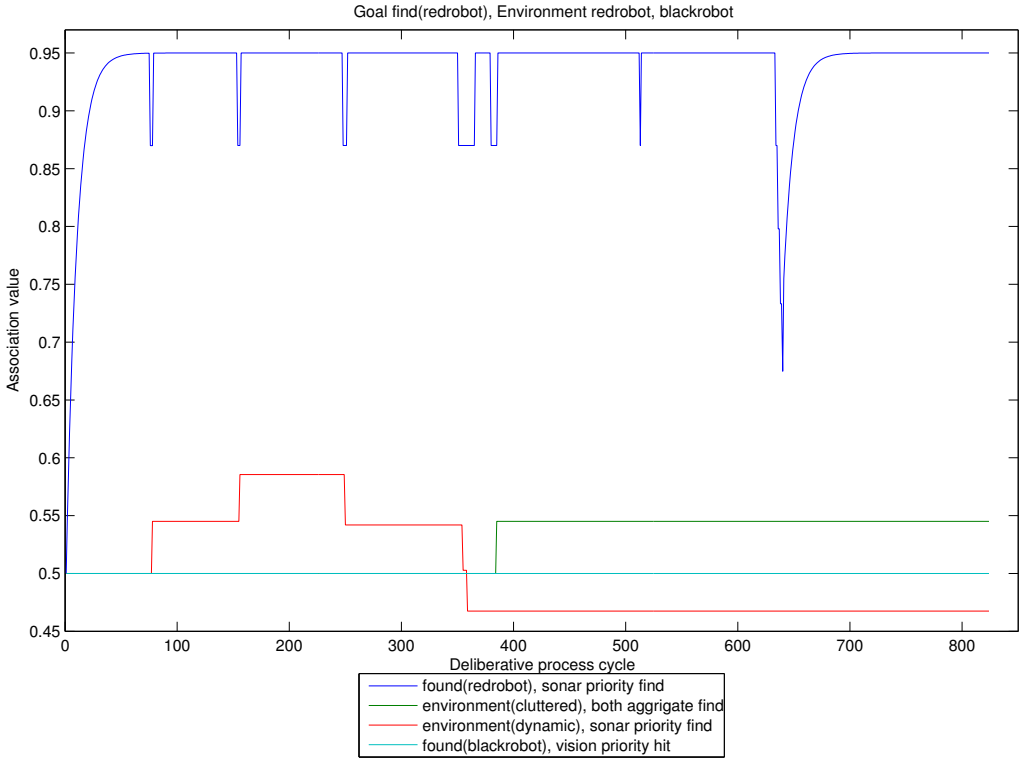

Goal find(redrobot), Environment blueball
Association value
Deliberative process cycle
environment(dynamic), both winner hit
environment(cluttered), vision aggrigate hit
found(blueball), sonar winner hit
environment(sparse), sonar aggrigate find

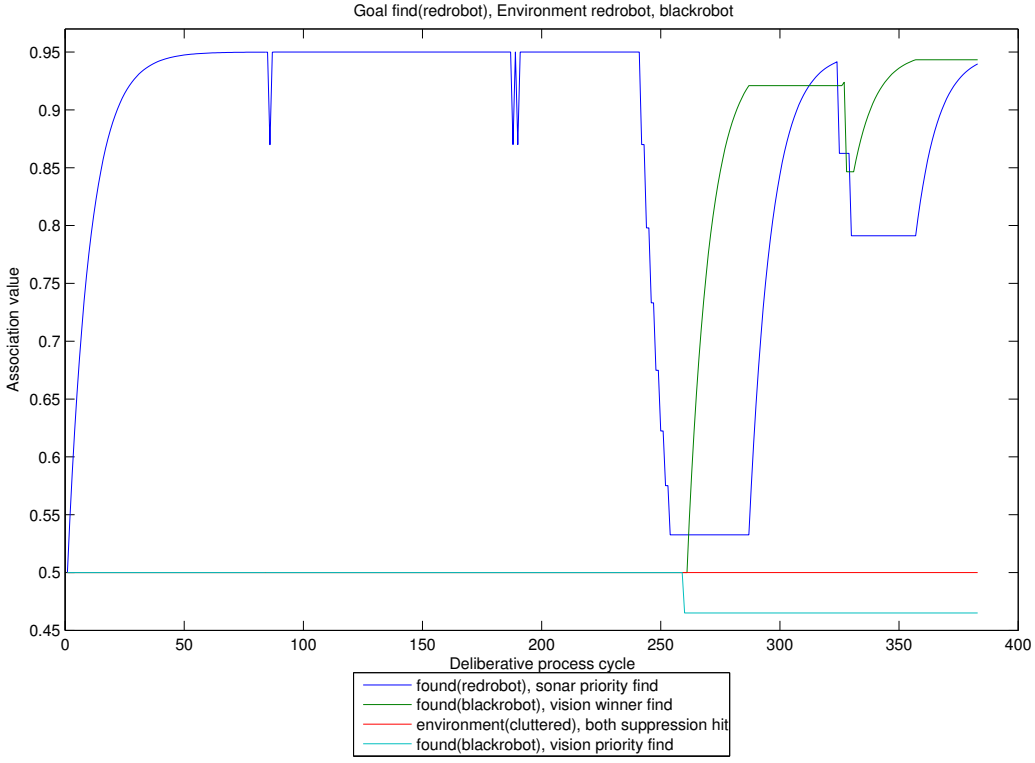

Goal find(redrobot), Environment redrobot
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(sparse), vision winner track
environment(sparse), vision suppression find
found(redrobot), both winner track

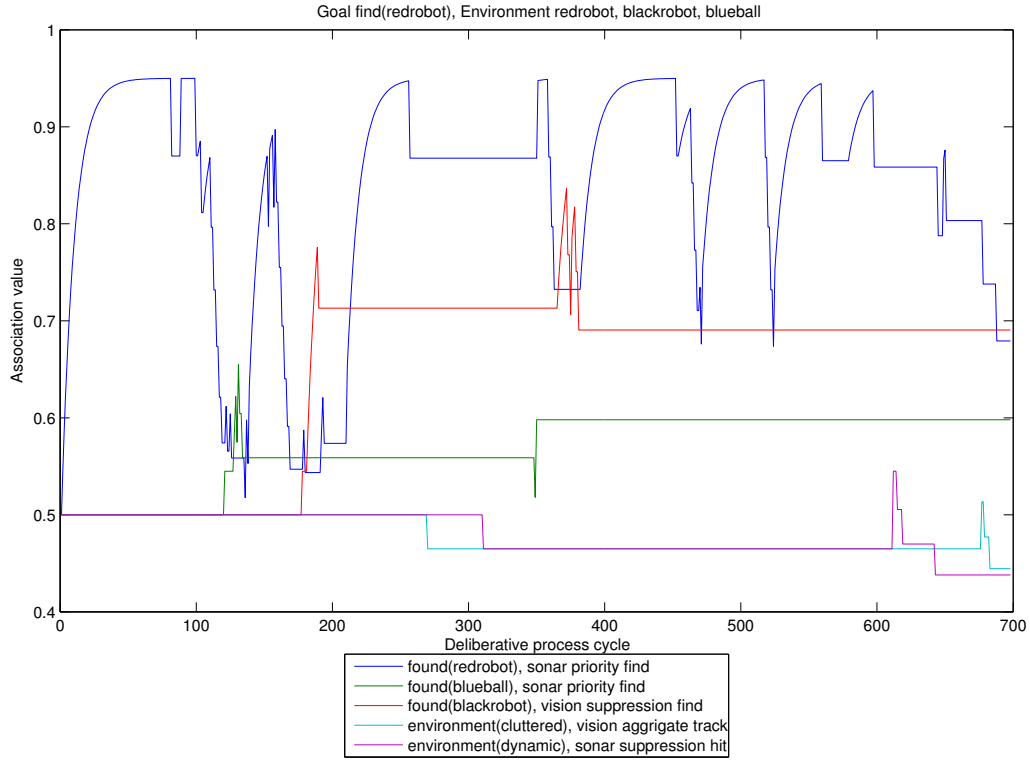

Goal find(redrobot), Environment redrobot
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(dynamic), vision winner hit
environment(sparse), vision suppression find
found(redrobot), both winner track

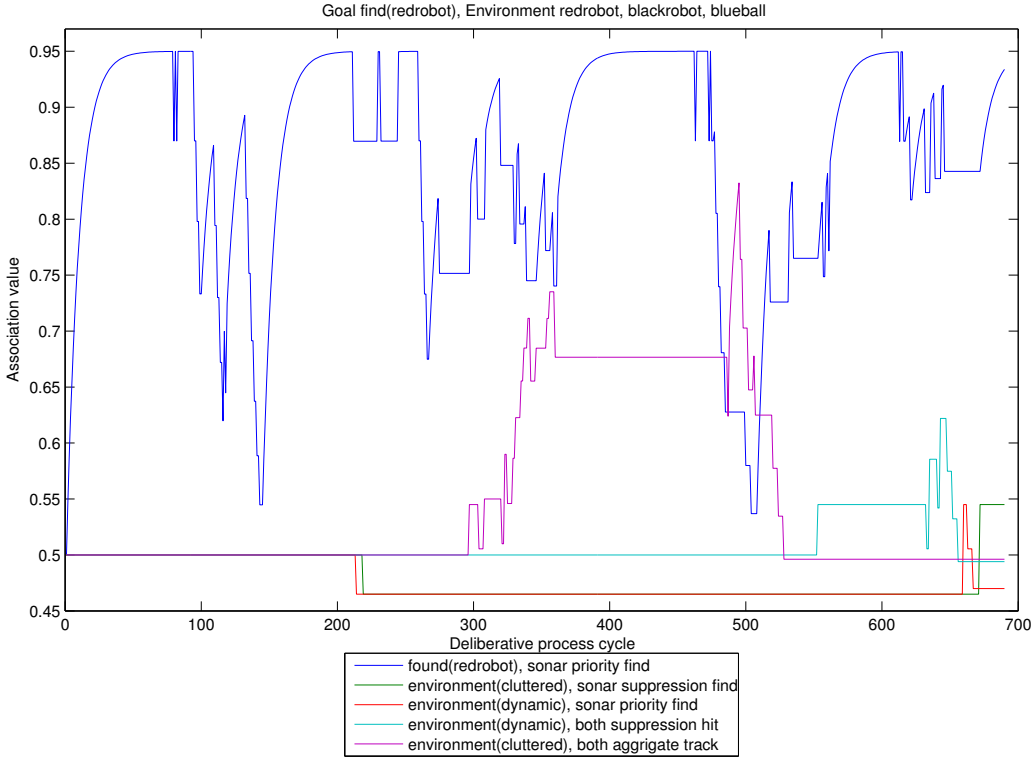

Goal find(redrobot), Environment redrobot
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(dynamic), both priority find
environment(sparse), sonar aggrigate find
environment(sparse), sonar priority find

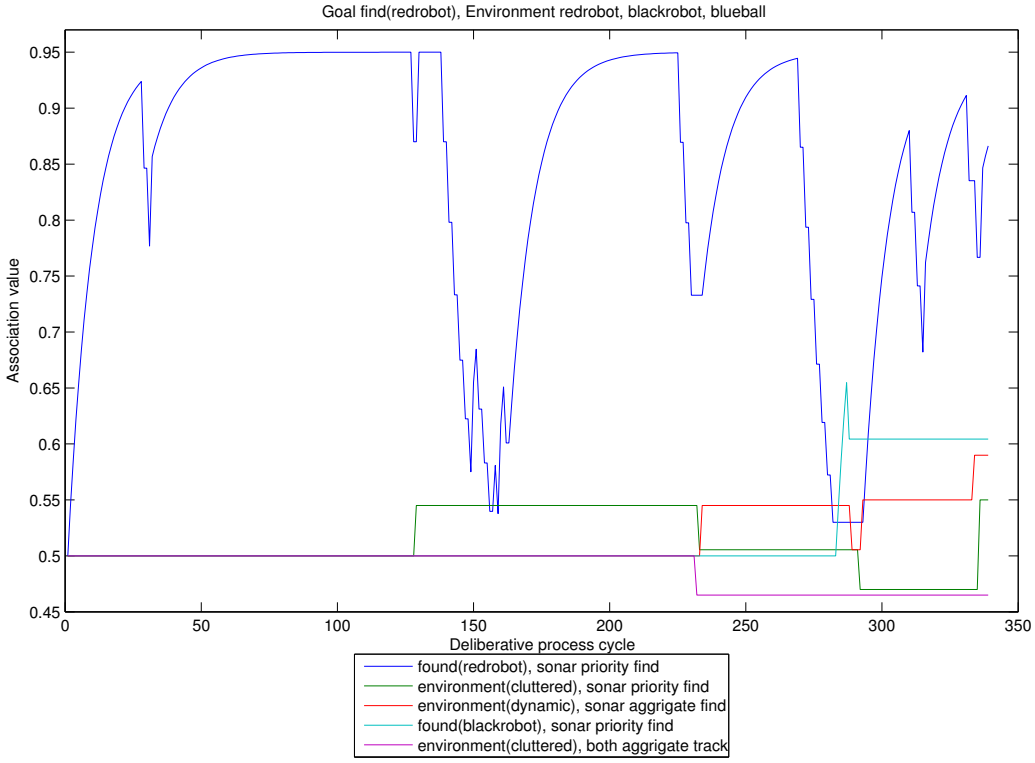

Goal find(redrobot), Environment redrobot, blackrobot
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(dynamic), both aggrigate find
environment(dynamic), both suppression hit
found(blackrobot), both priority hit
environment(cluttered), vision winner find
found(blackrobot), both aggrigate find
found(blackrobot), vision aggrigate track

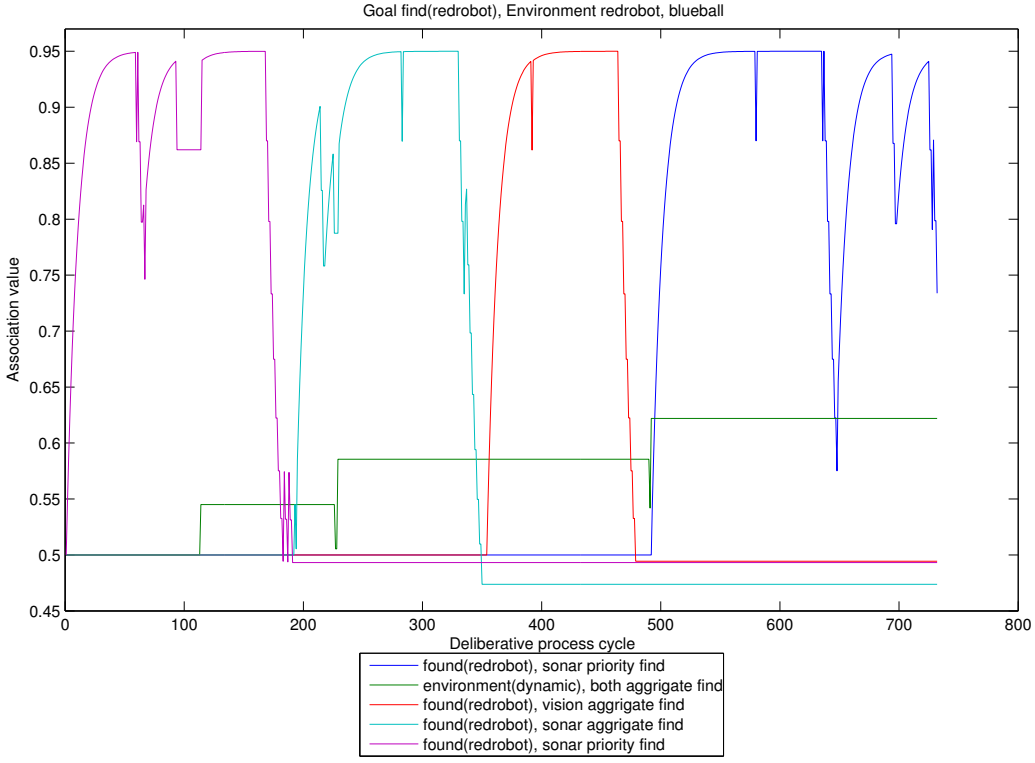

Goal find(redrobot), Environment redrobot, blackrobot
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(cluttered), both aggrigate find
environment(dynamic), sonar priority find
found(blackrobot), vision priority hit

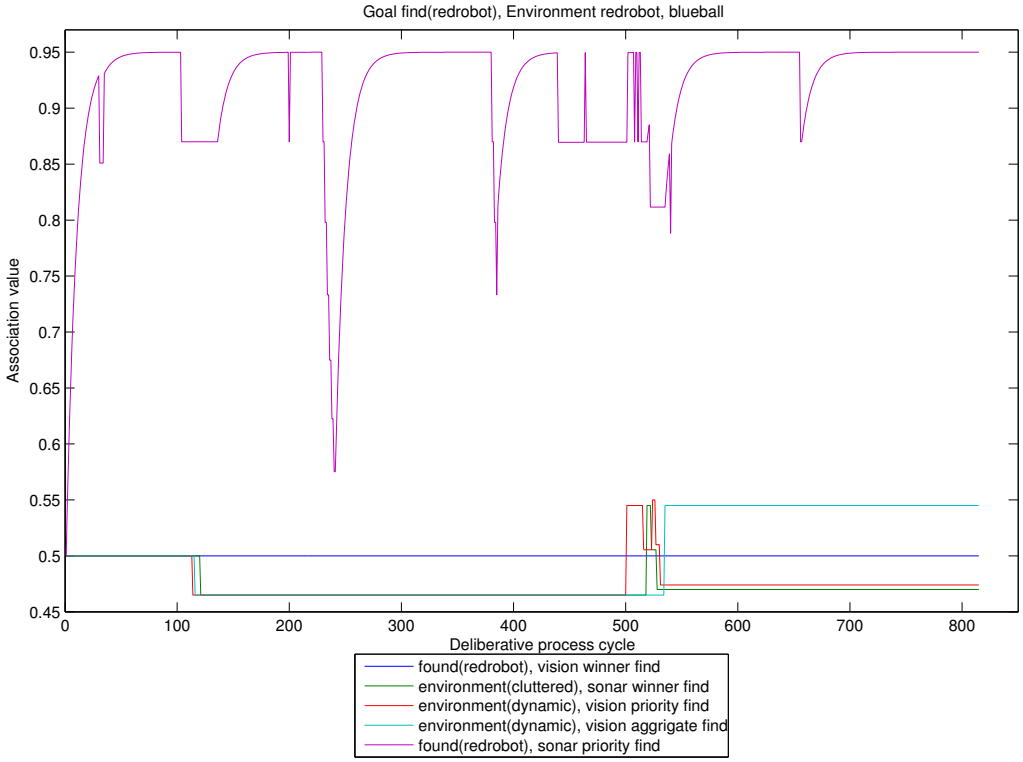

Goal find(redrobot), Environment redrobot, blackrobot
Association value
Deliberative process cycle
found(redrobot), sonar priority find
found(blackrobot), vision winner find
environment(cluttered), both suppression hit
found(blackrobot), vision priority find

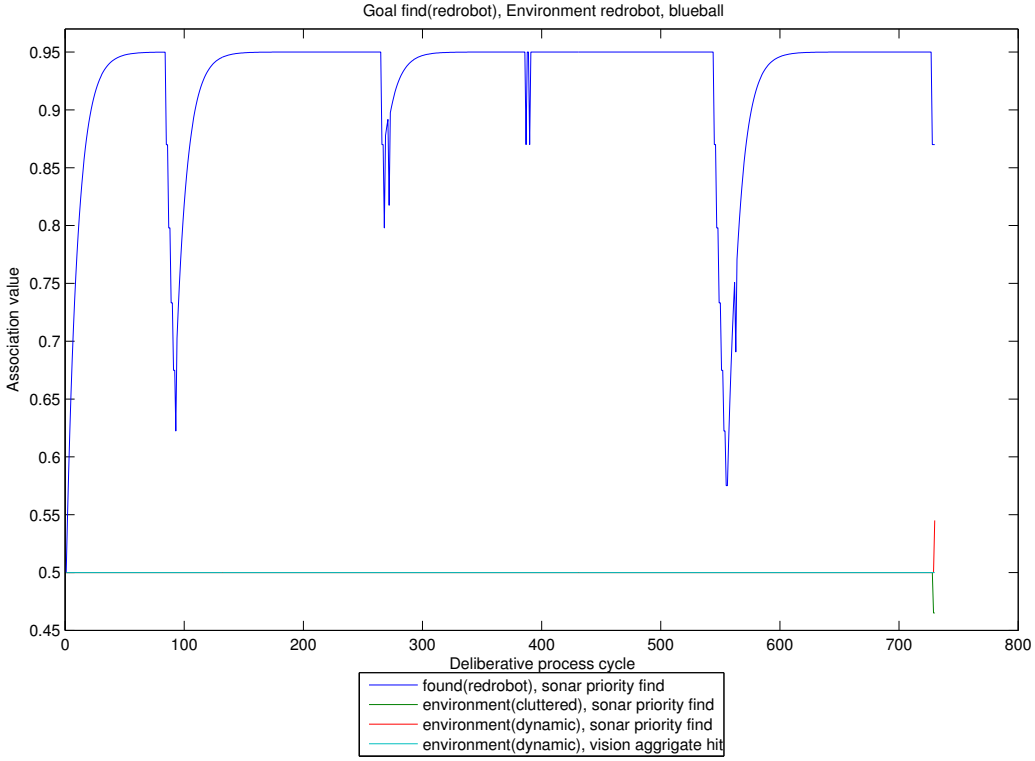

Goal find(redrobot), Environment redrobot, blackrobot, blueball
Association value
Deliberative process cycle
found(redrobot), sonar priority find
found(blueball), sonar priority find
found(blackrobot), vision suppression find
environment(cluttered), vision aggrigate track
environment(dynamic), sonar suppression hit


Goal find(redrobot), Environment redrobot, blackrobot, blueball
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(cluttered), sonar suppression find
environment(dynamic), sonar priority find
environment(dynamic), both suppression hit
environment(cluttered), both aggrigate track

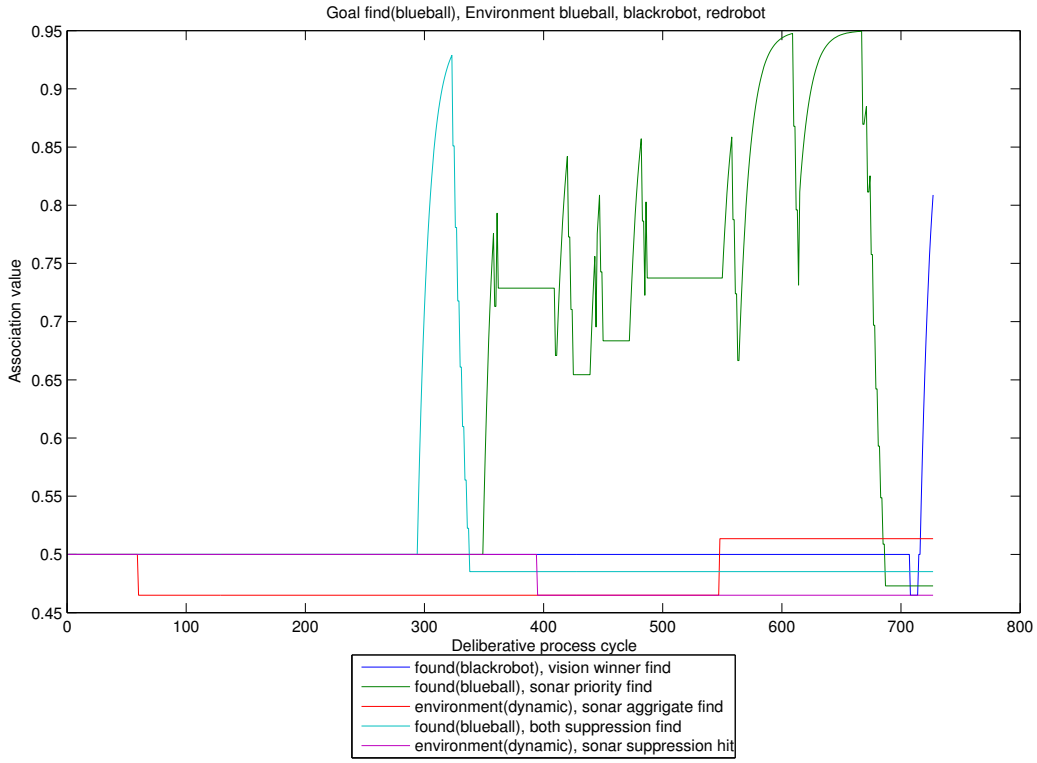

Goal find(redrobot), Environment redrobot, blackrobot, blueball
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(cluttered), sonar priority find
environment(dynamic), sonar aggrigate find
found(blackrobot), sonar priority find
environment(cluttered), both aggrigate track

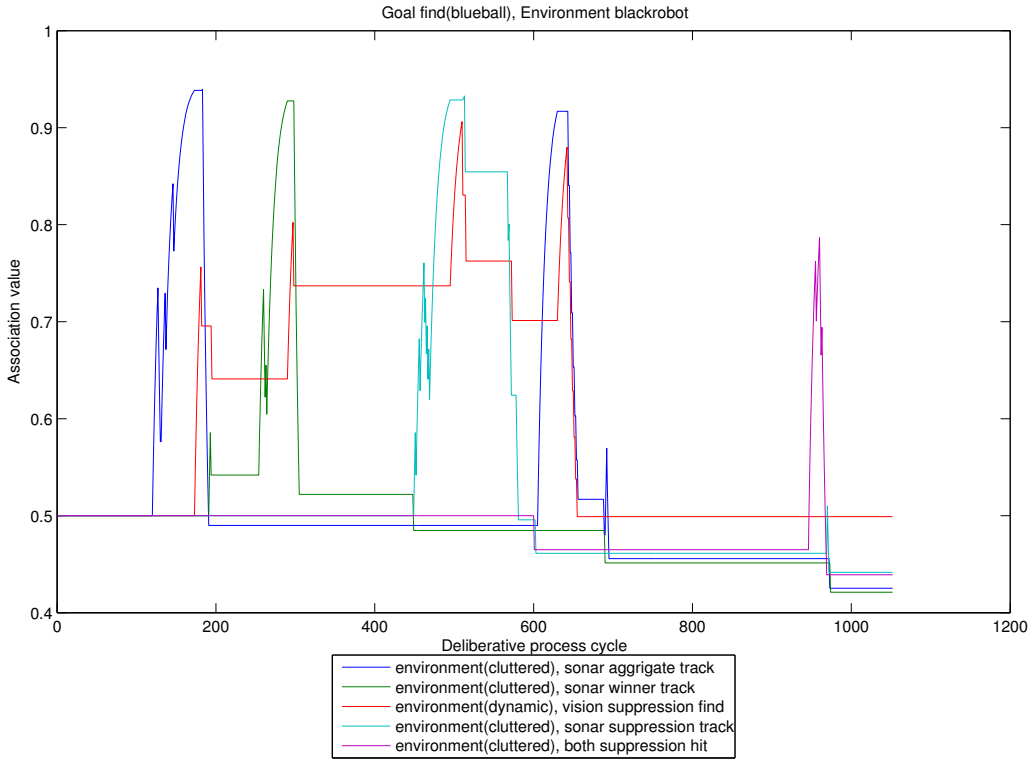

Goal find(redrobot), Environment redrobot, blueball
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(dynamic), both aggrigate find
found(redrobot), vision aggrigate find
found(redrobot), sonar aggrigate find
found(redrobot), sonar priority find

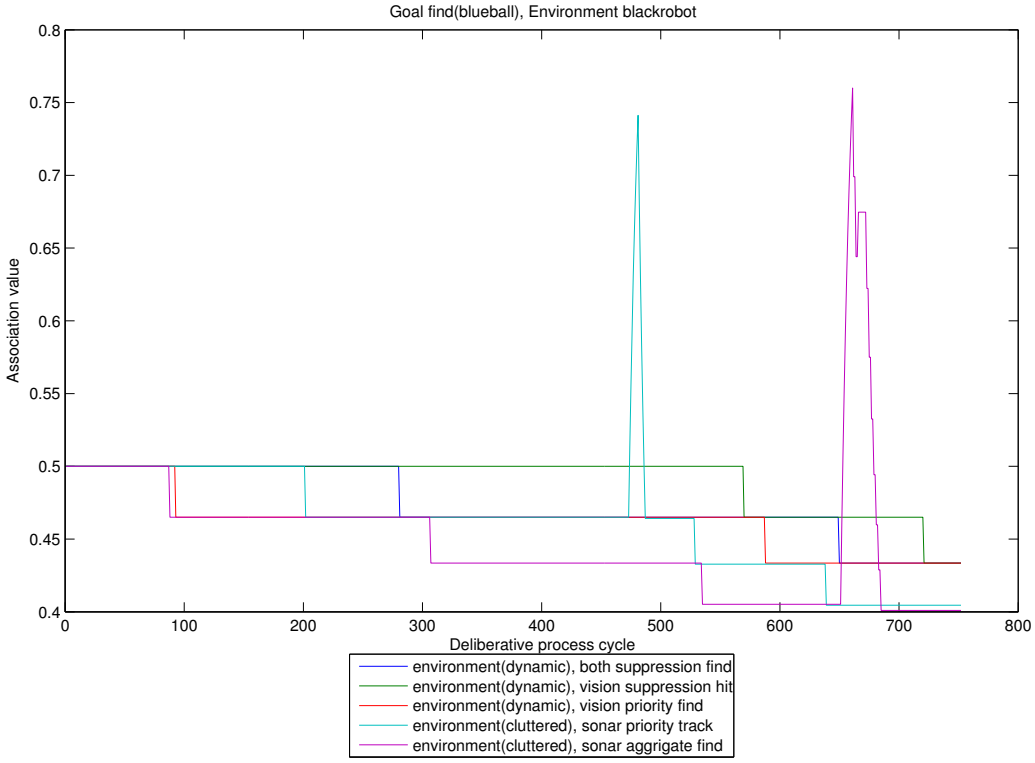

Goal find(redrobot), Environment redrobot, blueball
Association value
Deliberative process cycle
found(redrobot), vision winner find
environment(cluttered), sonar winner find
environment(dynamic), vision priority find
environment(dynamic), vision aggrigate find
found(redrobot), sonar priority find

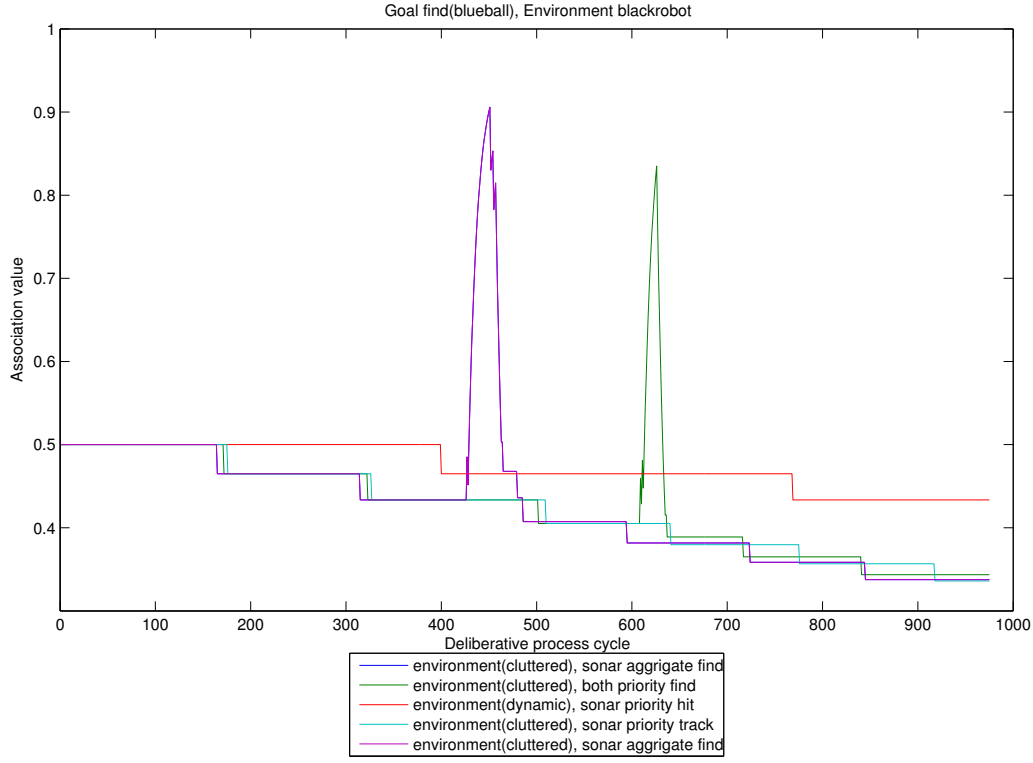

Goal find(redrobot), Environment redrobot, blueball
Association value
Deliberative process cycle
found(redrobot), sonar priority find
environment(cluttered), sonar priority find
environment(dynamic), sonar priority find
environment(dynamic), vision aggrigate hit

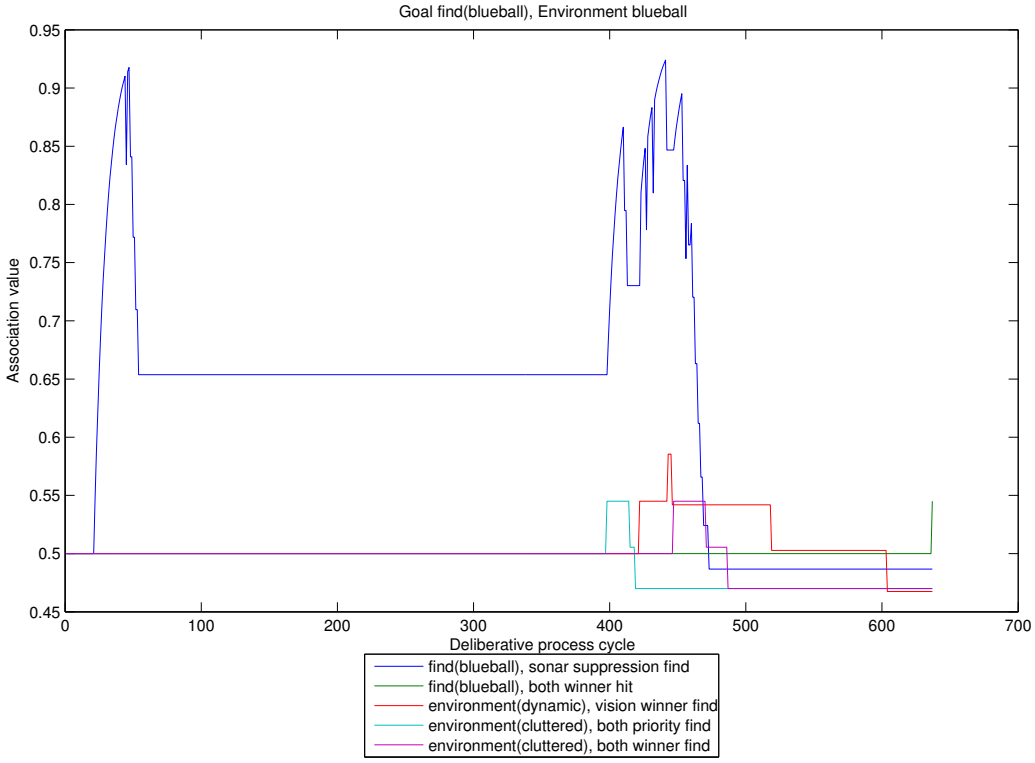
C.2.3 *find(blueball)*

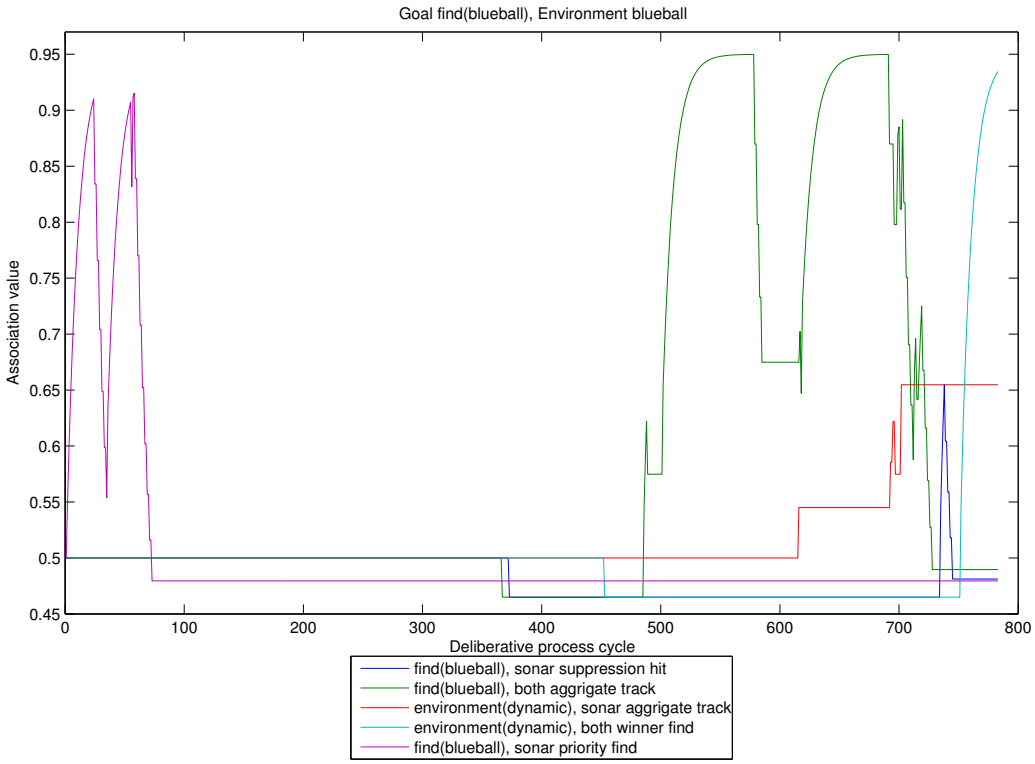


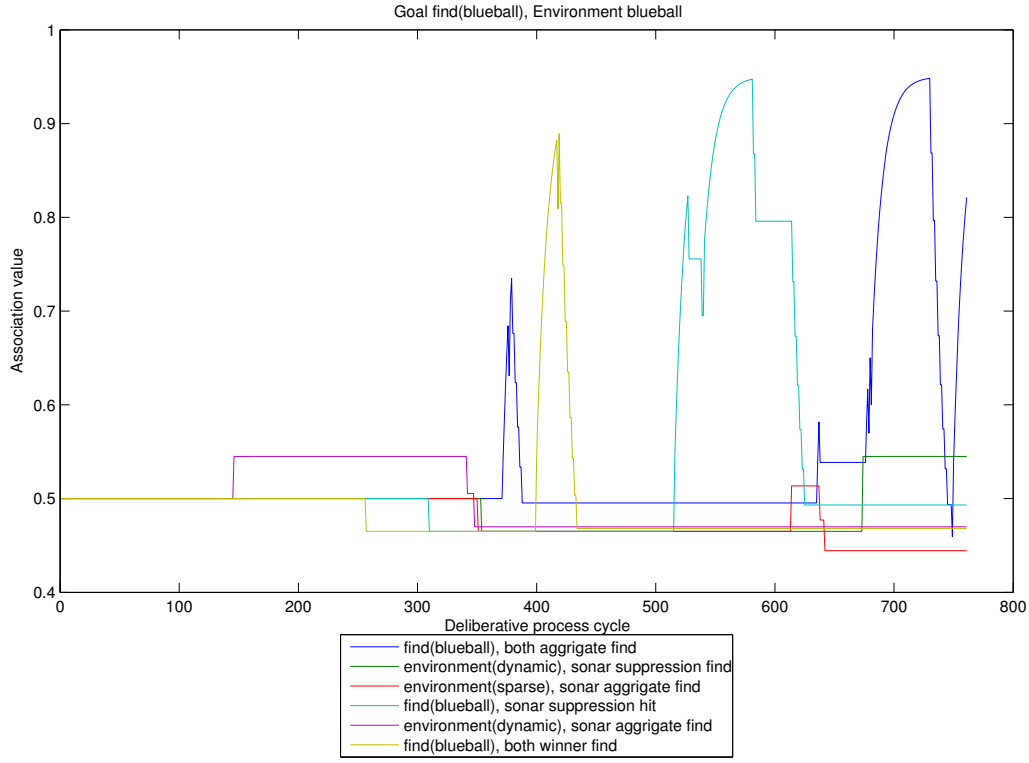


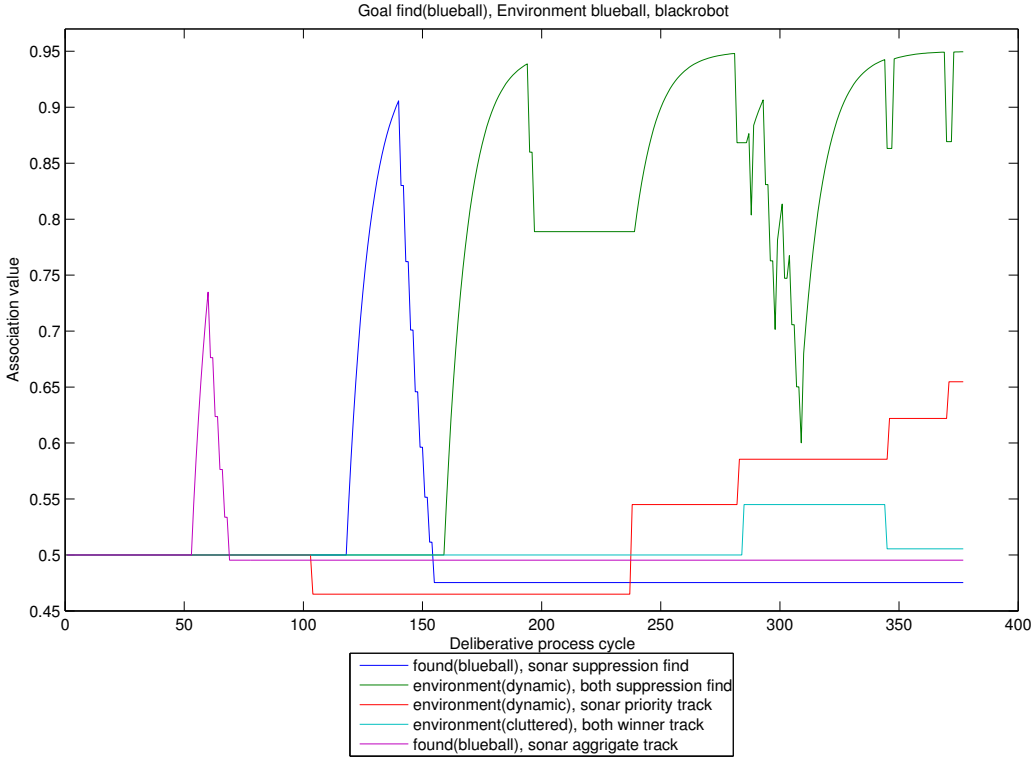

Goal find(blueball), Environment blackrobot
Association value
Deliberative process cycle
environment(dynamic), both suppression find
environment(dynamic), vision suppression hit
environment(dynamic), vision priority find
environment(cluttered), sonar priority track
environment(cluttered), sonar aggrigate find

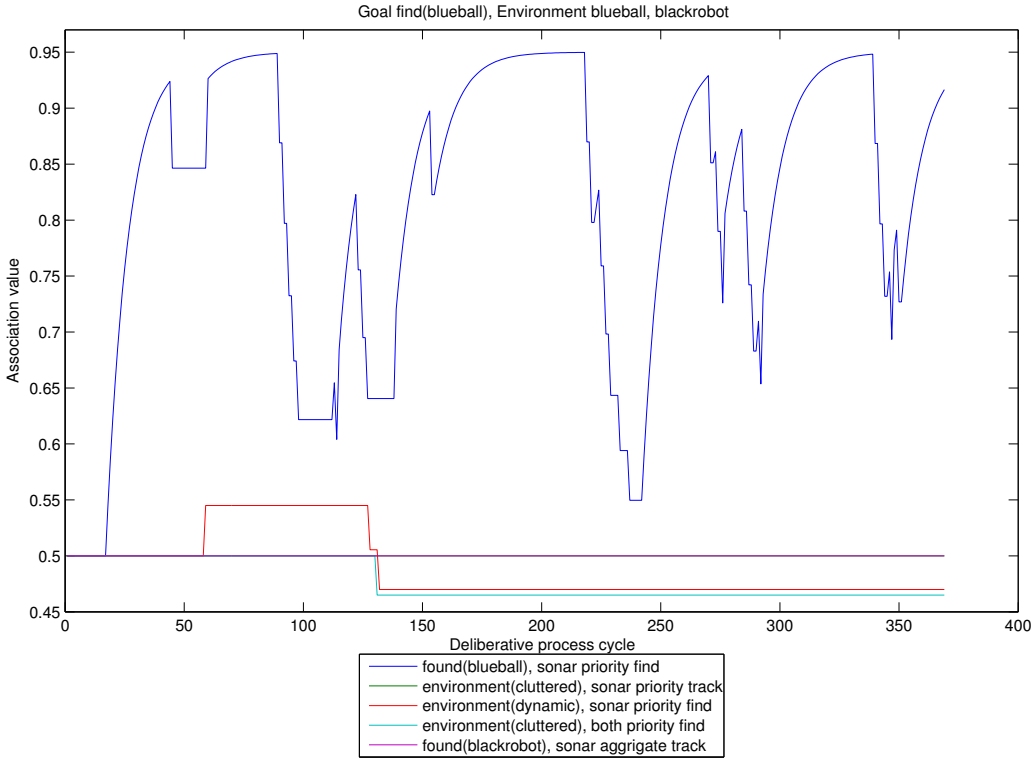

Goal find(blueball), Environment blackrobot
Association value
Deliberative process cycle
environment(cluttered), sonar aggrigate find
environment(cluttered), both priority find
environment(dynamic), sonar priority hit
environment(cluttered), sonar priority track
environment(cluttered), sonar aggrigate find

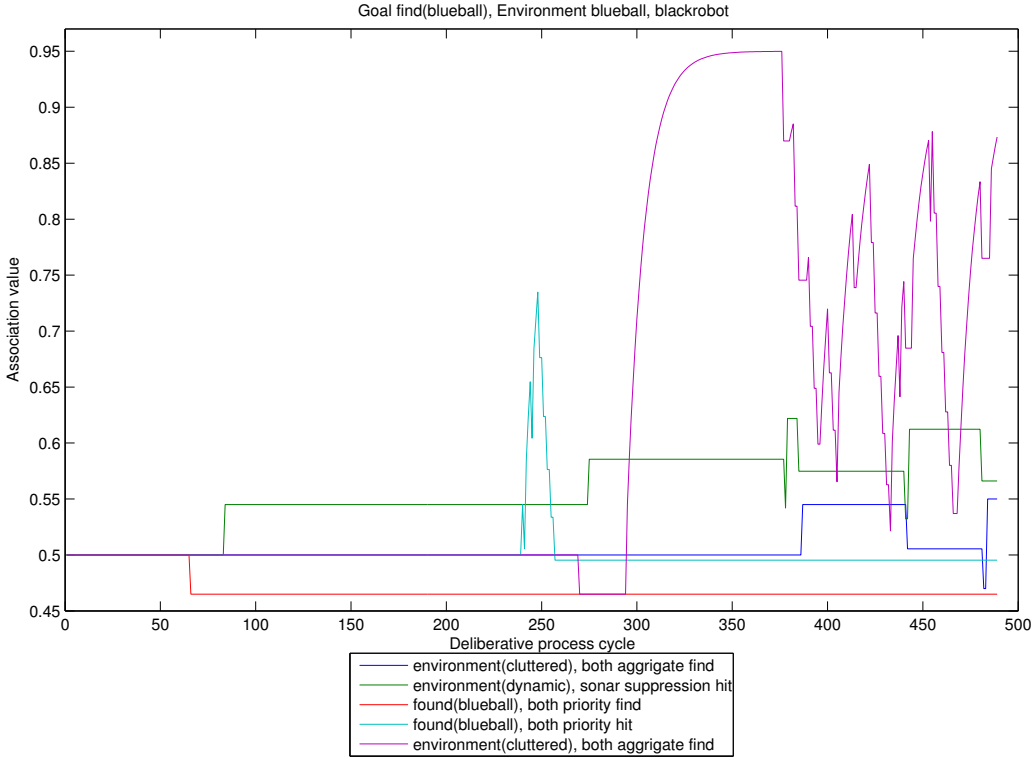

Goal find(blueball), Environment blueball
Association value
Deliberative process cycle
find(blueball), sonar suppression find
find(blueball), both winner hit
environment(dynamic), vision winner find
environment(cluttered), both priority find
environment(cluttered), both winner find

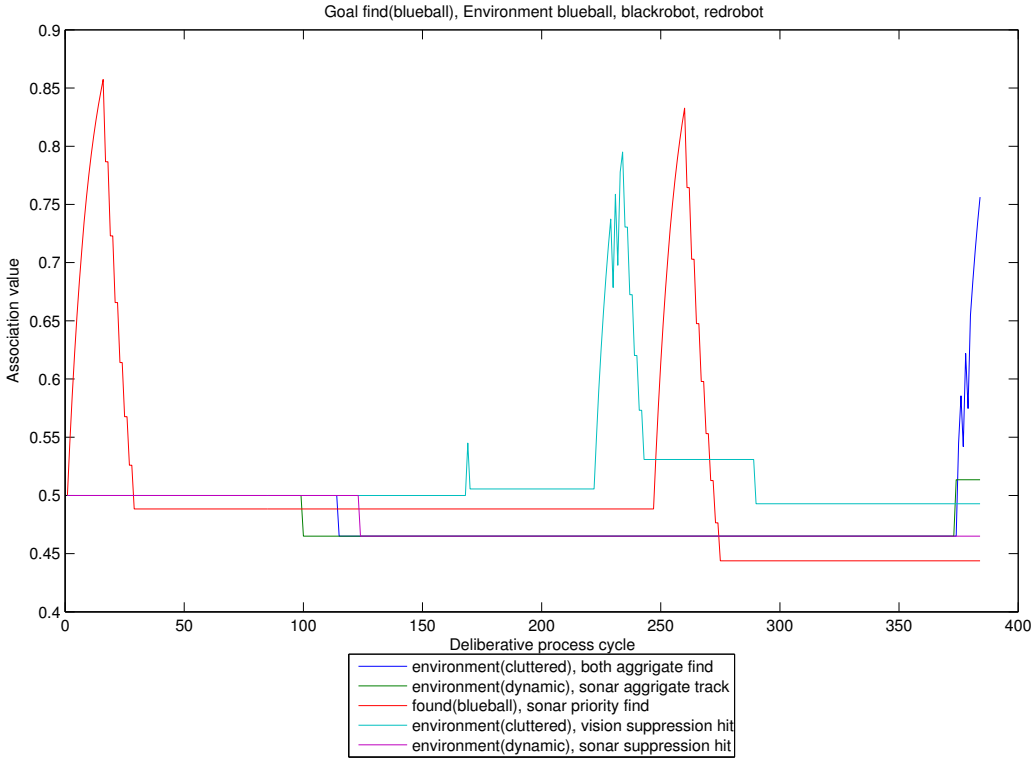

Goal find(blueball), Environment blueball
Association value
Deliberative process cycle
find(blueball), sonar suppression hit
find(blueball), both aggrigate track
environment(dynamic), sonar aggrigate track
environment(dynamic), both winner find
find(blueball), sonar priority find

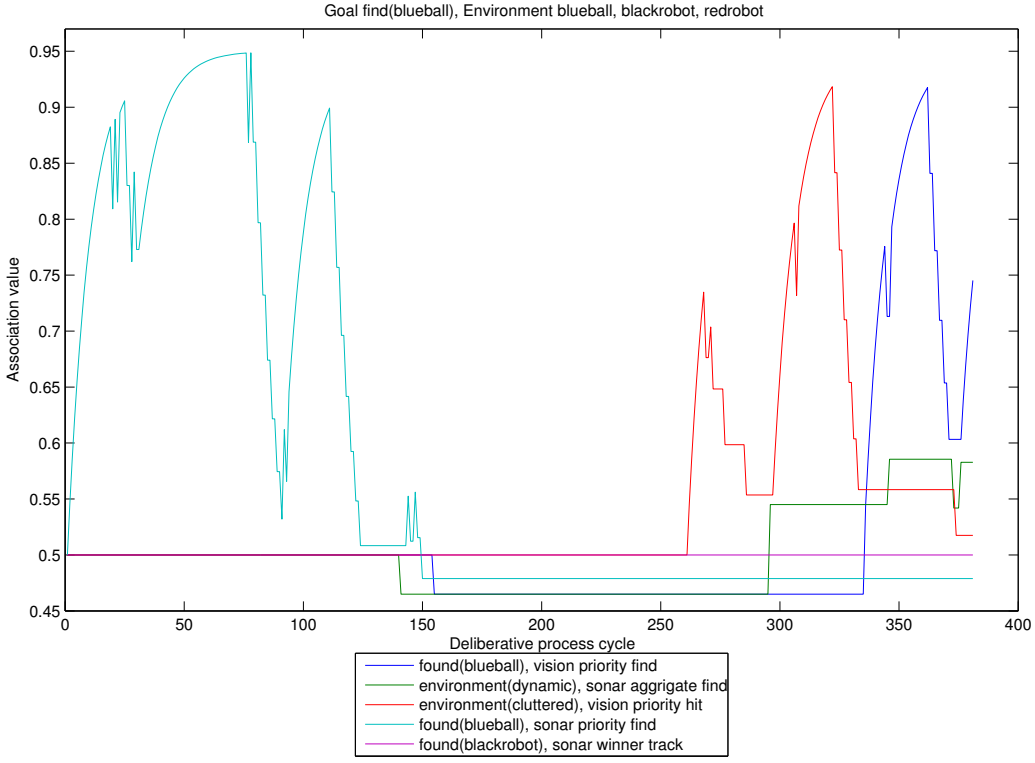

Goal find(blueball), Environment blueball
Association value
Deliberative process cycle
find(blueball), both aggrigate find
environment(dynamic), sonar suppression find
environment(sparse), sonar aggrigate find
find(blueball), sonar suppression hit
environment(dynamic), sonar aggrigate find
find(blueball), both winner find

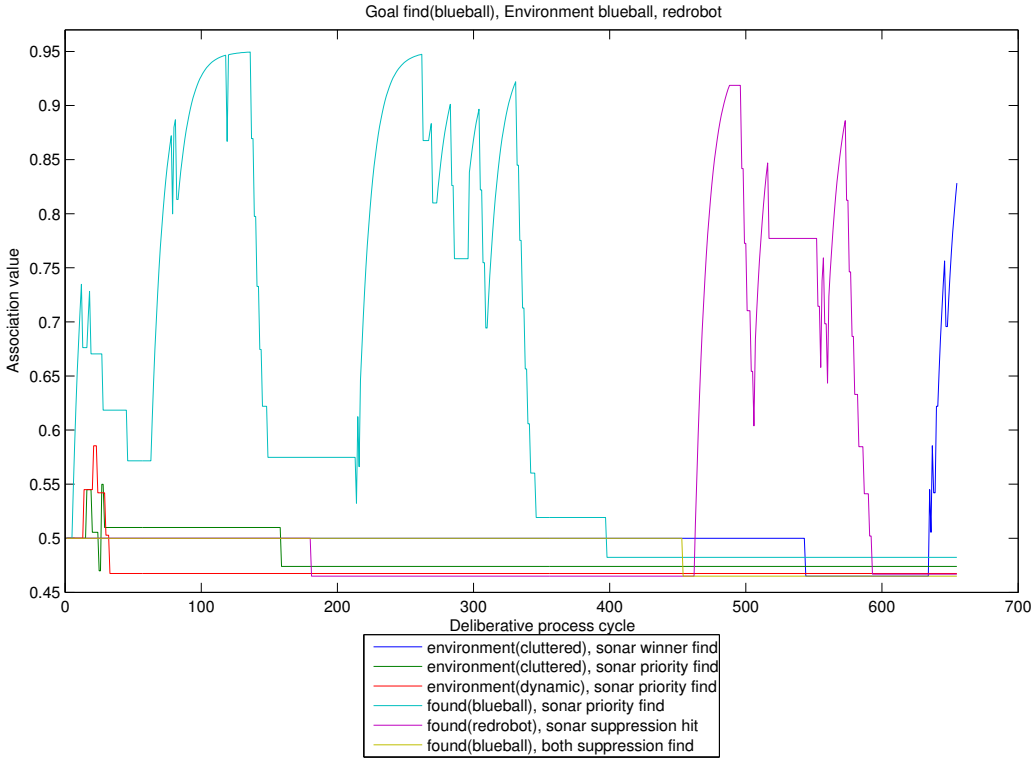

Goal find(blueball), Environment blueball, blackrobot
Association value
Deliberative process cycle
found(blueball), sonar suppression find
environment(dynamic), both suppression find
environment(dynamic), sonar priority track
environment(cluttered), both winner track
found(blueball), sonar aggrigate track

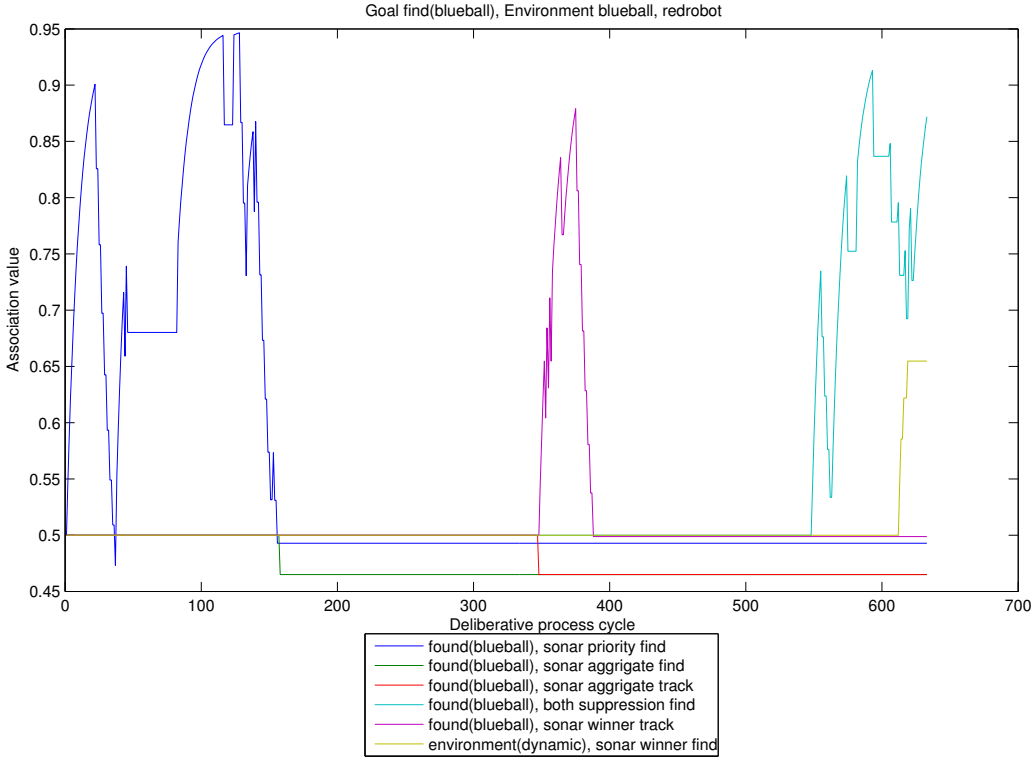

Goal find(blueball), Environment blueball, blackrobot
Association value
Deliberative process cycle
found(blueball), sonar priority find
environment(cluttered), sonar priority track
environment(dynamic), sonar priority find
environment(cluttered), both priority find
found(blackrobot), sonar aggrigate track

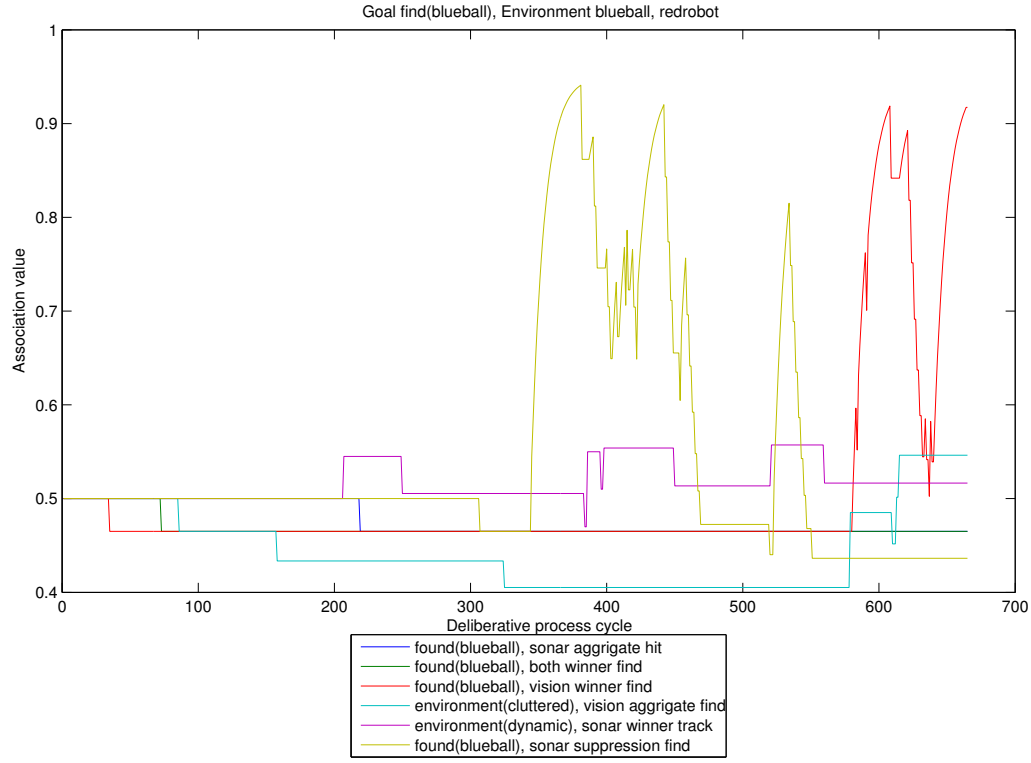

Goal find(blueball), Environment blueball, blackrobot
Association value
Deliberative process cycle
environment(cluttered), both aggrigate find
environment(dynamic), sonar suppression hit
found(blueball), both priority find
found(blueball), both priority hit
environment(cluttered), both aggrigate find

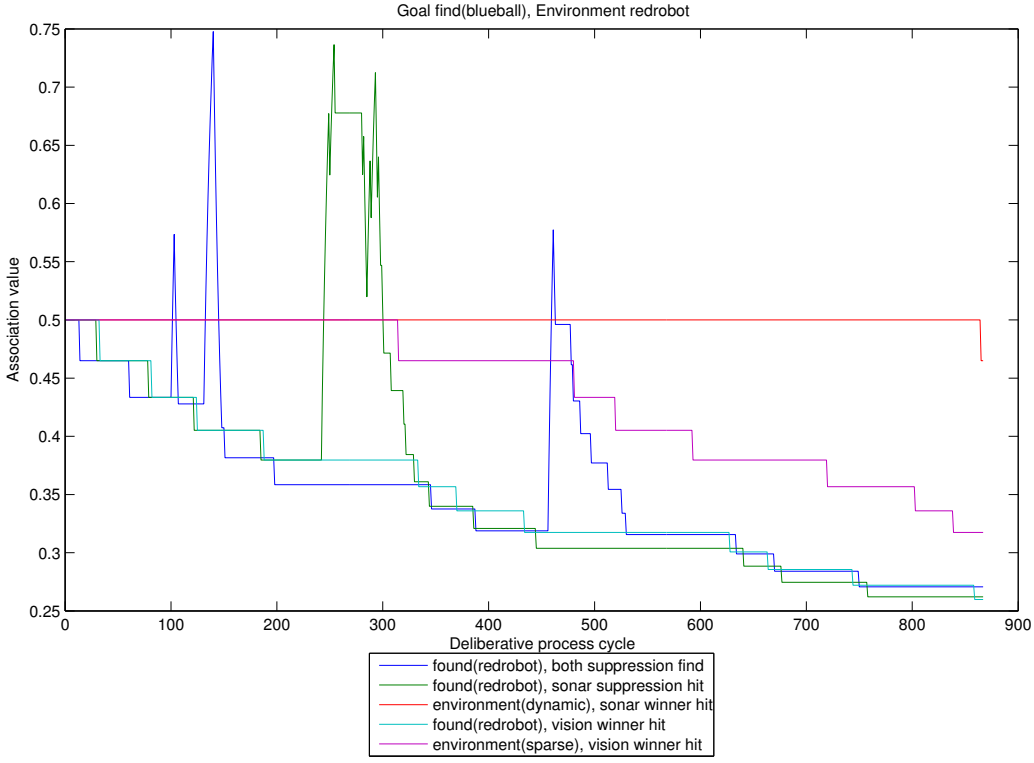

Goal find(blueball), Environment blueball, blackrobot, redrobot
Association value
Deliberative process cycle
environment(cluttered), both aggrigate find
environment(dynamic), sonar aggrigate track
found(blueball), sonar priority find
environment(cluttered), vision suppression hit
environment(dynamic), sonar suppression hit

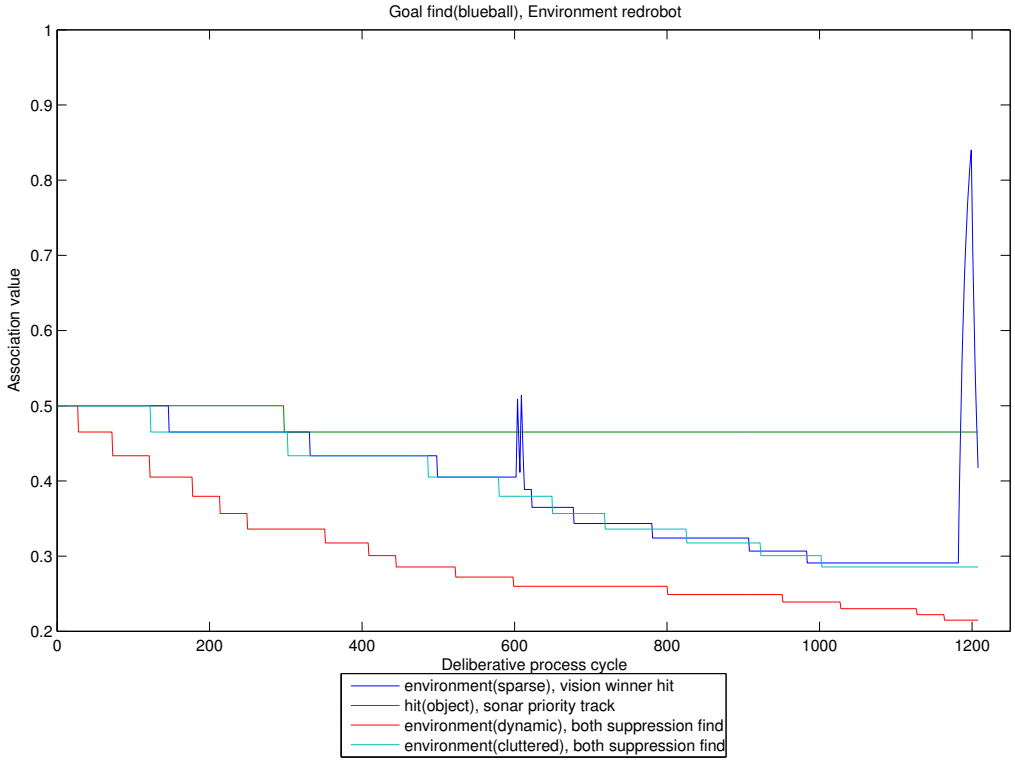

Goal find(blueball), Environment blueball, blackrobot, redrobot
Association value
Deliberative process cycle
found(blueball), vision priority find
environment(dynamic), sonar aggrigate find
environment(cluttered), vision priority hit
found(blueball), sonar priority find
found(blackrobot), sonar winner track

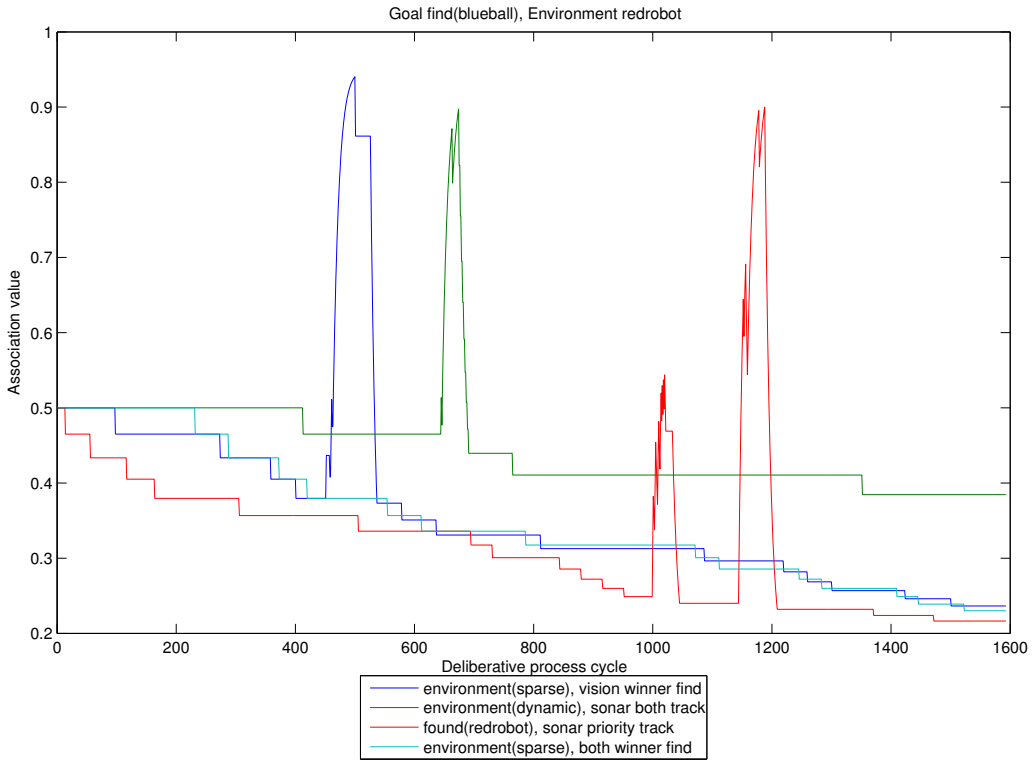

Goal find(blueball), Environment blueball, redrobot
Association value
Deliberative process cycle
environment(cluttered), sonar winner find
environment(cluttered), sonar priority find
environment(dynamic), sonar priority find
found(blueball), sonar priority find
found(redrobot), sonar suppression hit
found(blueball), both suppression find


Goal find(blueball), Environment blueball, redrobot
Association value
Deliberative process cycle
found(blueball), sonar priority find
found(blueball), sonar aggrigate find
found(blueball), sonar aggrigate track
found(blueball), both suppression find
found(blueball), sonar winner track
environment(dynamic), sonar winner find


Goal find(blueball), Environment blueball, redrobot
Association value
Deliberative process cycle
found(blueball), sonar aggrigate hit
found(blueball), both winner find
found(blueball), vision winner find
environment(cluttered), vision aggrigate find
environment(dynamic), sonar winner track
found(blueball), sonar suppression find


Goal find(blueball), Environment redrobot
Association value
Deliberative process cycle
found(redrobot), both suppression find
found(redrobot), sonar suppression hit
environment(dynamic), sonar winner hit
found(redrobot), vision winner hit
environment(sparse), vision winner hit

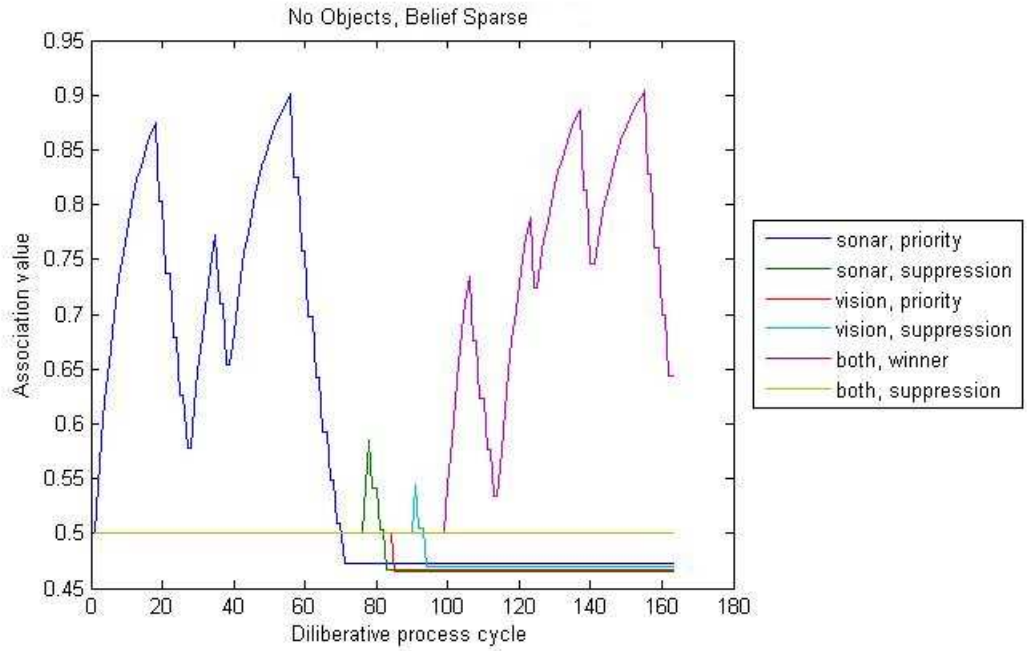

Goal find(blueball), Environment redrobot
Association value
Deliberative process cycle
environment(sparse), vision winner hit
hit(object), sonar priority track
environment(dynamic), both suppression find
environment(cluttered), both suppression find

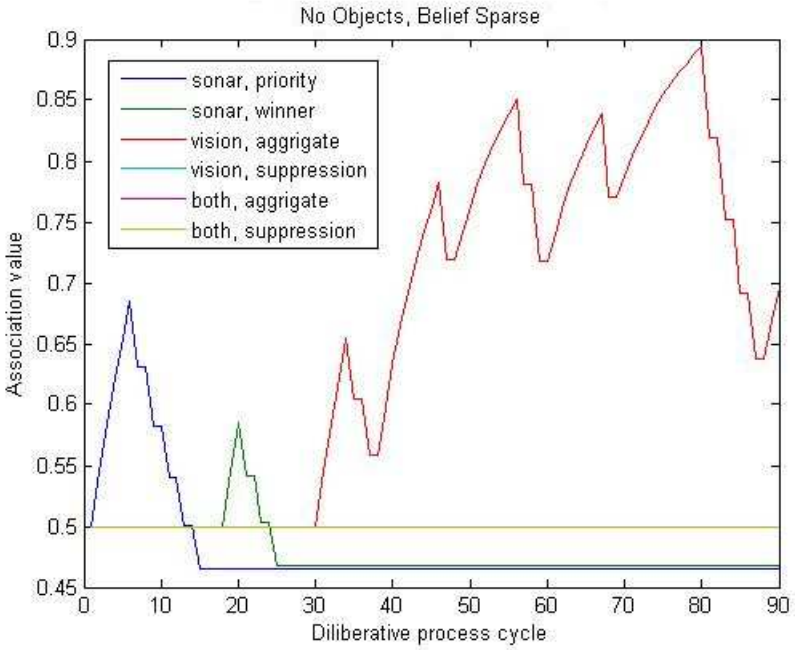

Goal find(blueball), Environment redrobot
Association value
Deliberative process cycle
environment(sparse), vision winner find
environment(dynamic), sonar both track
found(redrobot), sonar priority track
environment(sparse), both winner find

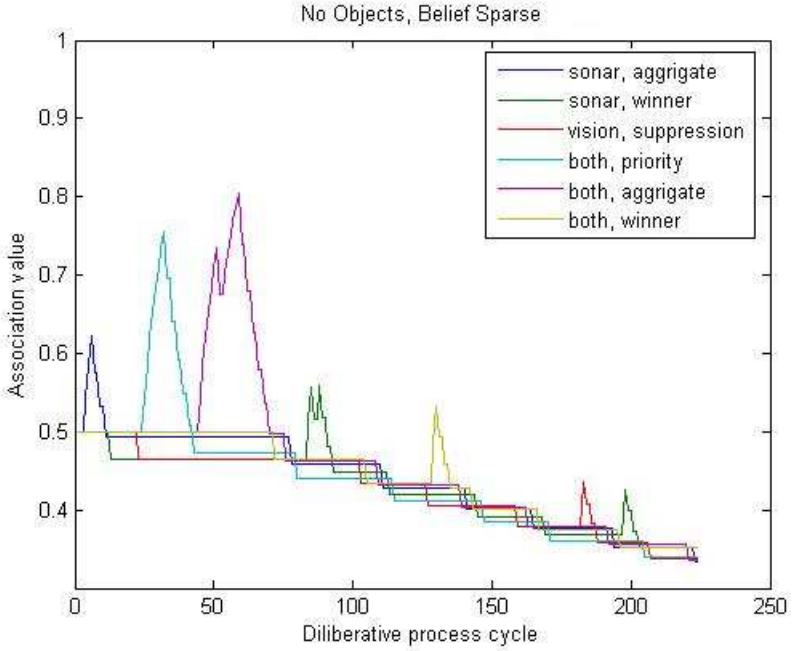
C.3 Second Learning Experimental Results

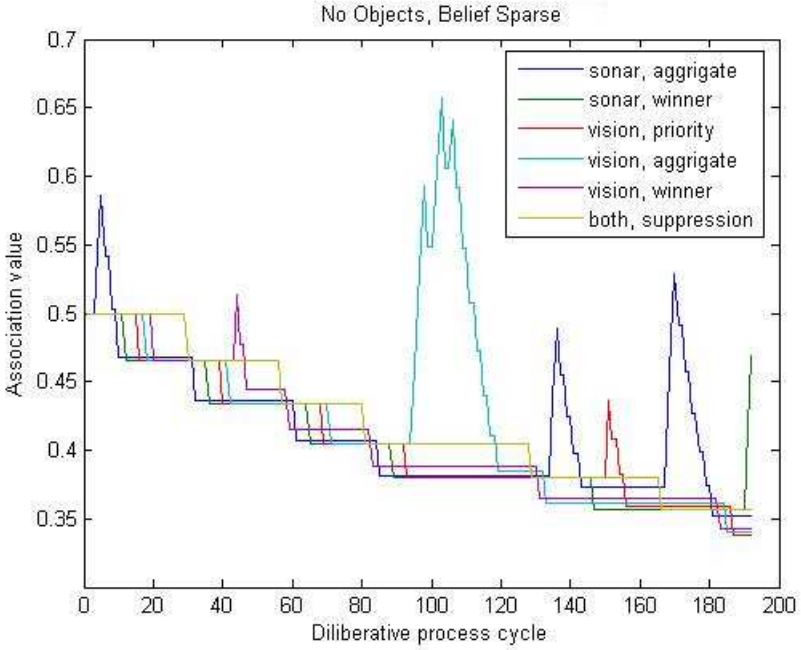
The results can be read as follows. All results refer to the goal *avoid(object)*. the results are broken down in to the number of objects present in the environment, and the nature of the environment, i.e. static or dynamic.

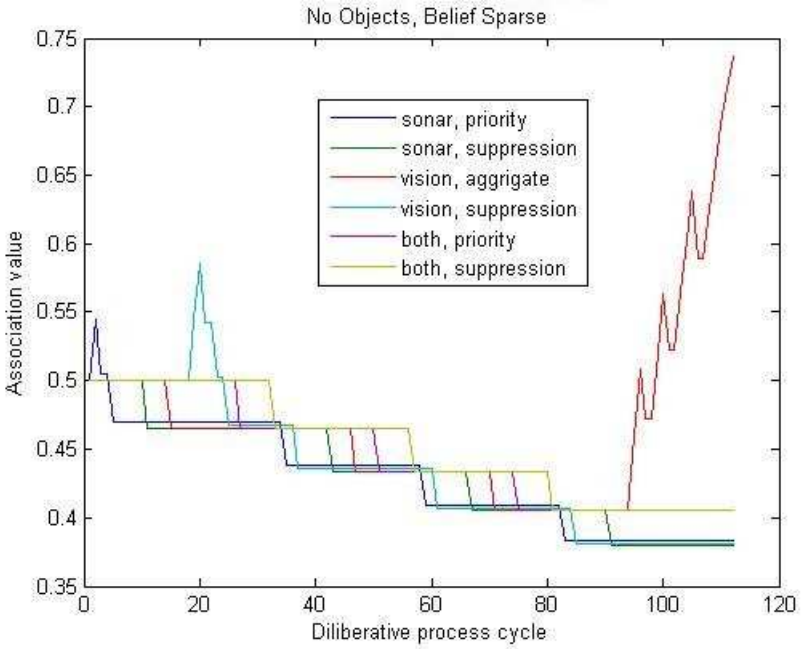
C.3.1 Zero Objects, Static Environment

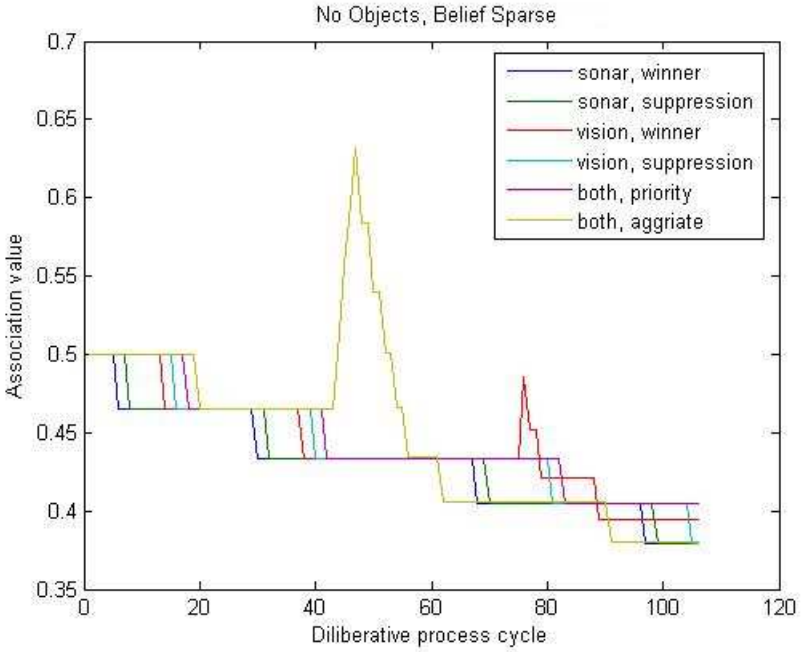


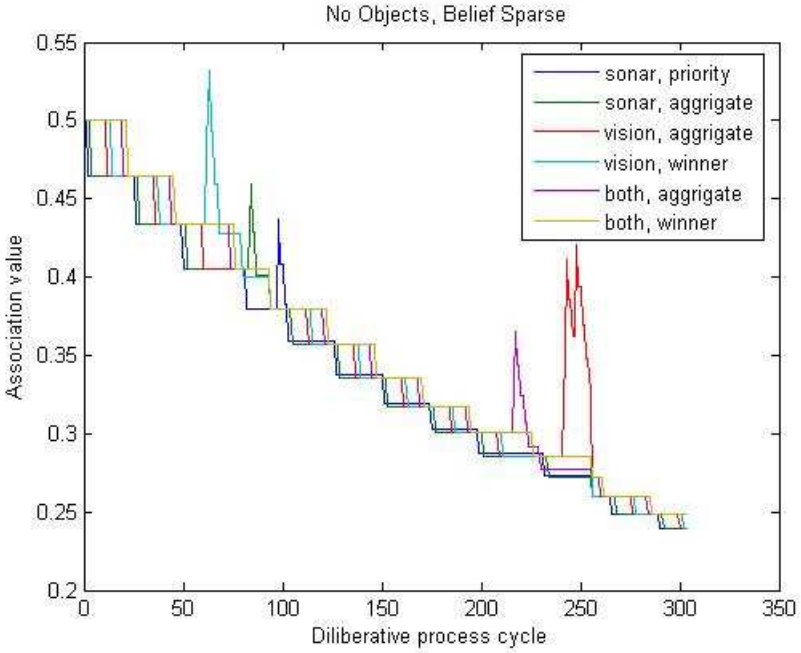

No Objects, Belief Sparse
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
10
20
30
40
50
60
70
80
90

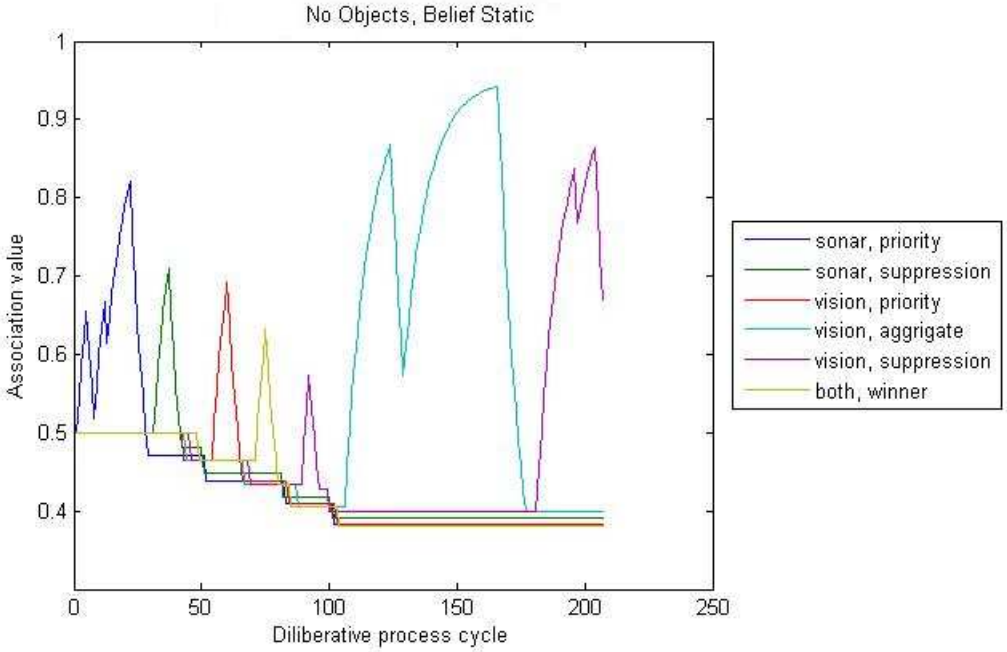

No Objects, Belief Sparse
sonar, aggrigate
sonar, winner
vision, suppression
both, priority
both, aggrigate
both, winner
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250

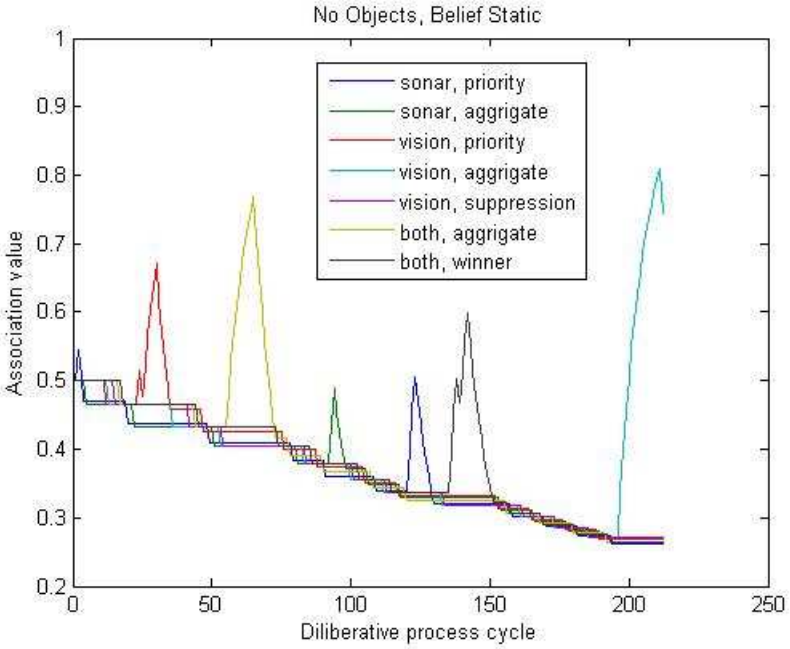

No Objects, Belief Sparse
sonar, aggrigate
sonar, winner
vision, priority
vision, aggrigate
vision, winner
both, suppression
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180
200

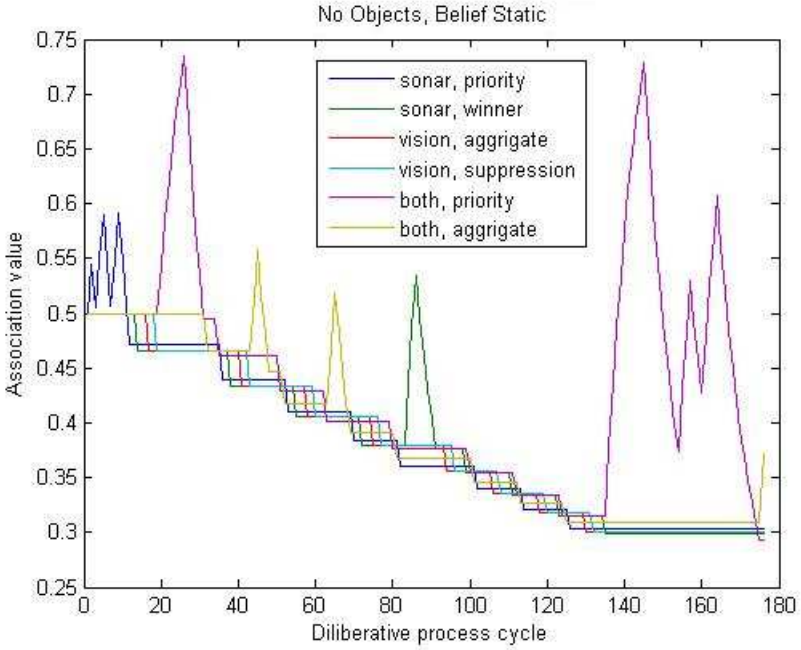

No Objects, Belief Sparse
sonar, priority
sonar, suppression
vision, aggrigate
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120

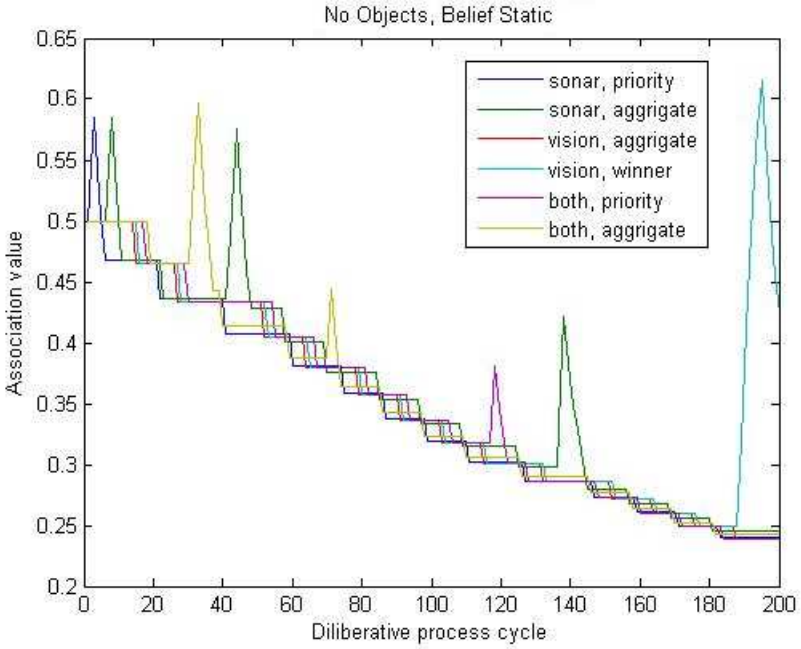

No Objects, Belief Sparse
sonar, winner
sonar, suppression
vision, winner
vision, suppression
both, priority
both, aggriate
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120

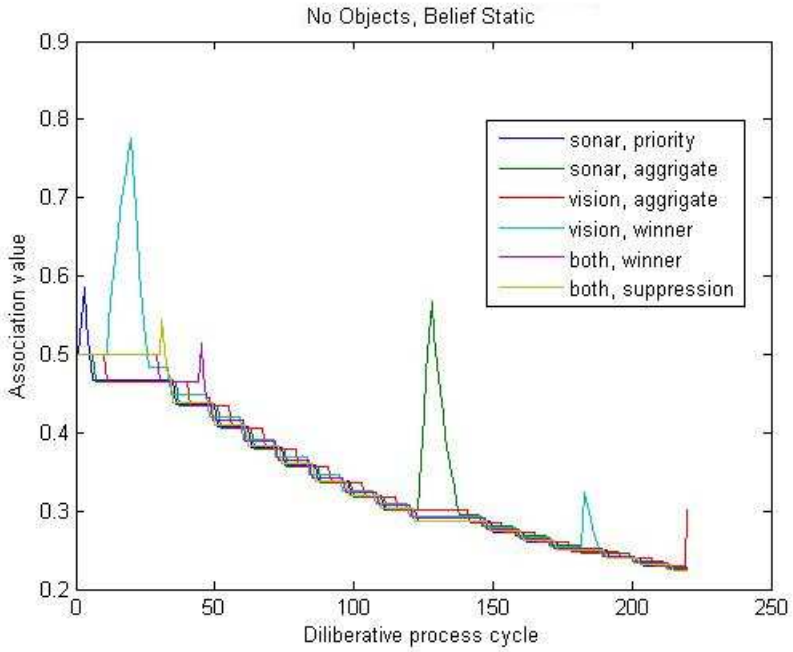

No Objects, Belief Sparse
sonar, priority
sonar, aggrigate
vision, aggrigate
vision, winner
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350


No Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, suppression
vision, priority
vision, aggrigate
vision, suppression
both, winner

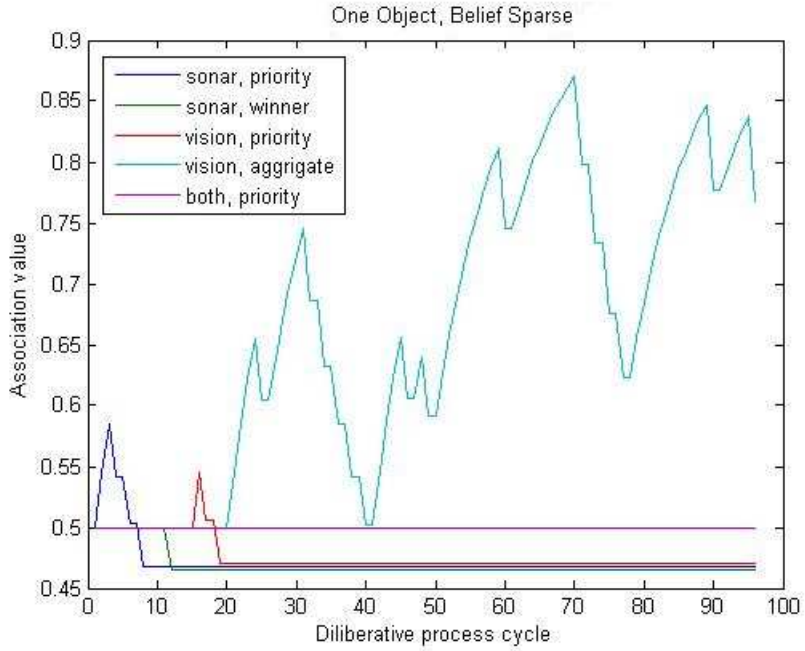

No Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, aggrigate
vision, priority
vision, aggrigate
vision, suppression
both, aggrigate
both, winner

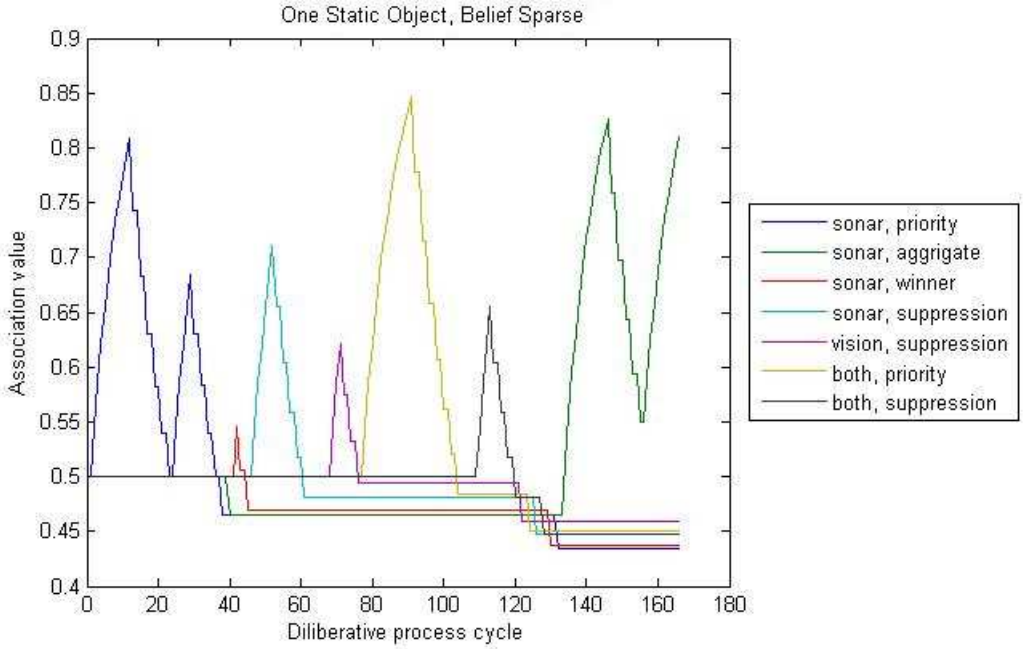

No Objects, Belief Static
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160
180

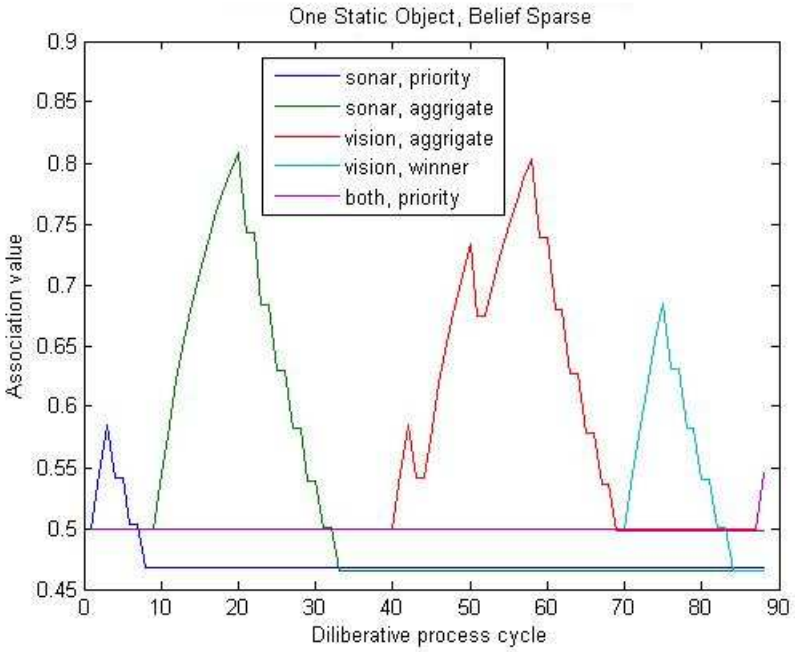

No Objects, Belief Static
sonar, priority
sonar, aggrigate
vision, aggrigate
vision, winner
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
20
40
60
80
100
120
140
160
180
200

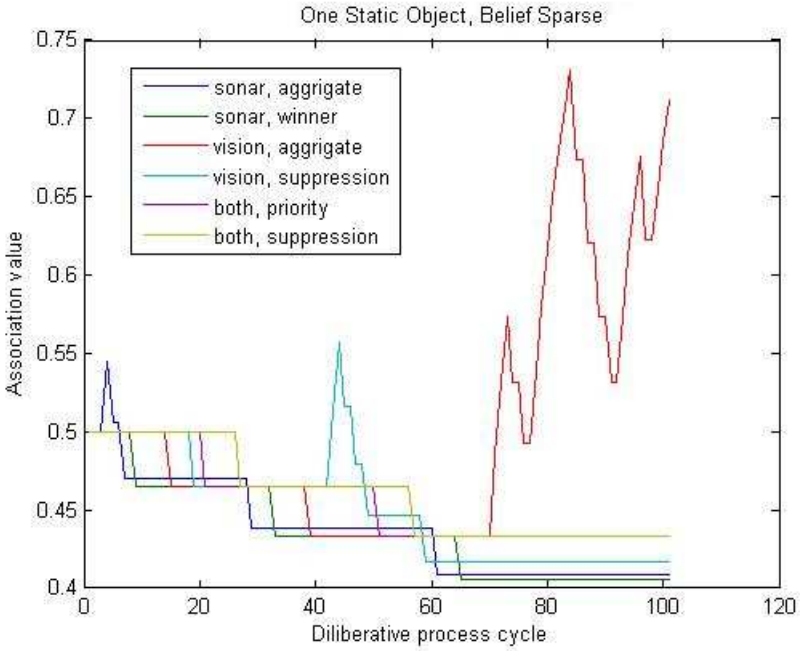

No Objects, Belief Static
Association value
Diliberative process cycle
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
50
100
150
200
250
sonar, priority
sonar, aggrigate
vision, aggrigate
vision, winner
both, winner
both, suppression

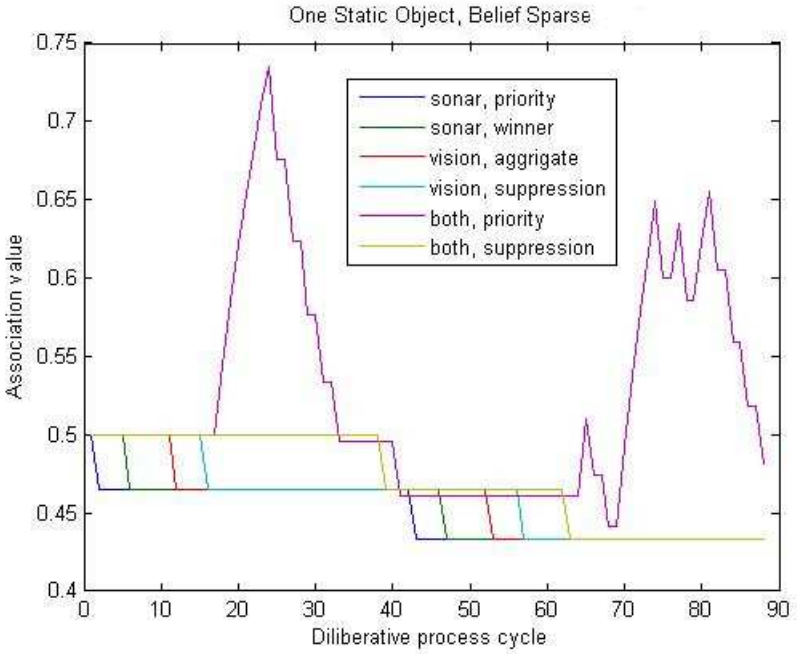
C.3.2 One Object, Static Environment

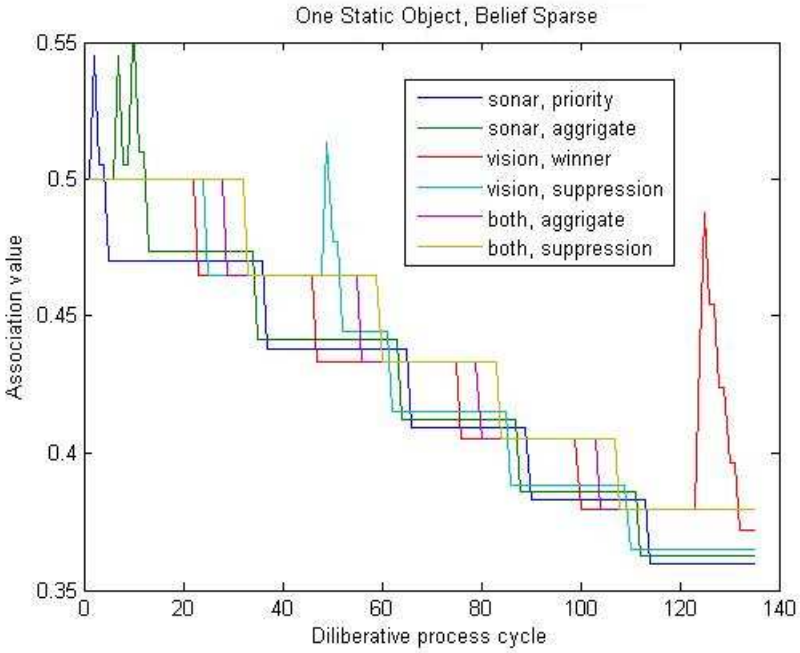


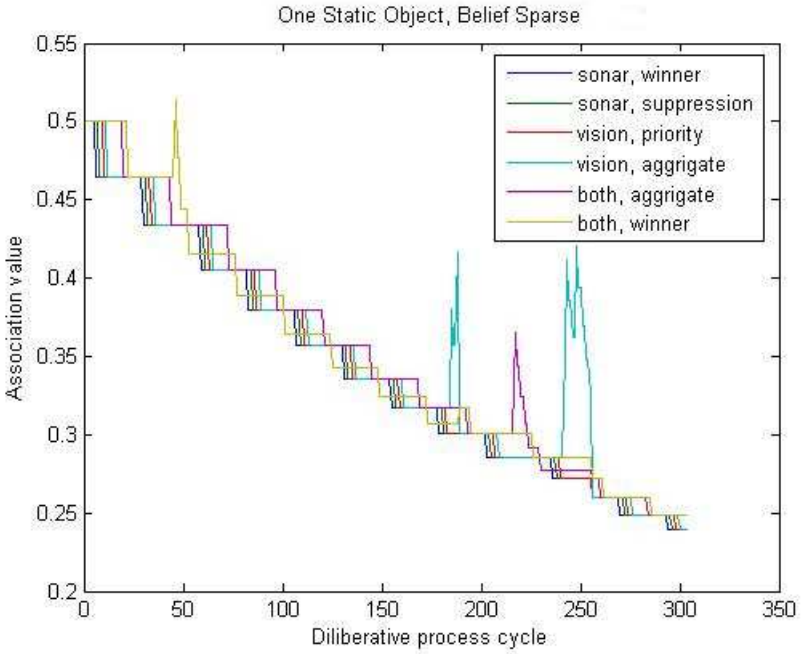


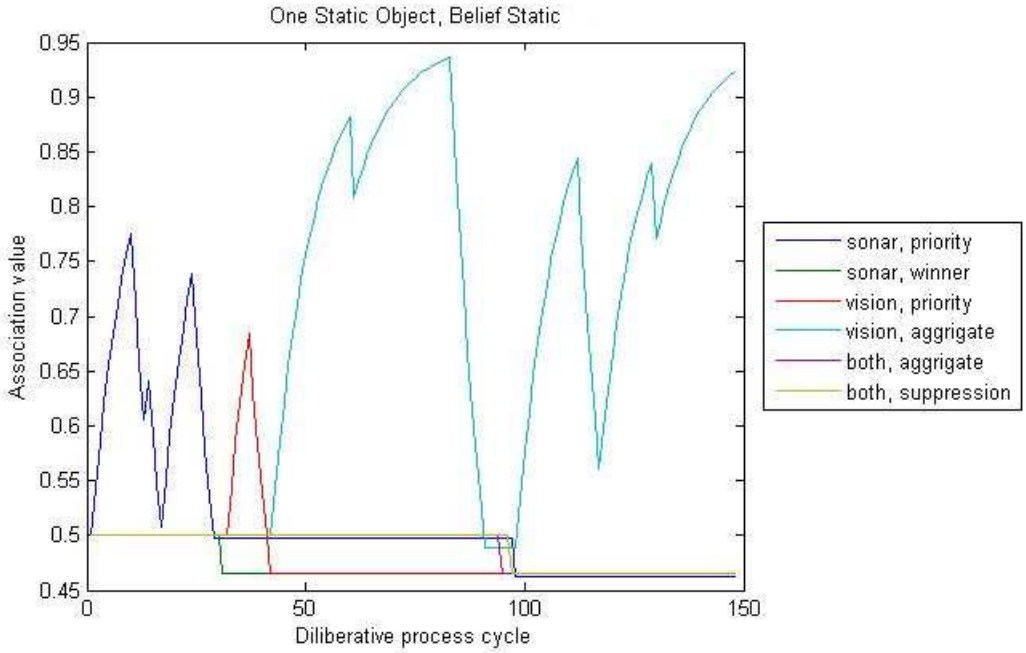

One Static Object, Belief Sparse
sonar, priority
sonar, aggrigate
vision, aggrigate
vision, winner
both, priority
Association value
Diliberative process cycle
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
10
20
30
40
50
60
70
80
90

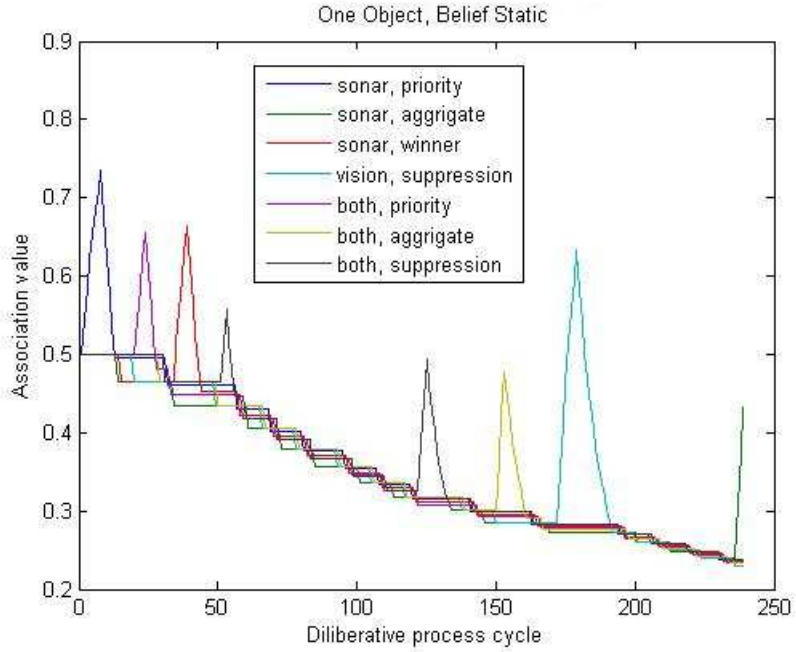

One Static Object, Belief Sparse
sonar, aggrigate
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
20
40
60
80
100
120

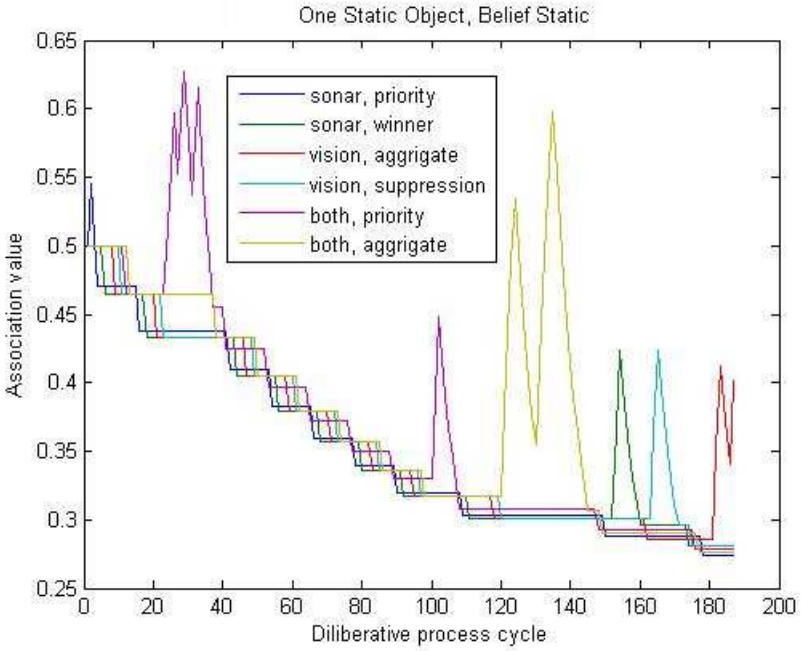

One Static Object, Belief Sparse
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
10
20
30
40
50
60
70
80
90

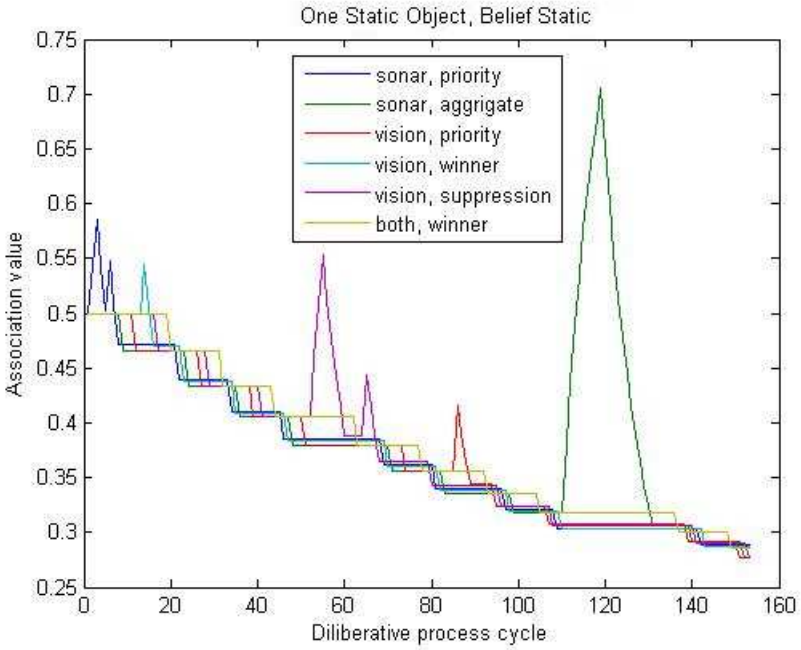

One Static Object, Belief Sparse
sonar, priority
sonar, aggrigate
vision, winner
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140

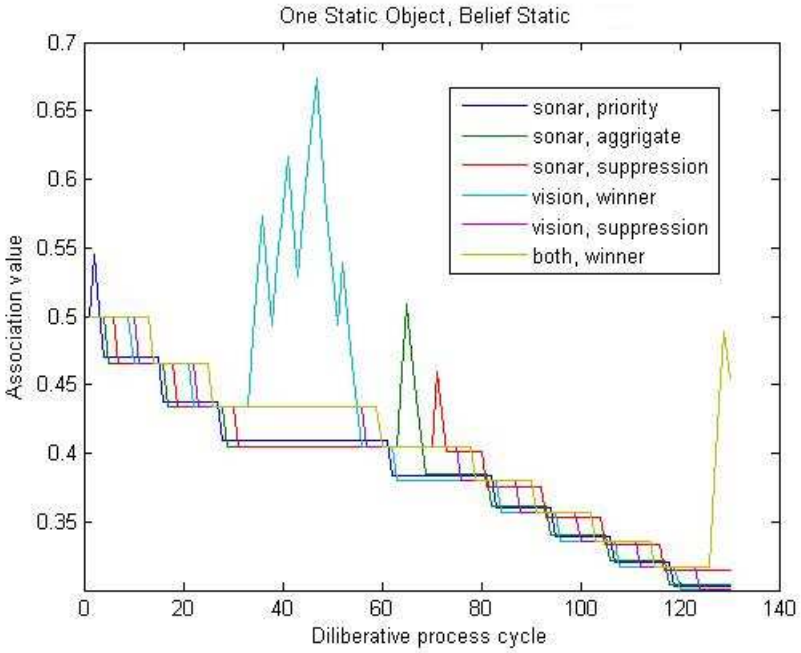

One Static Object, Belief Sparse
Association value
Diliberative process cycle
sonar, winner
sonar, suppression
vision, priority
vision, aggrigate
both, aggrigate
both, winner
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350

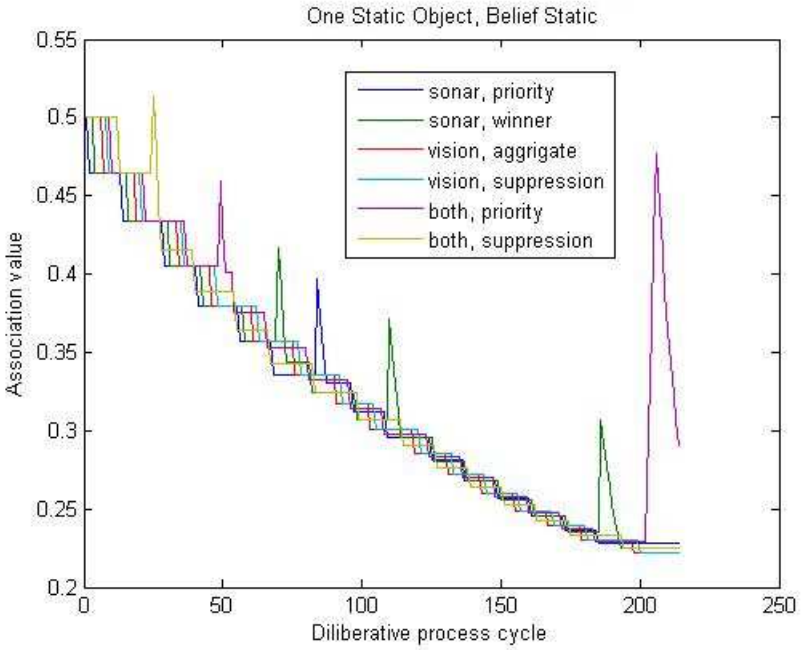

One Static Object, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, winner
vision, priority
vision, aggrigate
both, aggrigate
both, suppression
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
50
100
150

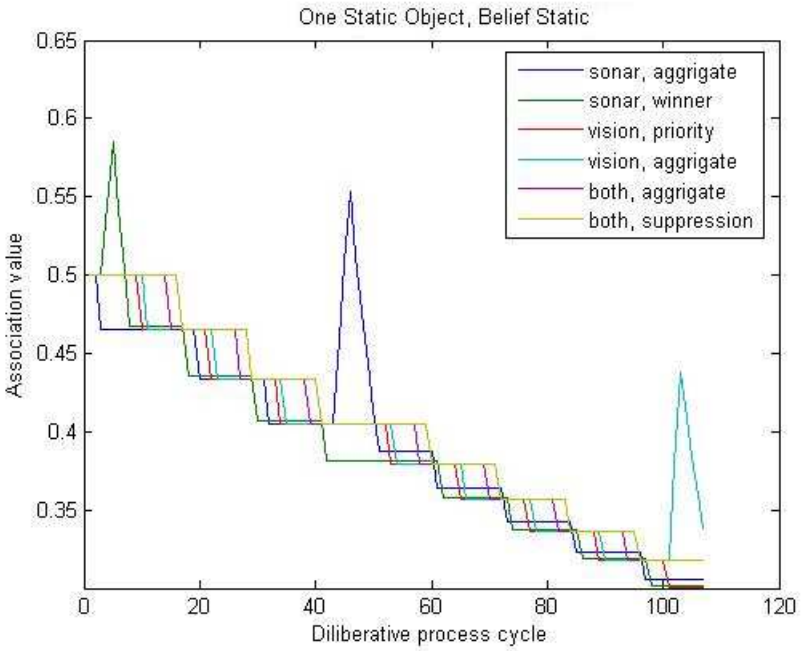

One Object, Belief Static
sonar, priority
sonar, aggrigate
sonar, winner
vision, suppression
both, priority
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
50
100
150
200
250


One Static Object, Belief Static
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160
180
200

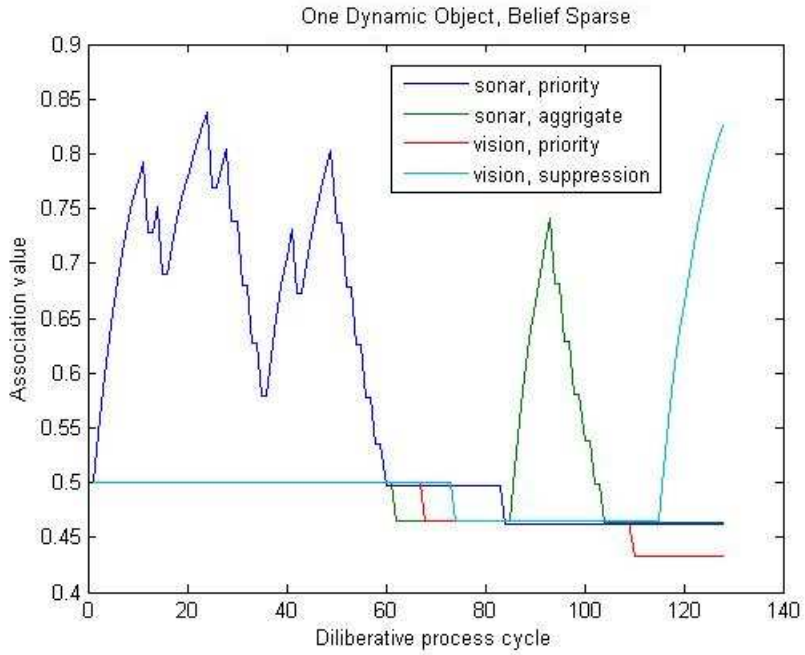

One Static Object, Belief Static
sonar, priority
sonar, aggrigate
vision, priority
vision, winner
vision, suppression
both, winner
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160

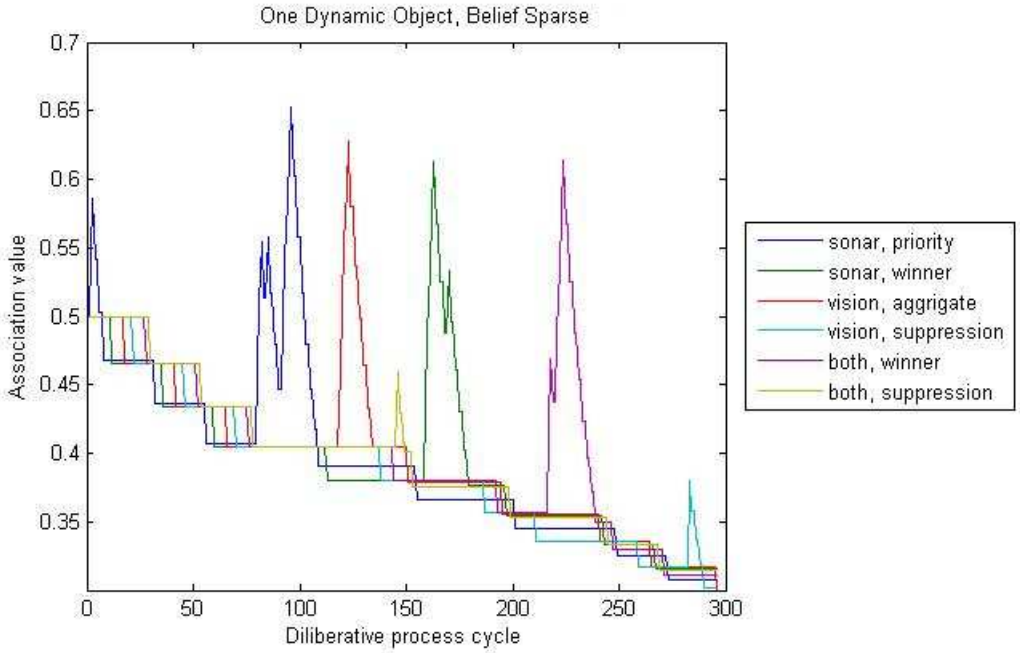

One Static Object, Belief Static
sonar, priority
sonar, aggrigate
sonar, suppression
vision, winner
vision, suppression
both, winner
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140

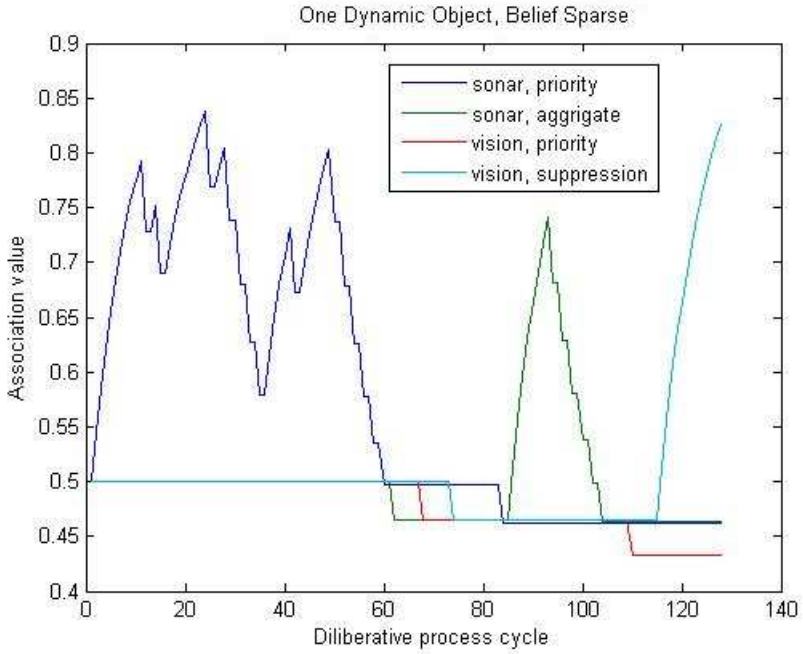

One Static Object, Belief Static
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

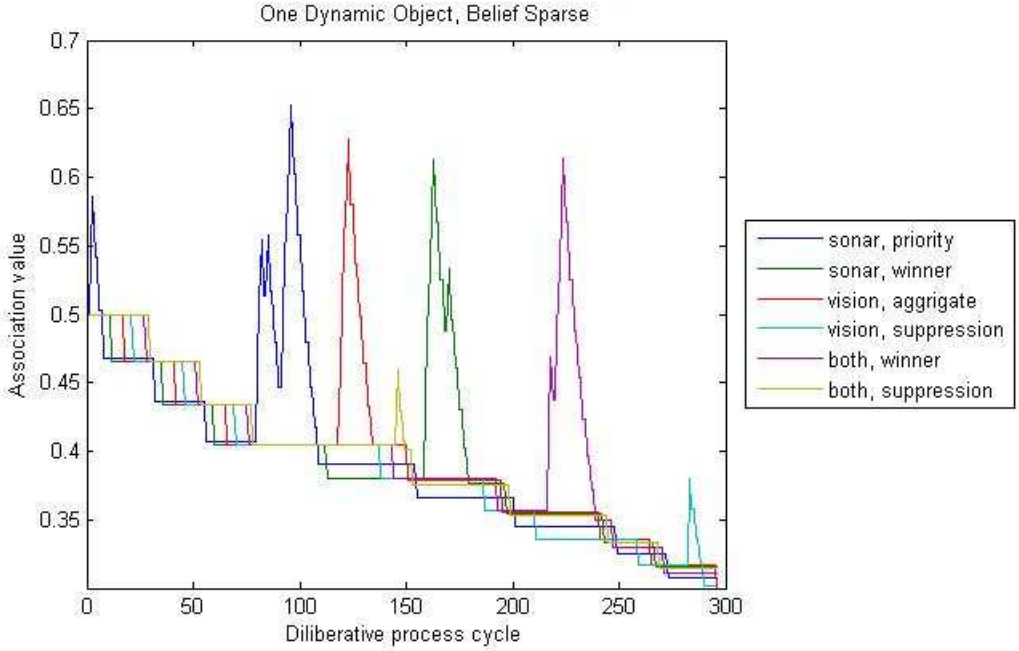

One Static Object, Belief Static
sonar, aggrigate
sonar, winner
vision, priority
vision, aggrigate
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120

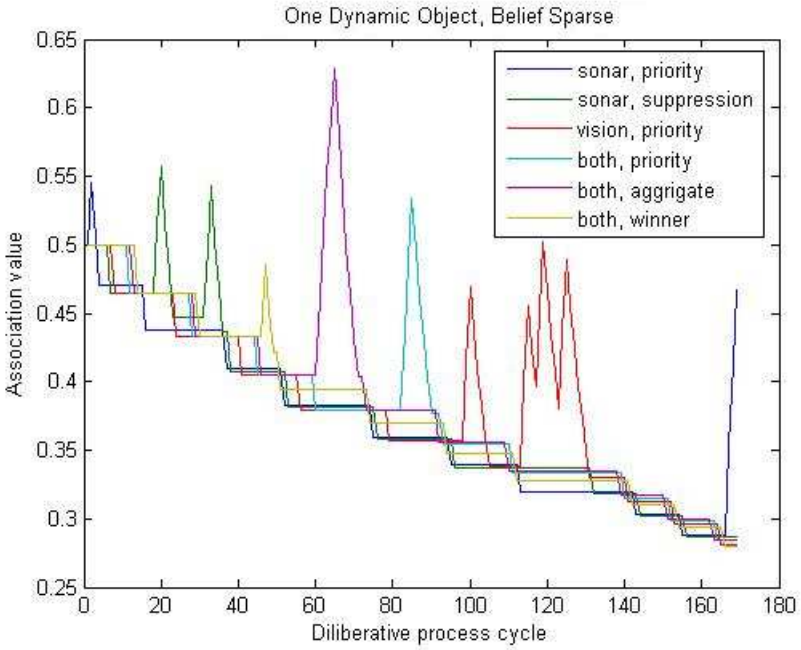
C.3.3 One Object, Dynamic Environment

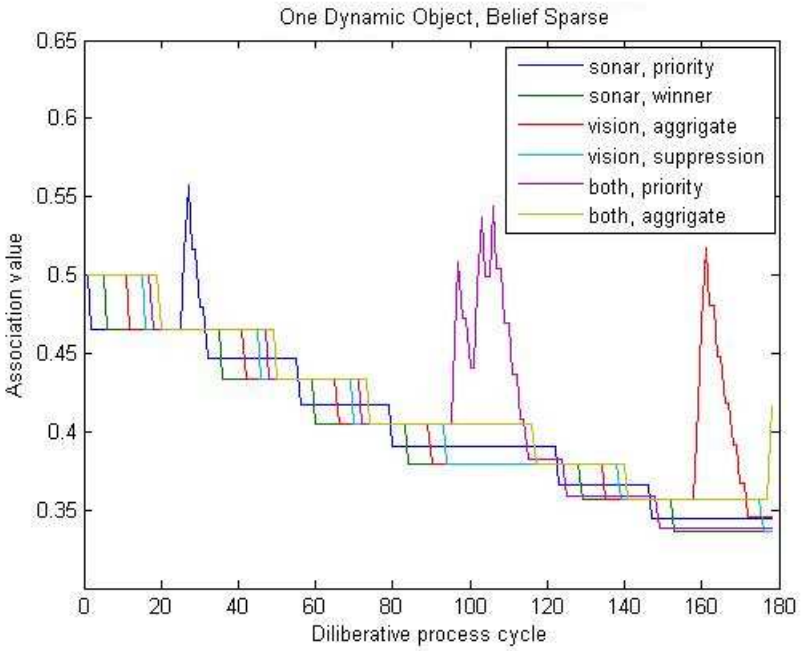


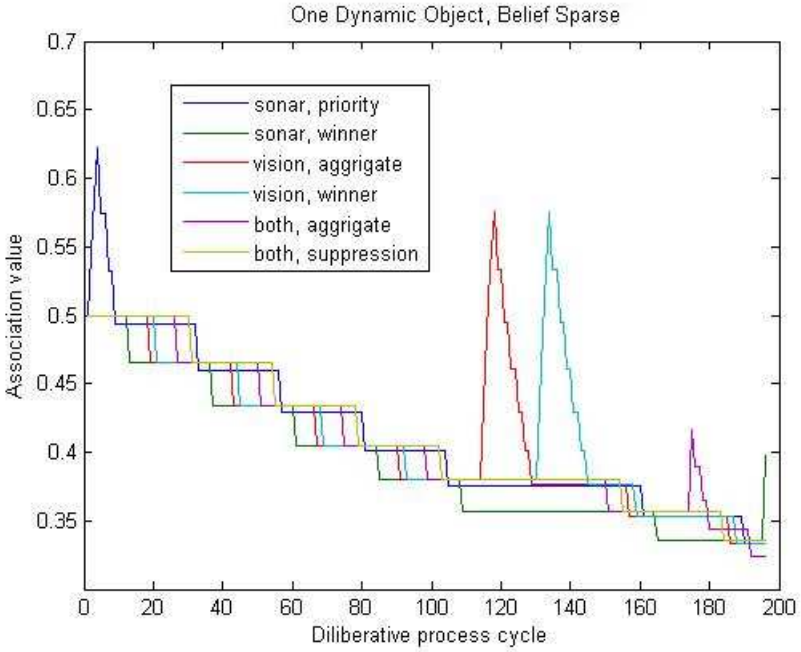


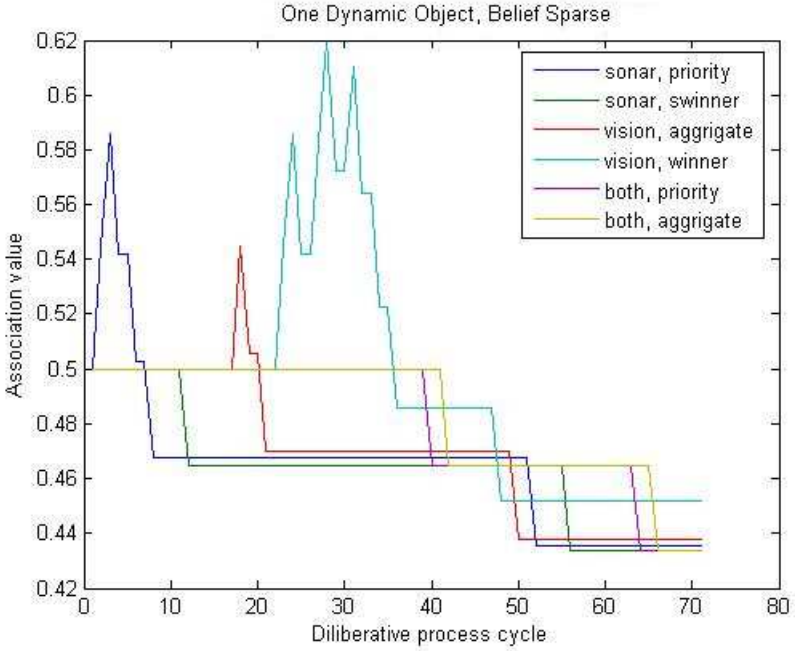

One Dynamic Object, Belief Sparse
sonar, priority
sonar, aggrigate
vision, priority
vision, suppression
Association value
Diliberative process cycle
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
20
40
60
80
100
120
140

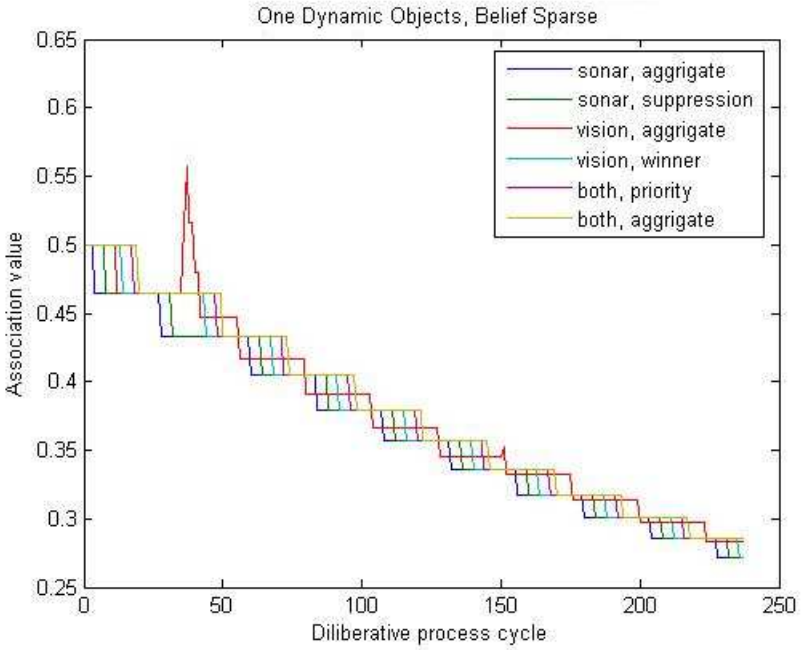

One Dynamic Object, Belief Sparse
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, winner
both, suppression
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
50
100
150
200
250
300

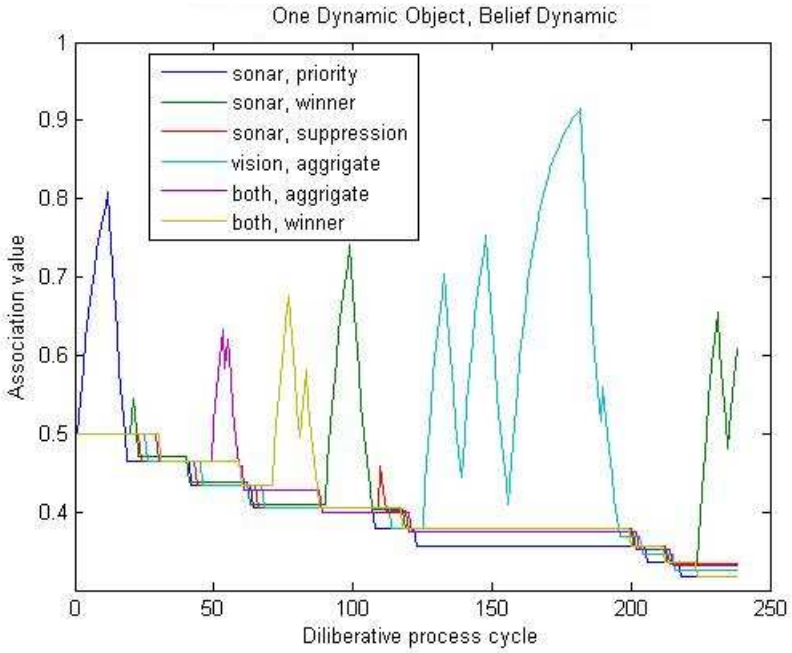

One Dynamic Object, Belief Sparse
sonar, priority
sonar, suppression
vision, priority
both, priority
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160
180

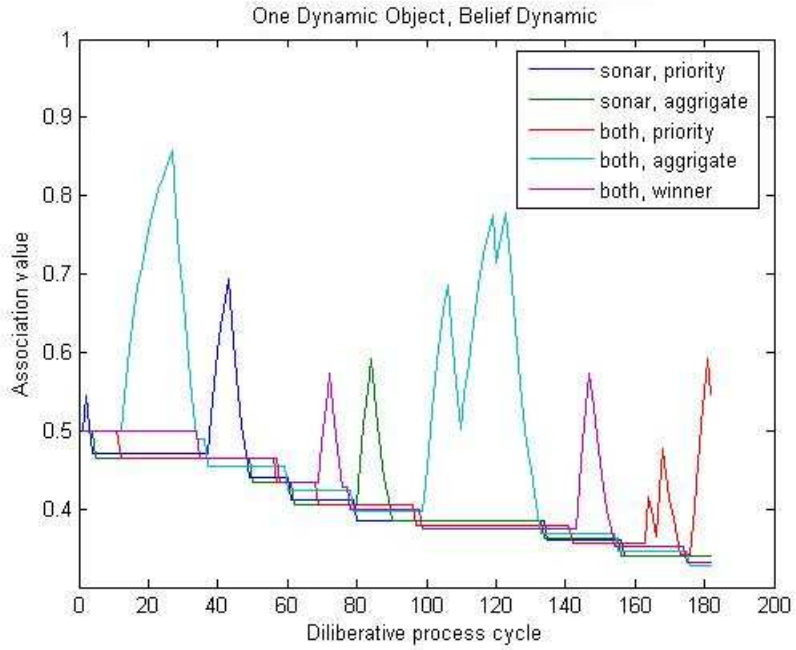

One Dynamic Object, Belief Sparse
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180

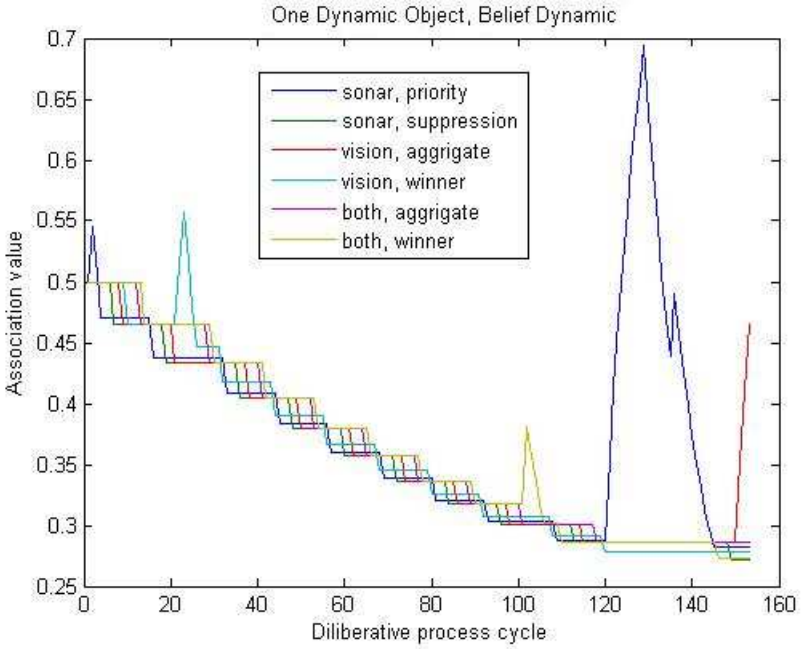

One Dynamic Object, Belief Sparse
sonar, priority
sonar, winner
vision, aggrigate
vision, winner
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180
200

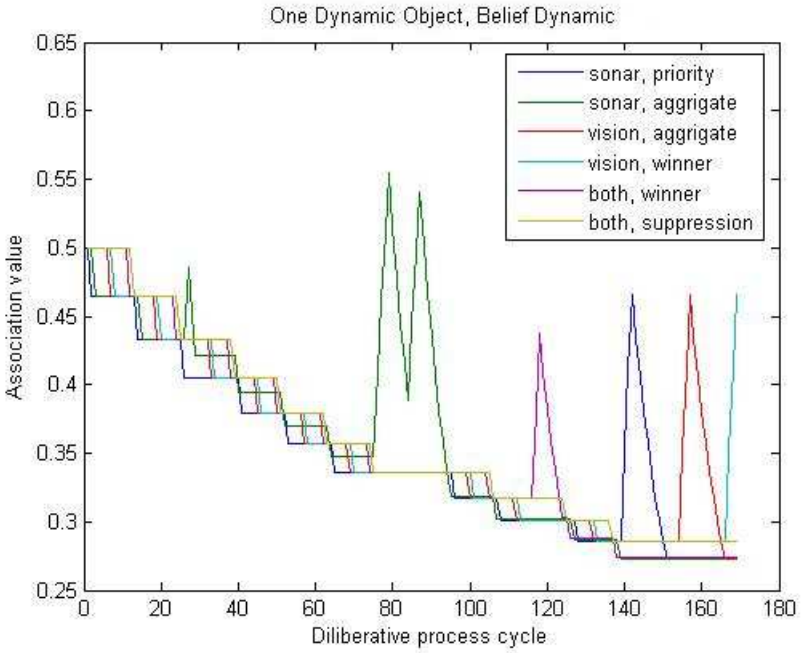

One Dynamic Object, Belief Sparse
sonar, priority
sonar, swinner
vision, aggrigate
vision, winner
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.62
0.6
0.58
0.56
0.54
0.52
0.5
0.48
0.46
0.44
0.42
0
10
20
30
40
50
60
70
80

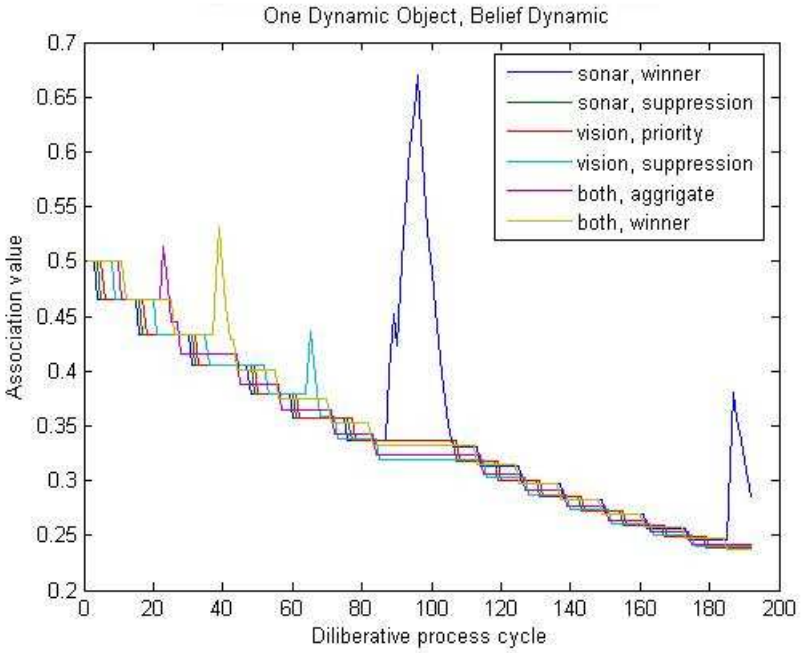

One Dynamic Objects, Belief Sparse
sonar, aggrigate
sonar, suppression
vision, aggrigate
vision, winner
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
50
100
150
200
250

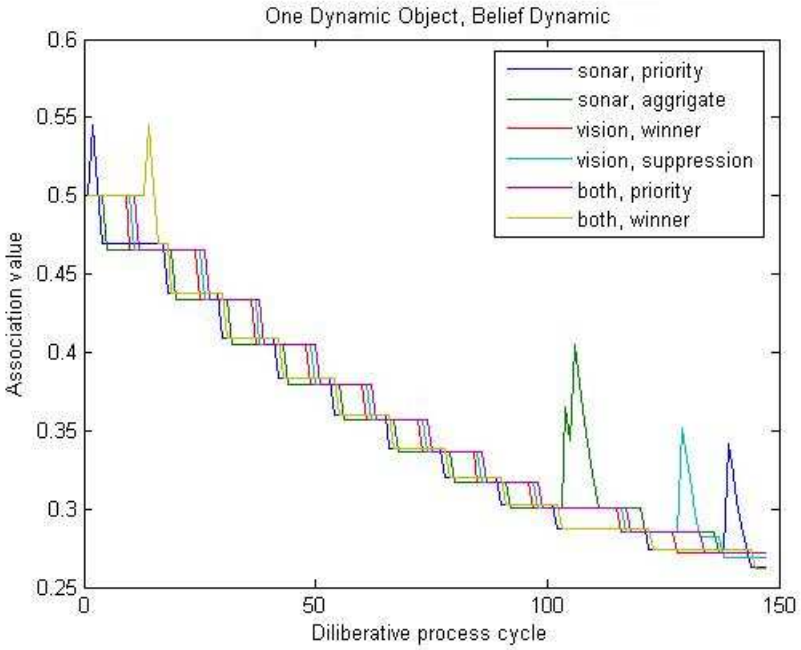

One Dynamic Object, Belief Dynamic
sonar, priority
sonar, winner
sonar, suppression
vision, aggrigate
both, aggrigate
both, winner
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250

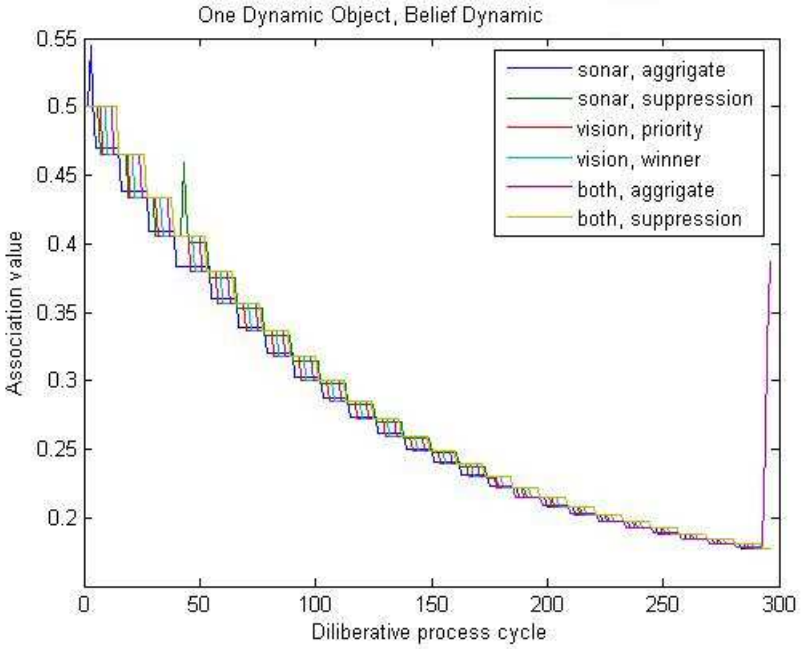

One Dynamic Object, Belief Dynamic
sonar, priority
sonar, aggrigate
both, priority
both, aggrigate
both, winner
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120
140
160
180
200


One Dynamic Object, Belief Dynamic
sonar, priority
sonar, suppression
vision, aggrigate
vision, winner
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160

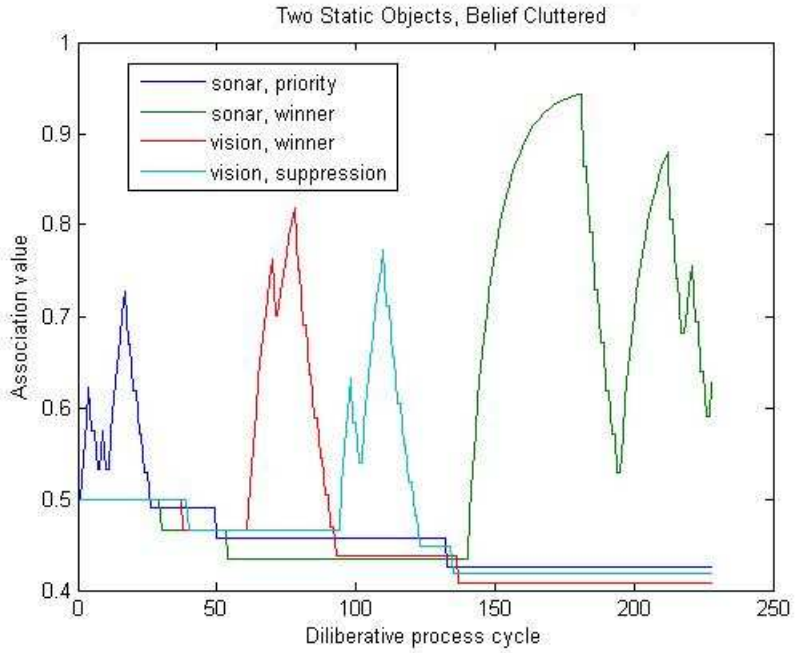

One Dynamic Object, Belief Dynamic
sonar, priority
sonar, aggrigate
vision, aggrigate
vision, winner
both, winner
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160
180

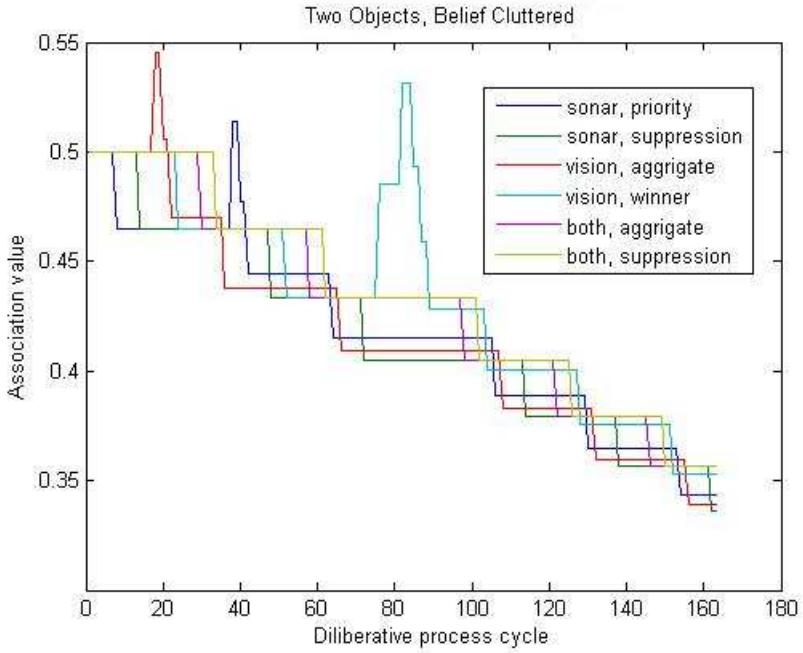

One Dynamic Object, Belief Dynamic
sonar, winner
sonar, suppression
vision, priority
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
20
40
60
80
100
120
140
160
180
200

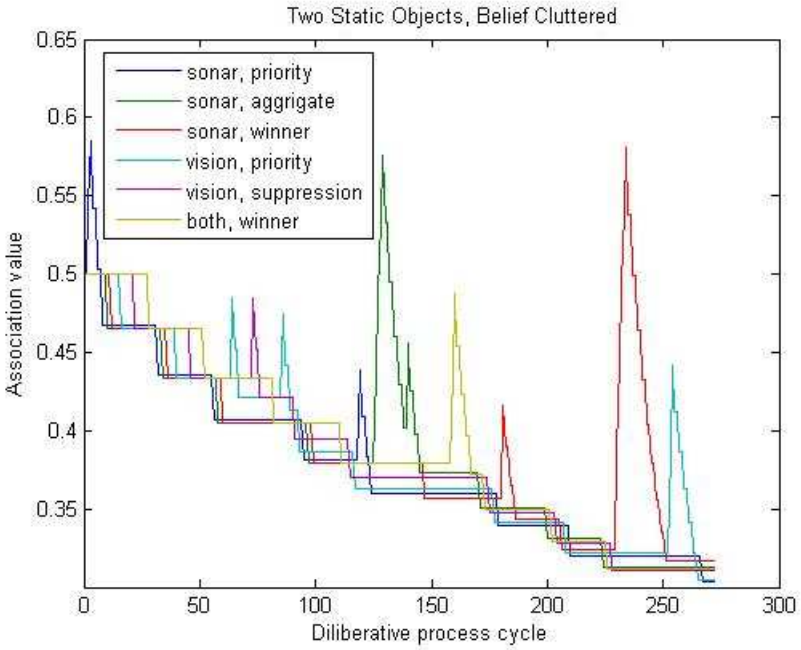

One Dynamic Object, Belief Dynamic
sonar, priority
sonar, aggrigate
vision, winner
vision, suppression
both, priority
both, winner
Association value
Diliberative process cycle
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
50
100
150

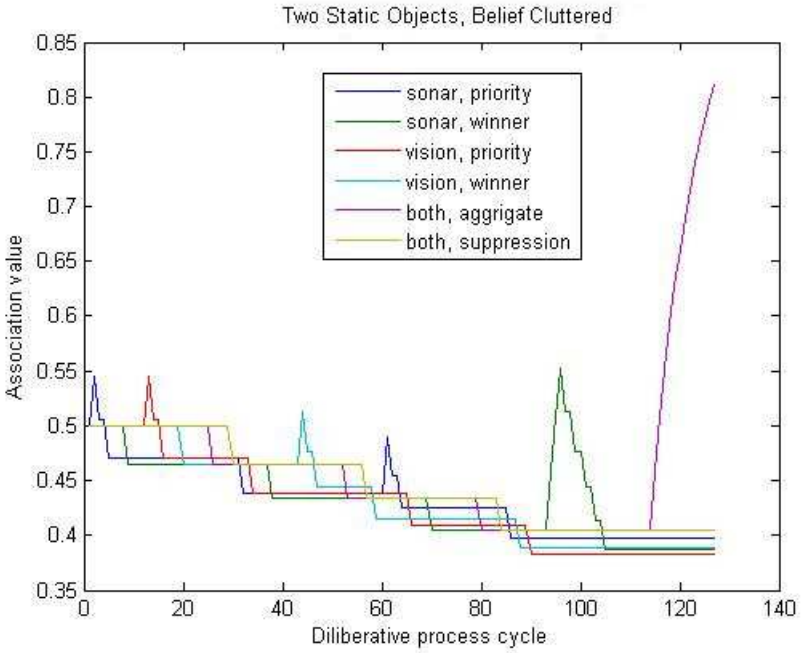

One Dynamic Object, Belief Dynamic
sonar, aggrigate
sonar, suppression
vision, priority
vision, winner
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300

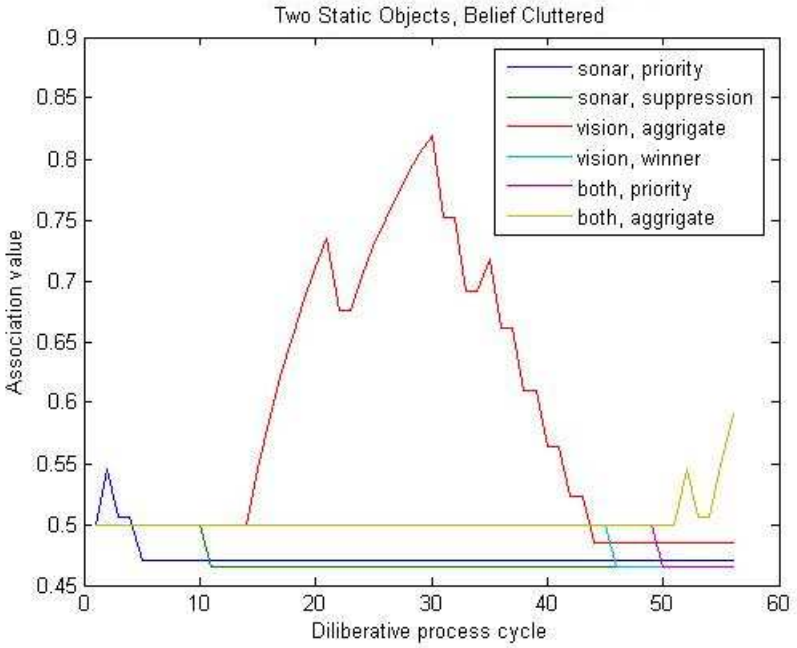
C.3.4 Two Objects, Static Environment

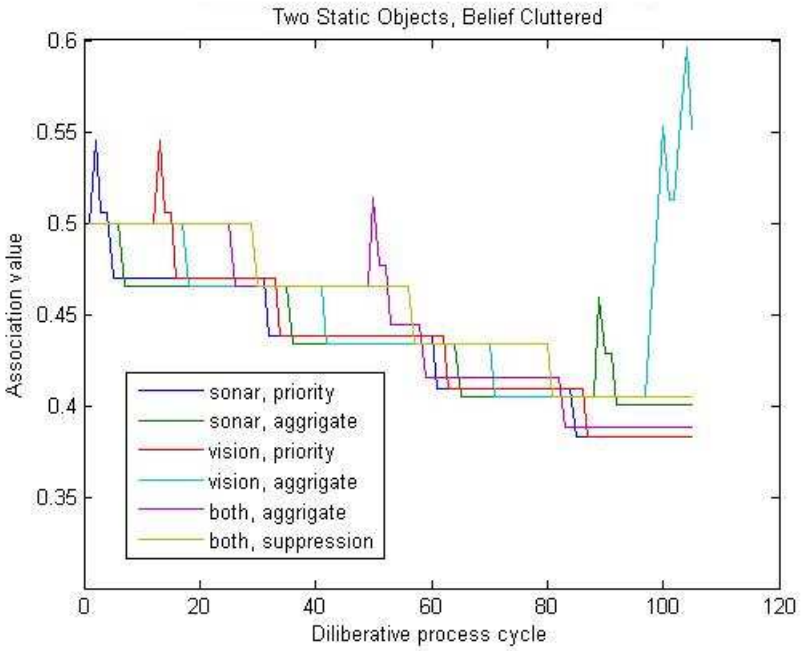


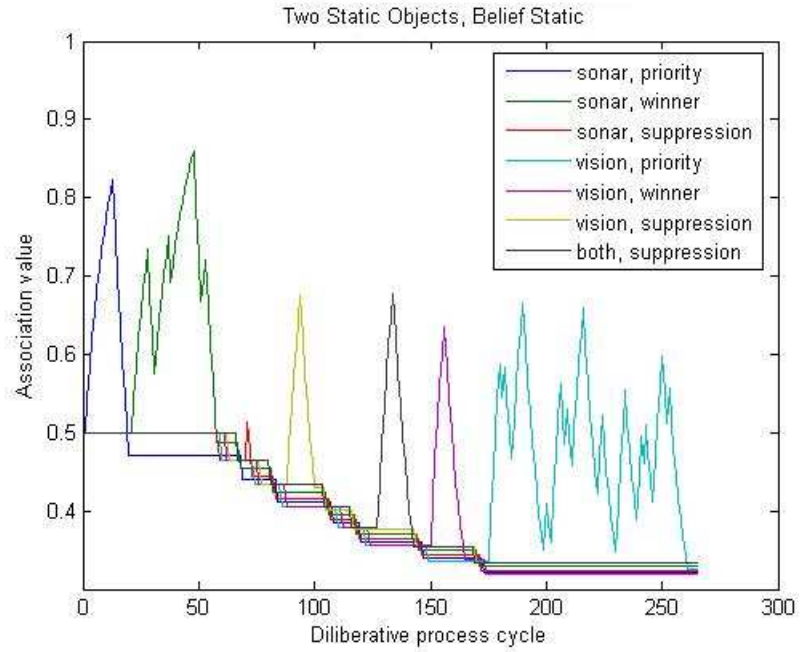


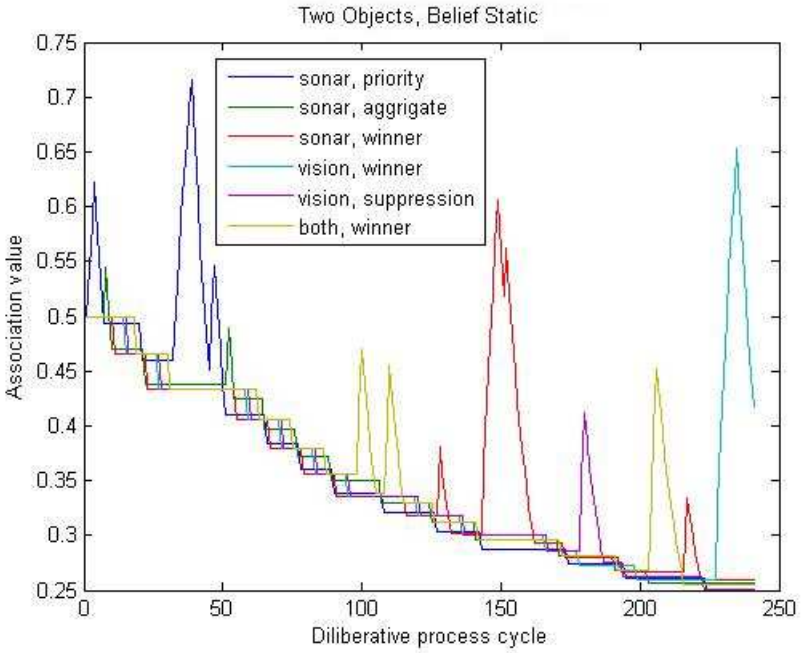

Two Static Objects, Belief Cluttered
sonar, priority
sonar, aggrigate
sonar, winner
vision, priority
vision, suppression
both, winner
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
50
100
150
200
250
300

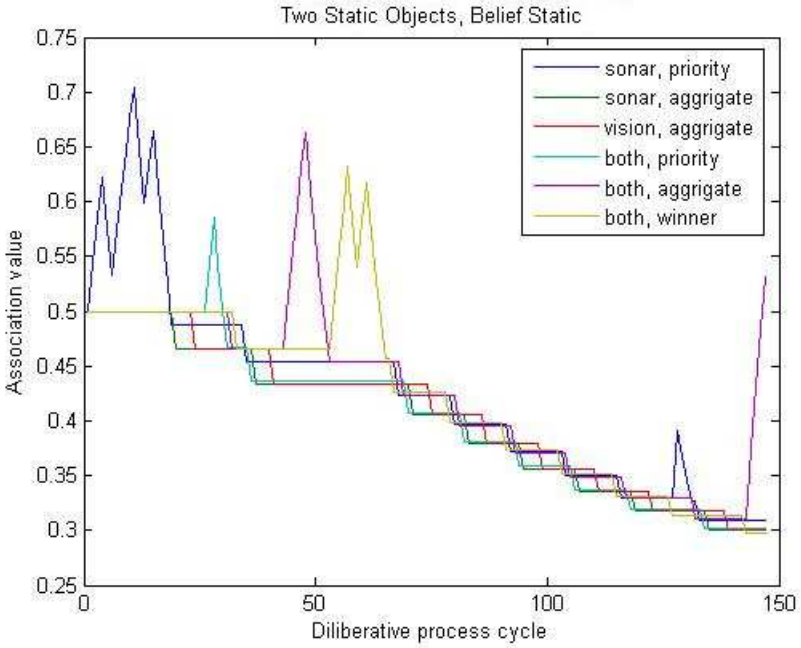

Two Static Objects, Belief Cluttered
sonar, priority
sonar, winner
vision, priority
vision, winner
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140

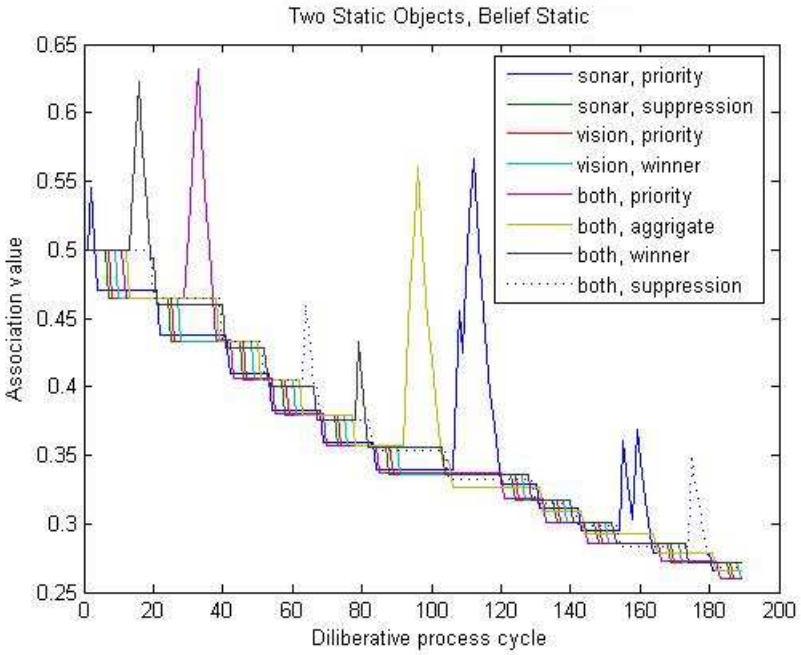

Two Static Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, priority
sonar, suppression
vision, aggrigate
vision, winner
both, priority
both, aggrigate
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
10
20
30
40
50
60

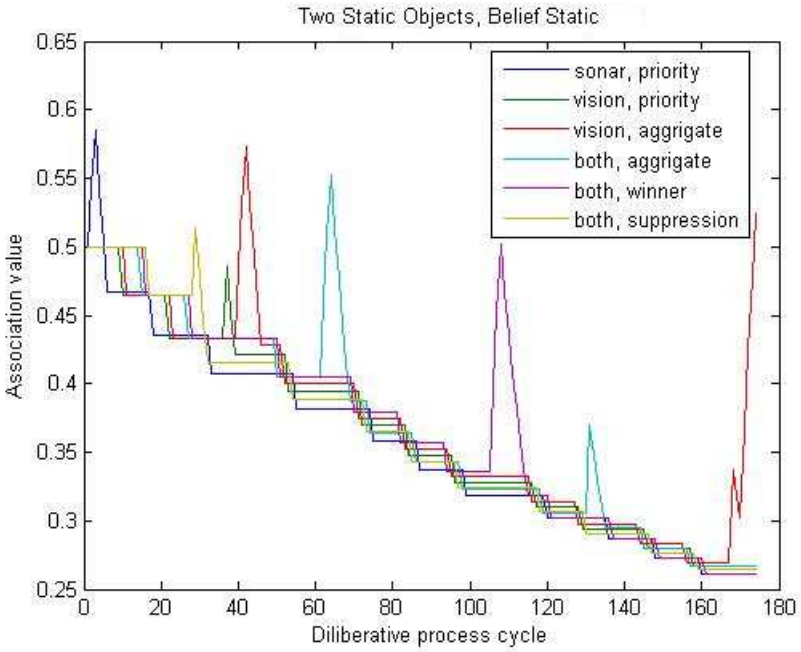

Two Static Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, priority
sonar, aggrigate
vision, priority
vision, aggrigate
both, aggrigate
both, suppression
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120

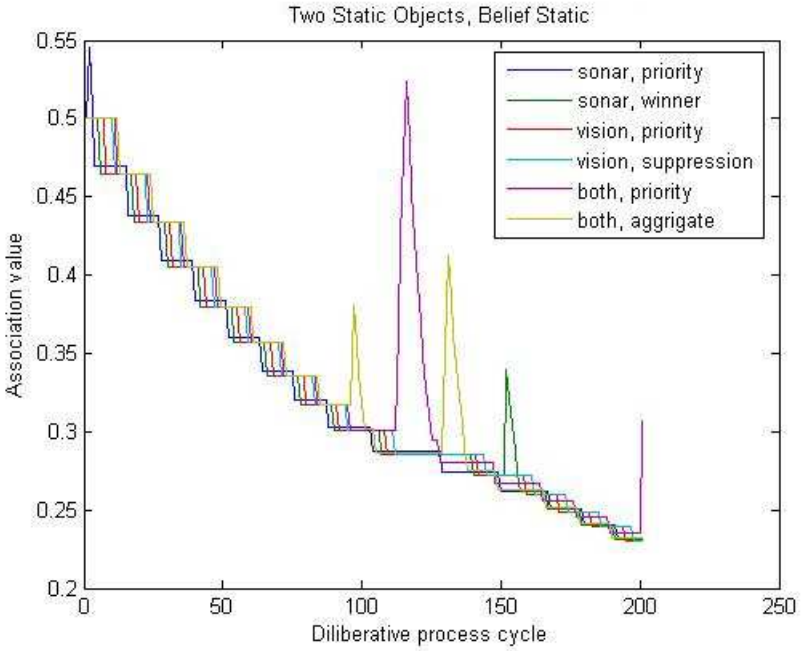

Two Static Objects, Belief Static
sonar, priority
sonar, winner
sonar, suppression
vision, priority
vision, winner
vision, suppression
both, suppression
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250
300

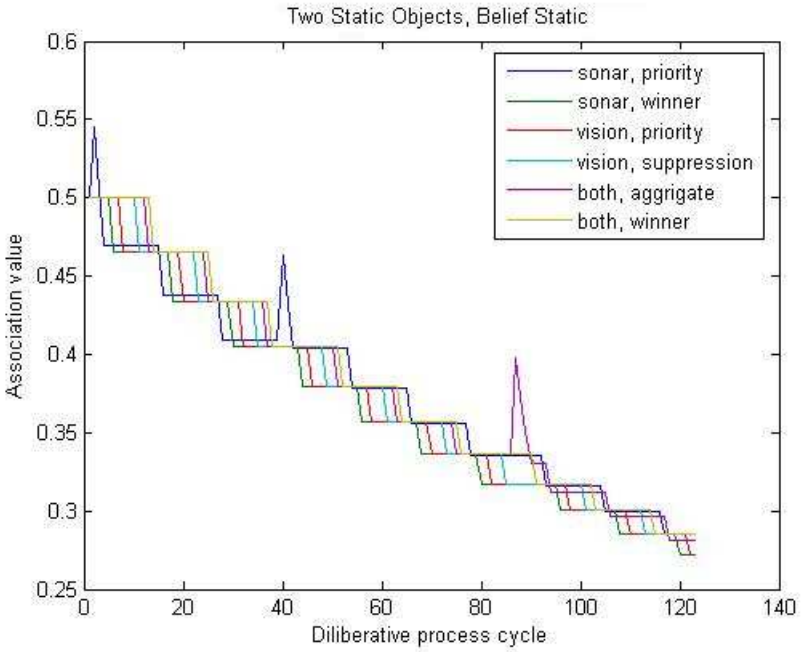

Two Objects, Belief Static
sonar, priority
sonar, aggrigate
sonar, winner
vision, winner
vision, suppression
both, winner
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
50
100
150
200
250


Two Static Objects, Belief Static
sonar, priority
sonar, aggrigate
vision, aggrigate
both, priority
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
50
100
150

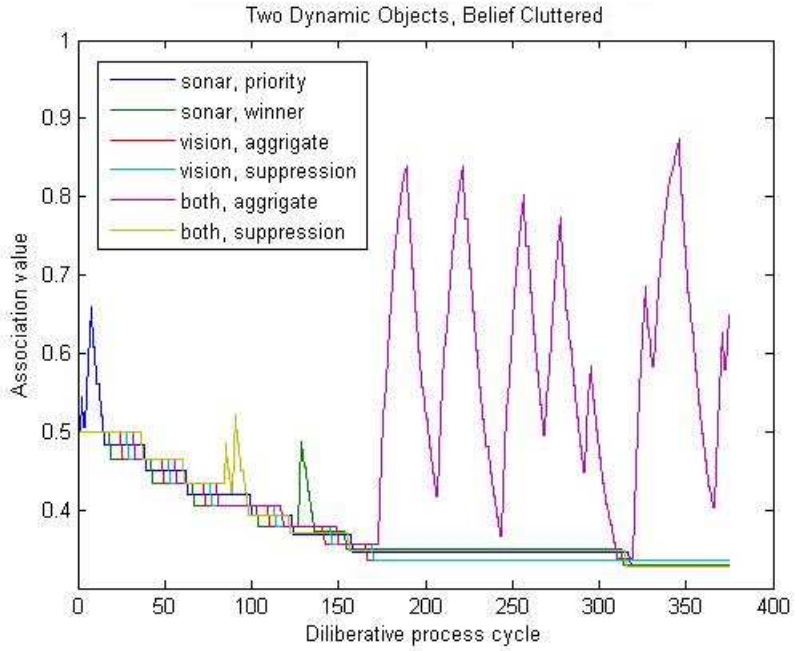

Two Static Objects, Belief Static
sonar, priority
sonar, suppression
vision, priority
vision, winner
both, priority
both, aggrigate
both, winner
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160
180
200

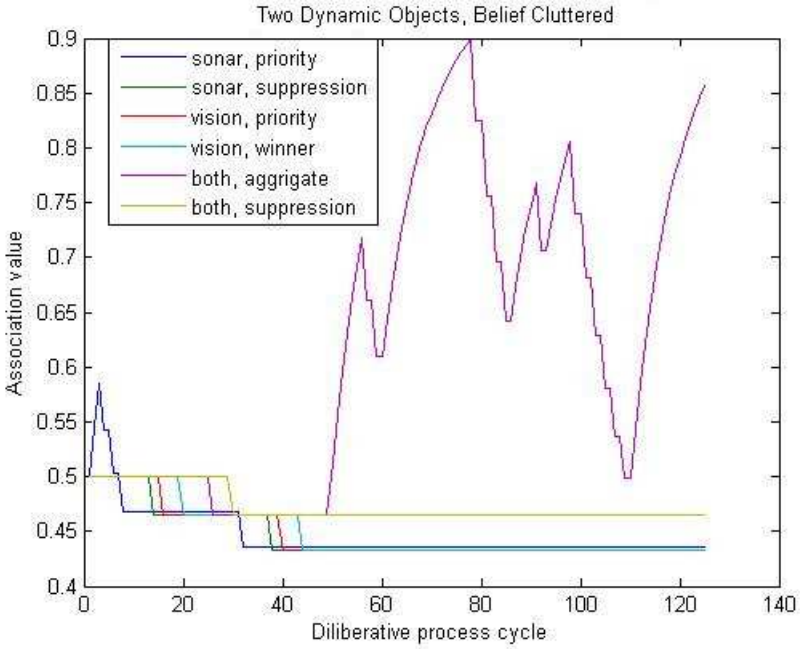

Two Static Objects, Belief Static
sonar, priority
vision, priority
vision, aggrigate
both, aggrigate
both, winner
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160
180

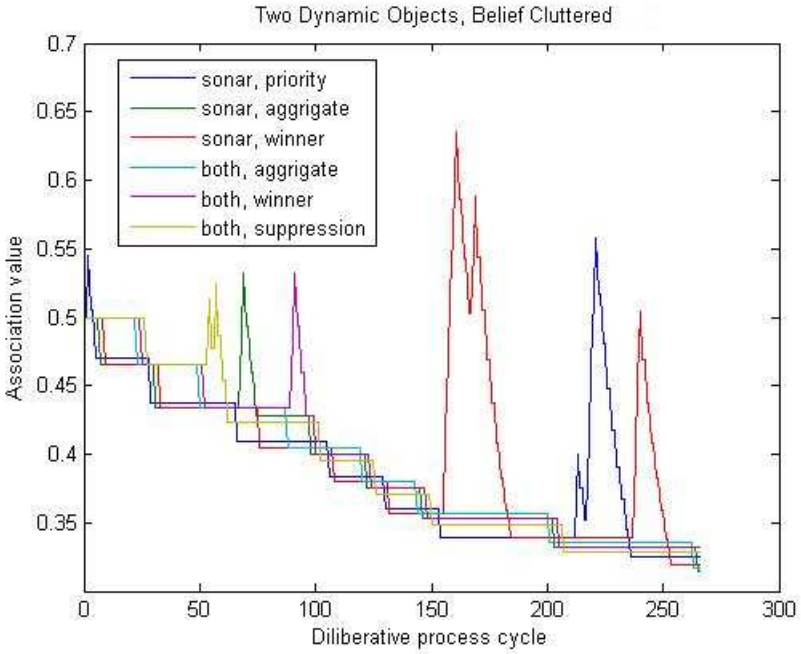

Two Static Objects, Belief Static
sonar, priority
sonar, winner
vision, priority
vision, suppression
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

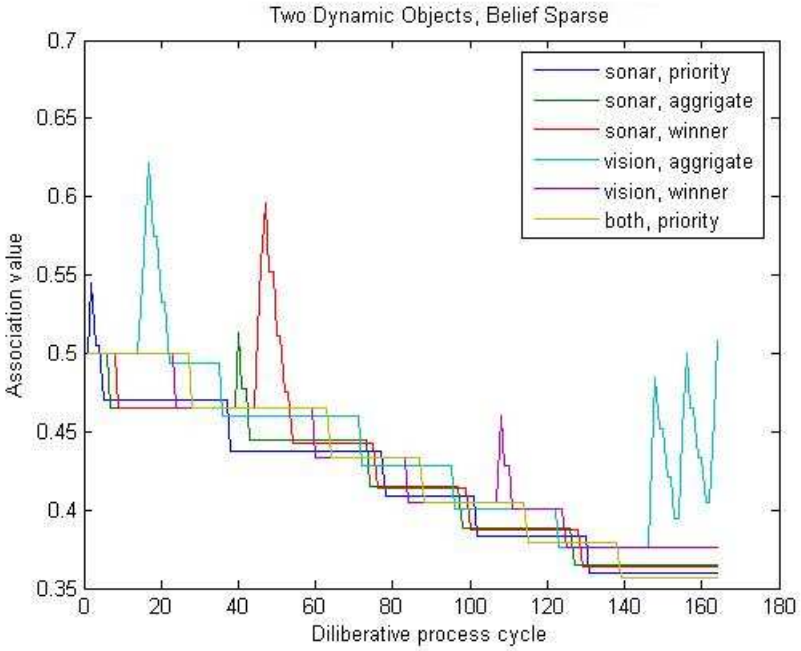

Two Static Objects, Belief Static
sonar, priority
sonar, winner
vision, priority
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140

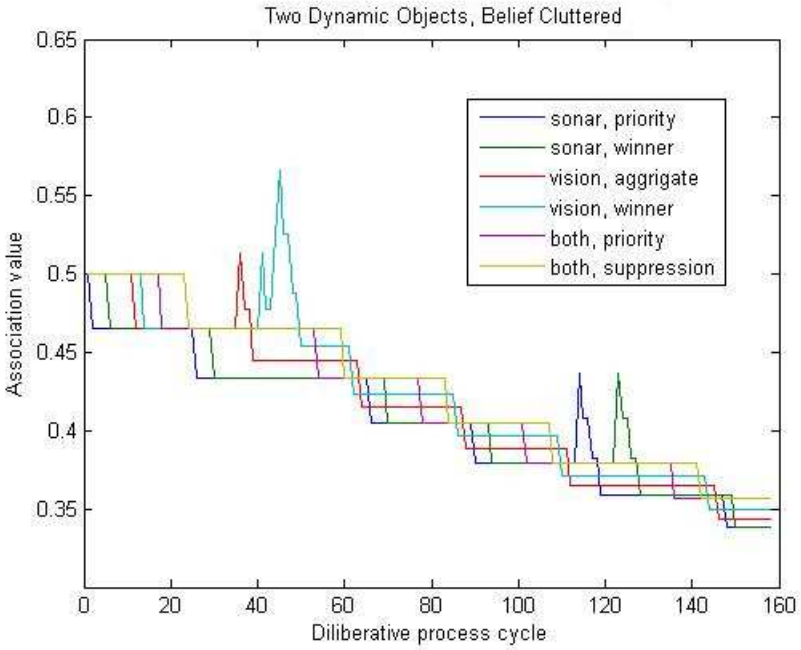
C.3.5 Two Objects, Dynamic Environment

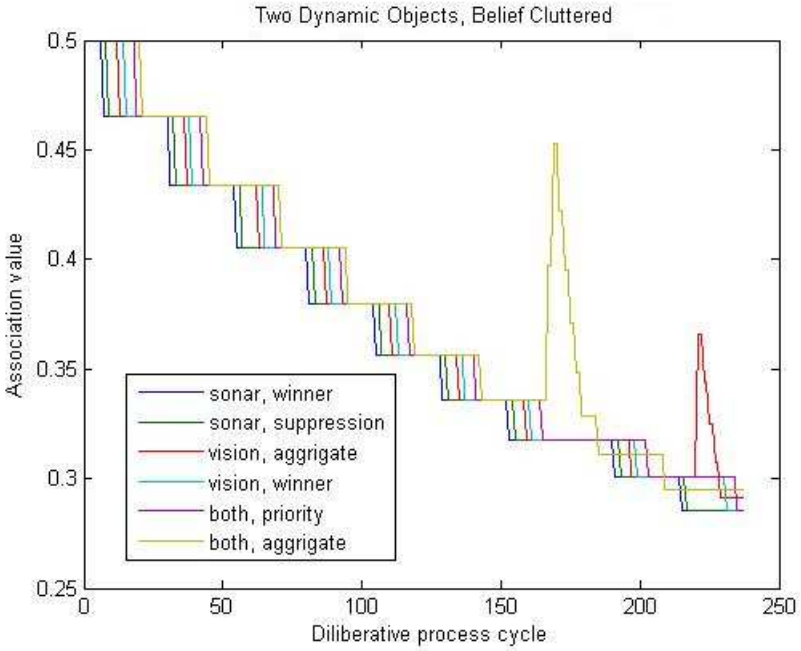


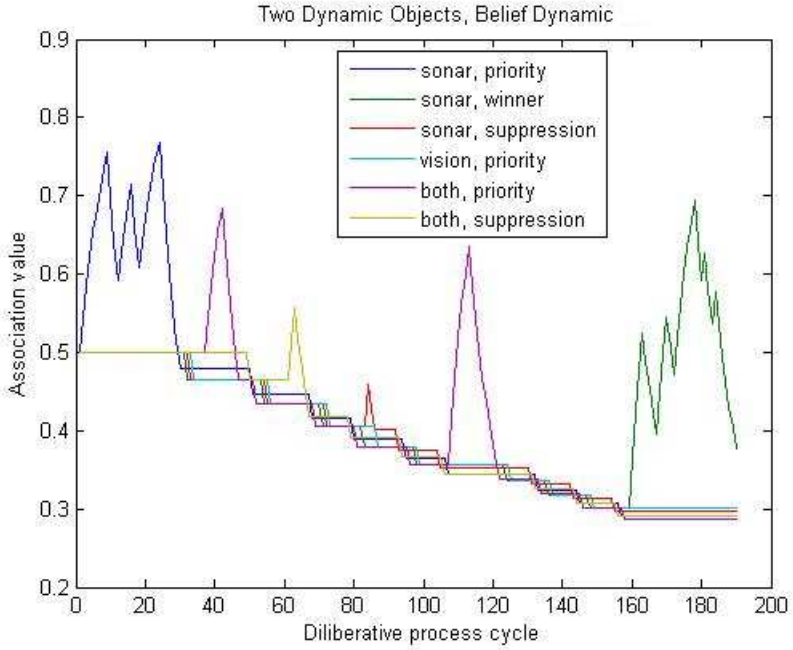


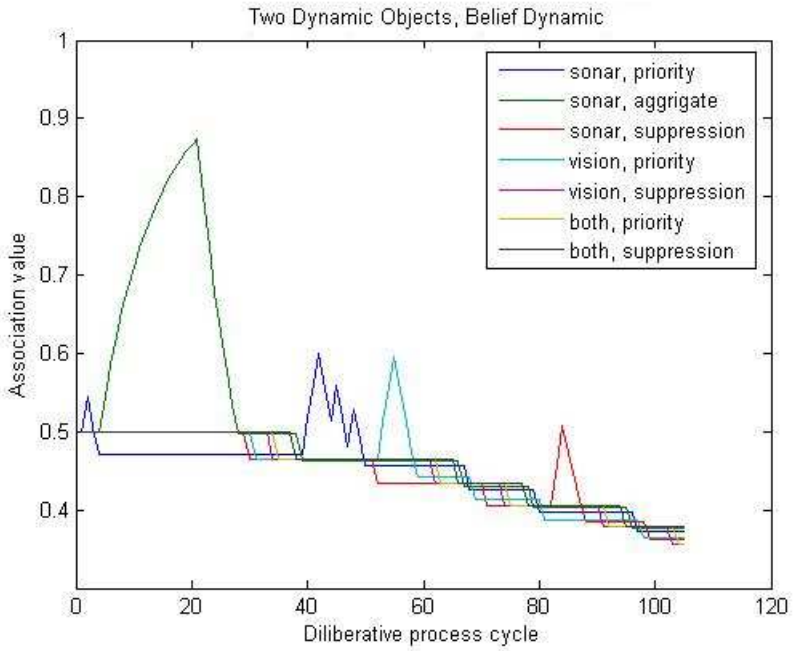

Two Dynamic Objects, Belief Cluttered
sonar, priority
sonar, aggrigate
sonar, winner
both, aggrigate
both, winner
both, suppression
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
50
100
150
200
250
300

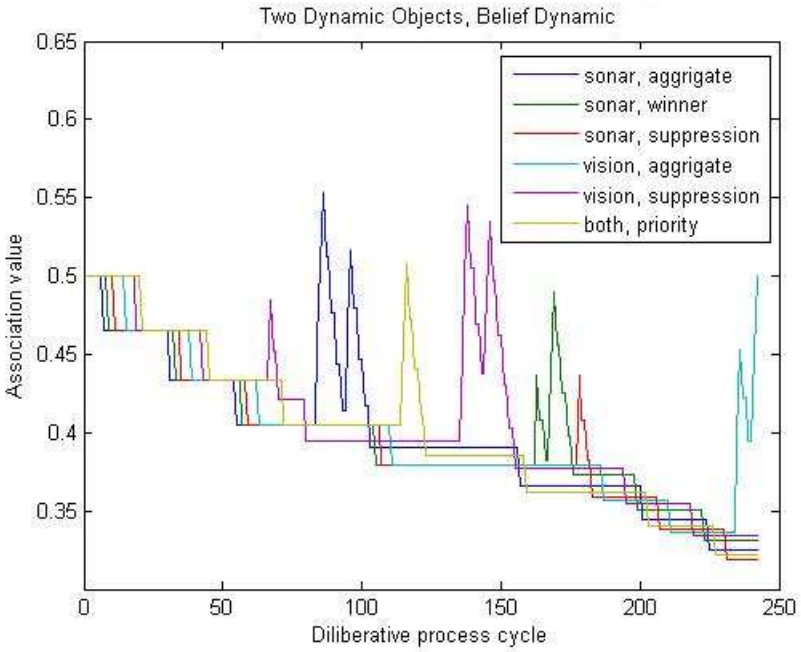

Two Dynamic Objects, Belief Sparse
sonar, priority
sonar, aggrigate
sonar, winner
vision, aggrigate
vision, winner
both, priority
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180

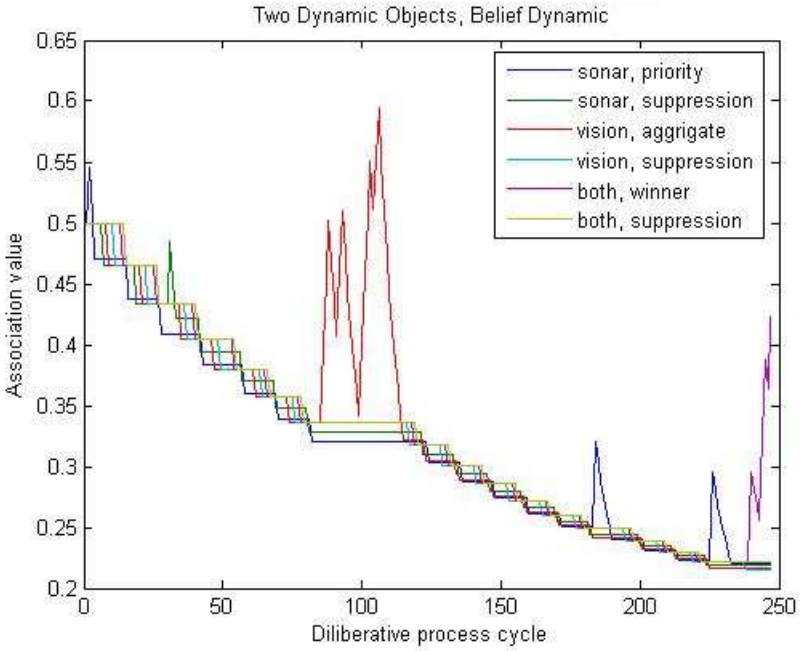

Two Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, priority
sonar, winner
vision, aggrigate
vision, winner
both, priority
both, suppression
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160

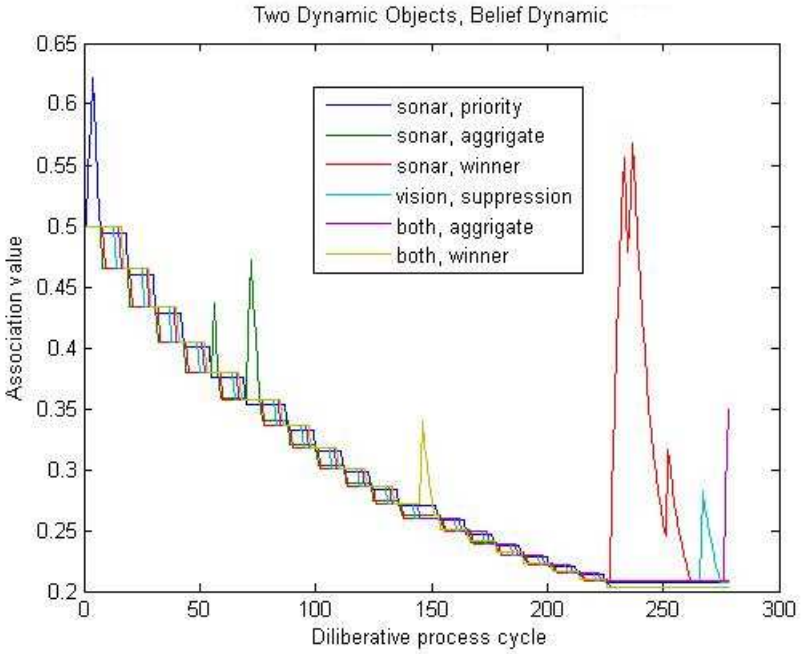

Two Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, winner
sonar, suppression
vision, aggrigate
vision, winner
both, priority
both, aggrigate
0.5
0.45
0.4
0.35
0.3
0.25
0
50
100
150
200
250

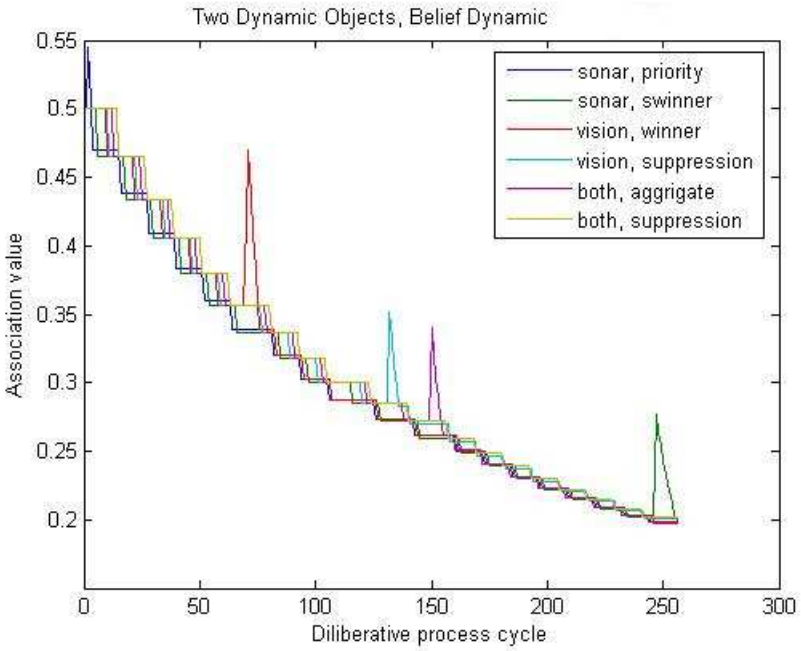

Two Dynamic Objects, Belief Dynamic
sonar, priority
sonar, winner
sonar, suppression
vision, priority
both, priority
both, suppression
Association value
Diliberative process cycle
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
120
140
160
180
200


Two Dynamic Objects, Belief Dynamic
sonar, priority
sonar, aggrigate
sonar, suppression
vision, priority
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120

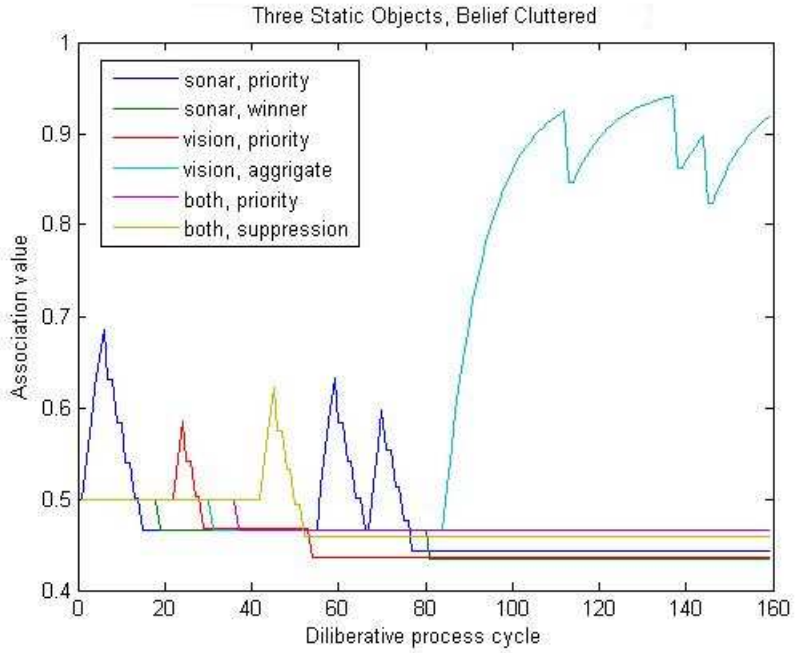

Two Dynamic Objects, Belief Dynamic
sonar, aggrigate
sonar, winner
sonar, suppression
vision, aggrigate
vision, suppression
both, priority
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
50
100
150
200
250

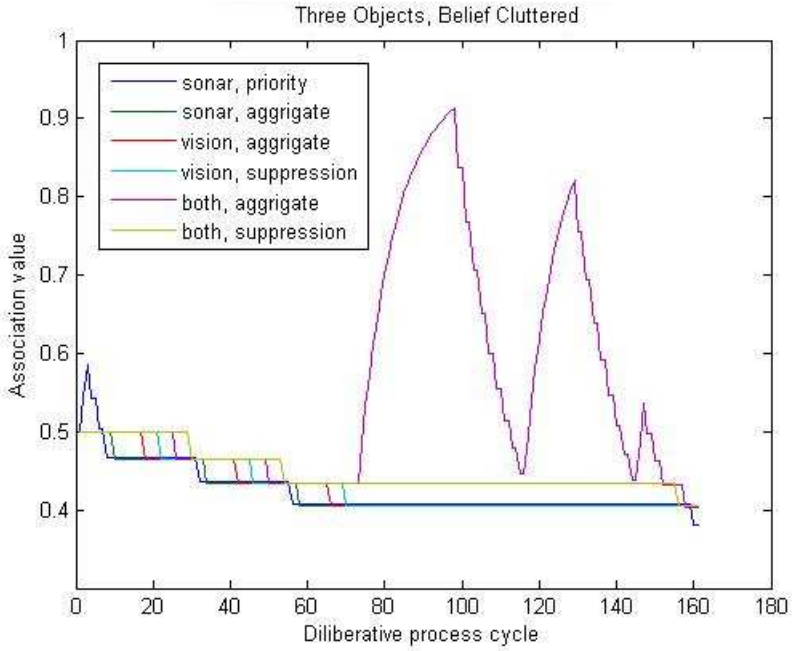

Two Dynamic Objects, Belief Dynamic
sonar, priority
sonar, suppression
vision, aggrigate
vision, suppression
both, winner
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

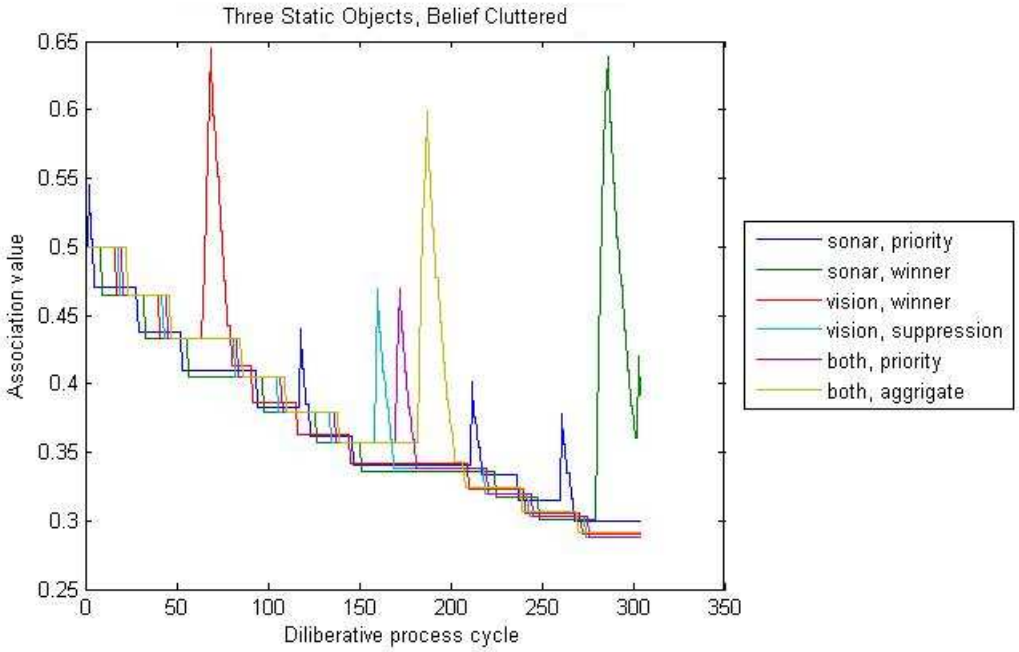

Two Dynamic Objects, Belief Dynamic
sonar, priority
sonar, aggrigate
sonar, winner
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300

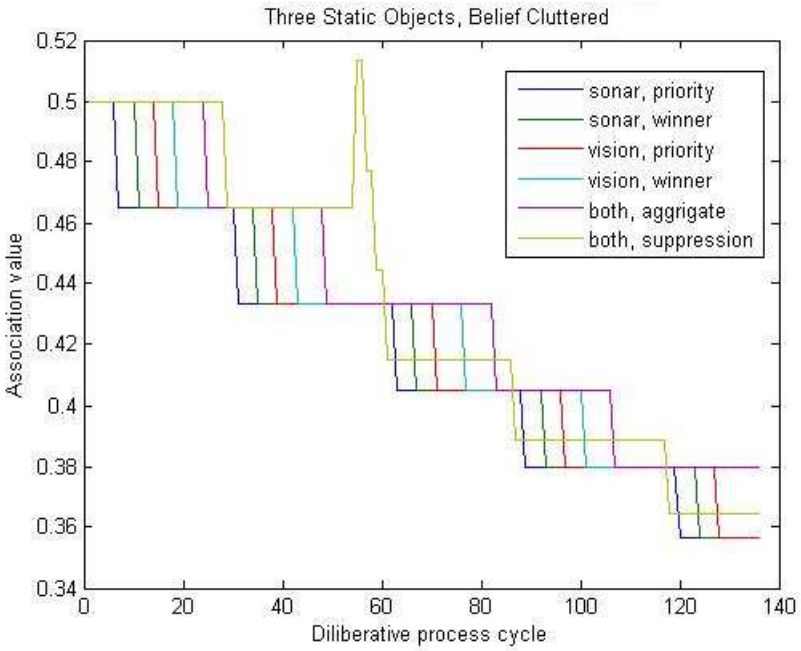

Two Dynamic Objects, Belief Dynamic
sonar, priority
sonar, swinner
vision, winner
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300

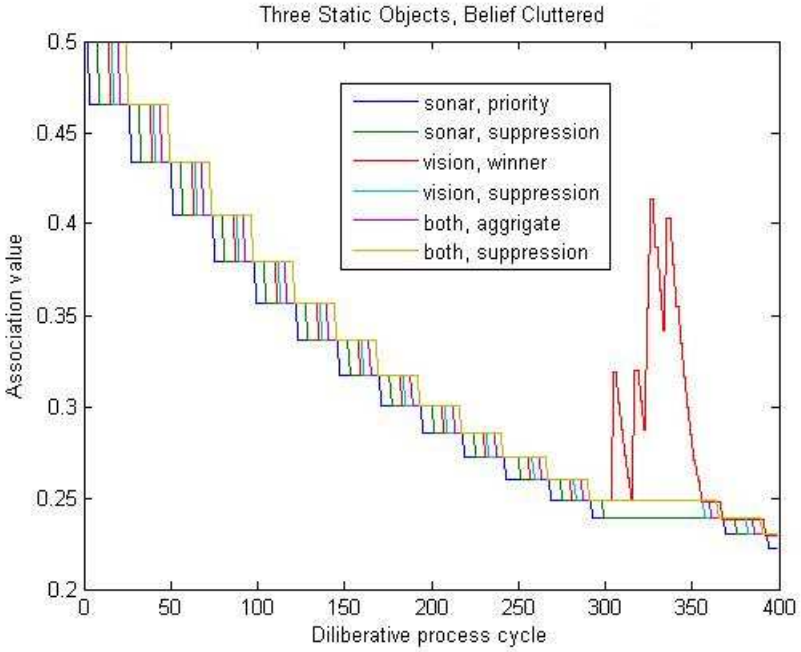
C.3.6 Three Objects, Static Environment

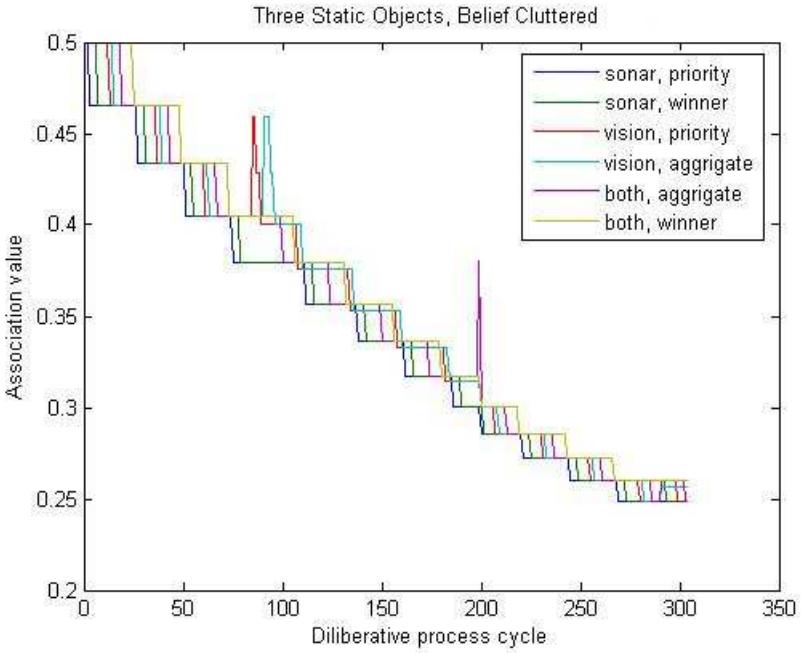


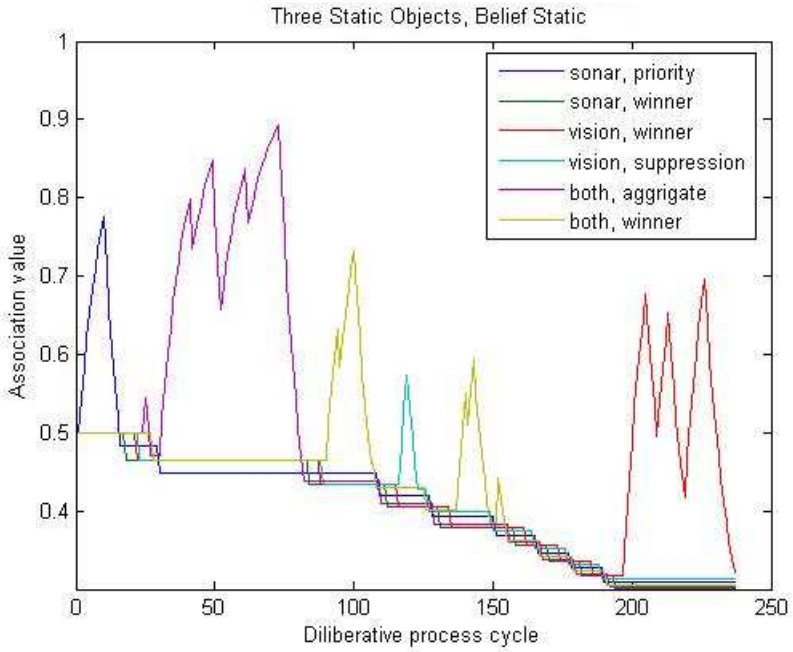


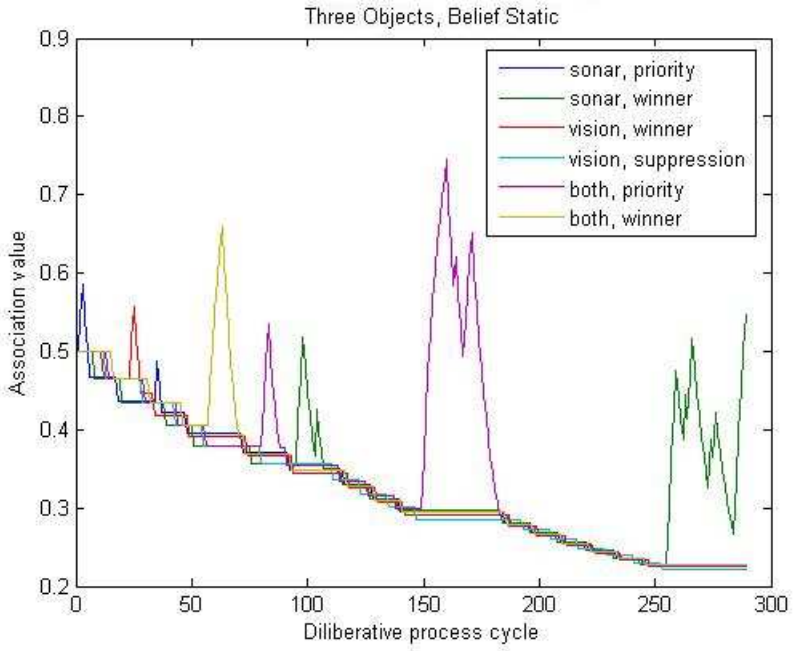

Three Static Objects, Belief Cluttered
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
50
100
150
200
250
300
350
sonar, priority
sonar, winner
vision, winner
vision, suppression
both, priority
both, aggrigate

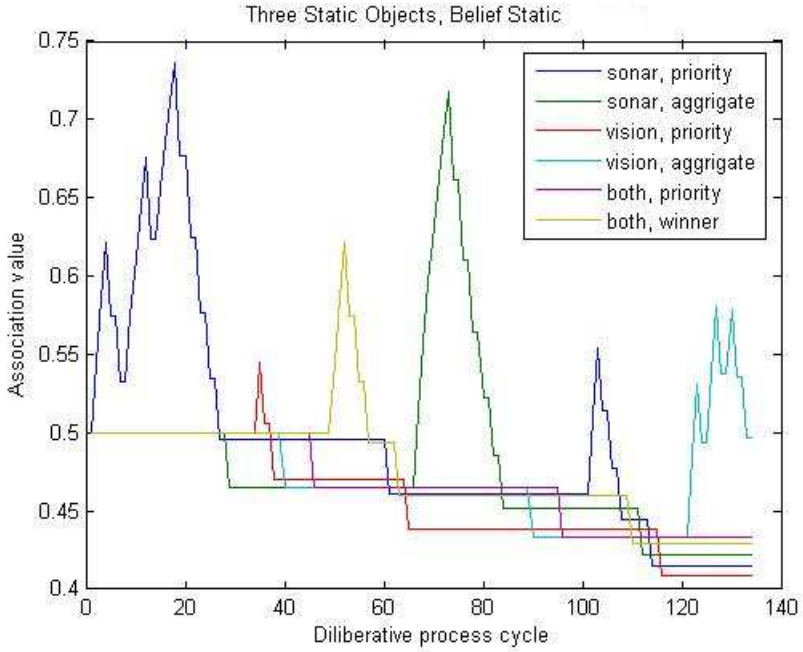

Three Static Objects, Belief Cluttered
Association value
Diliberative process cycle
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0
20
40
60
80
100
120
140
sonar, priority
sonar, winner
vision, priority
vision, winner
both, aggrigate
both, suppression

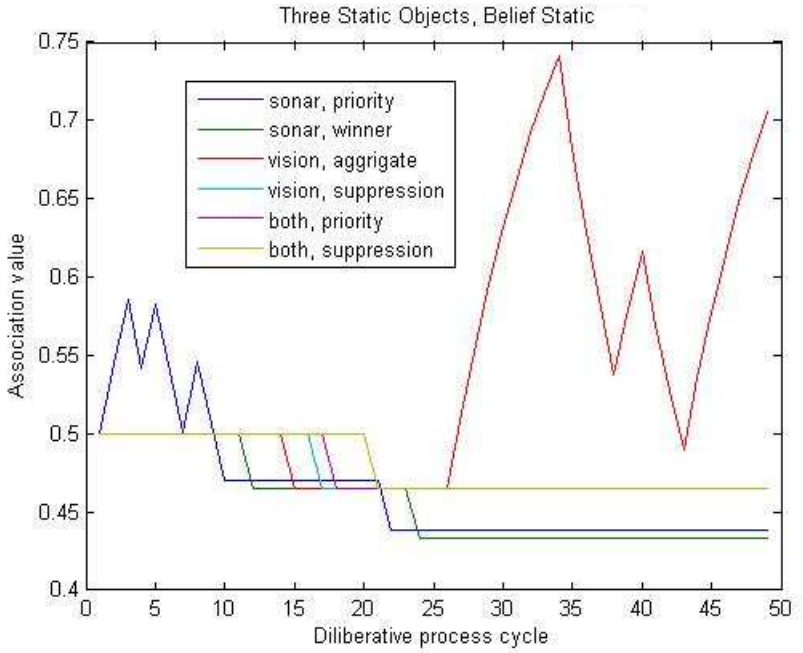

Three Static Objects, Belief Cluttered
sonar, priority
sonar, suppression
vision, winner
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350
400

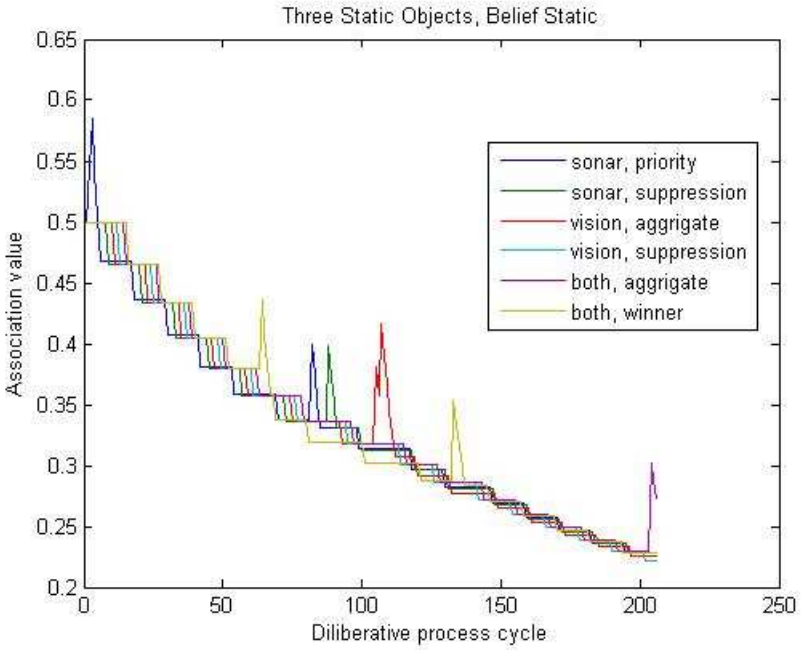

Three Static Objects, Belief Cluttered
sonar, priority
sonar, winner
vision, priority
vision, aggrigate
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350

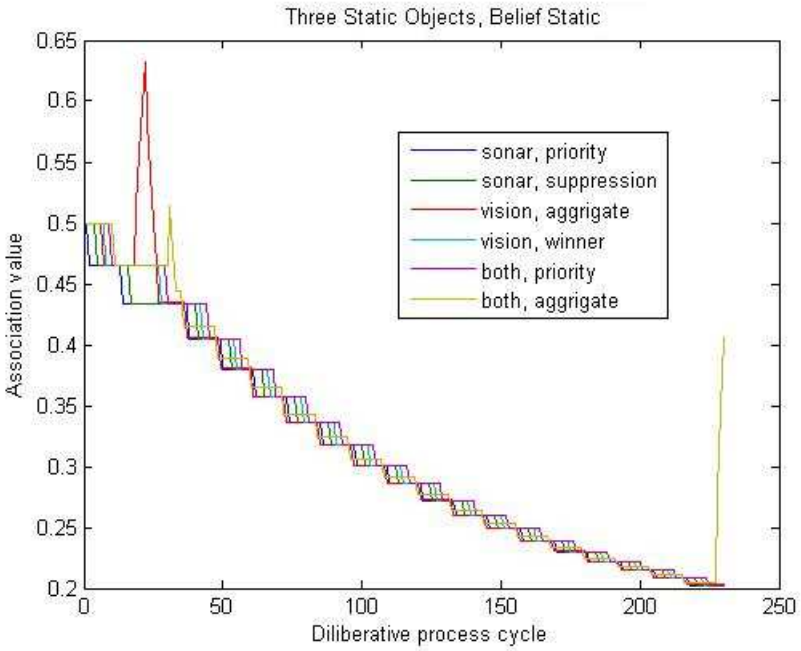

Three Static Objects, Belief Static
sonar, priority
sonar, winner
vision, winner
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0
50
100
150
200
250
0.4
0.5
0.6
0.7
0.8
0.9
1

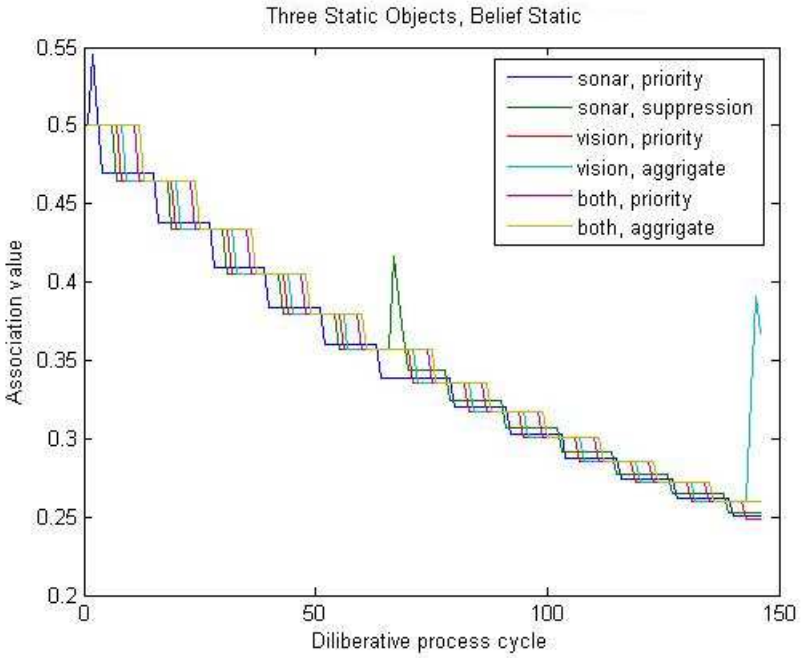

Three Objects, Belief Static
sonar, priority
sonar, winner
vision, winner
vision, suppression
both, priority
both, winner
Association value
Diliberative process cycle
0
50
100
150
200
250
300
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


Three Static Objects, Belief Static
sonar, priority
sonar, aggrigate
vision, priority
vision, aggrigate
both, priority
both, winner
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
20
40
60
80
100
120
140

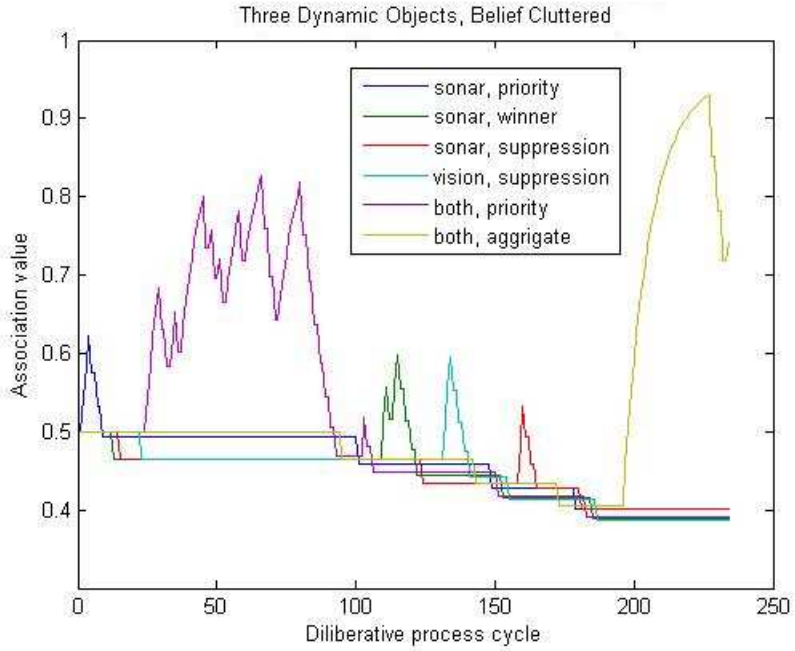

Three Static Objects, Belief Static
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
5
10
15
20
25
30
35
40
45
50

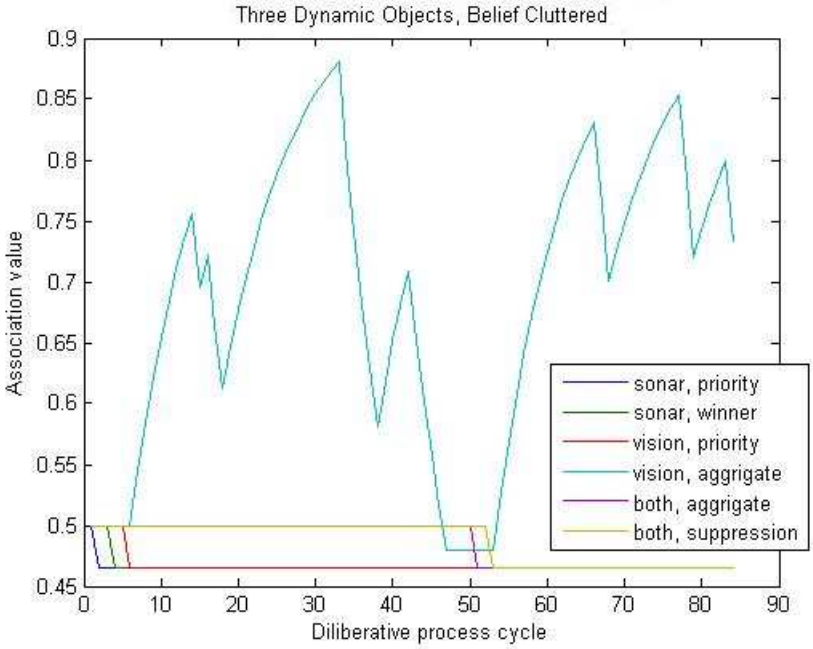

Three Static Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, suppression
vision, aggrigate
vision, suppression
both, aggrigate
both, winner
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

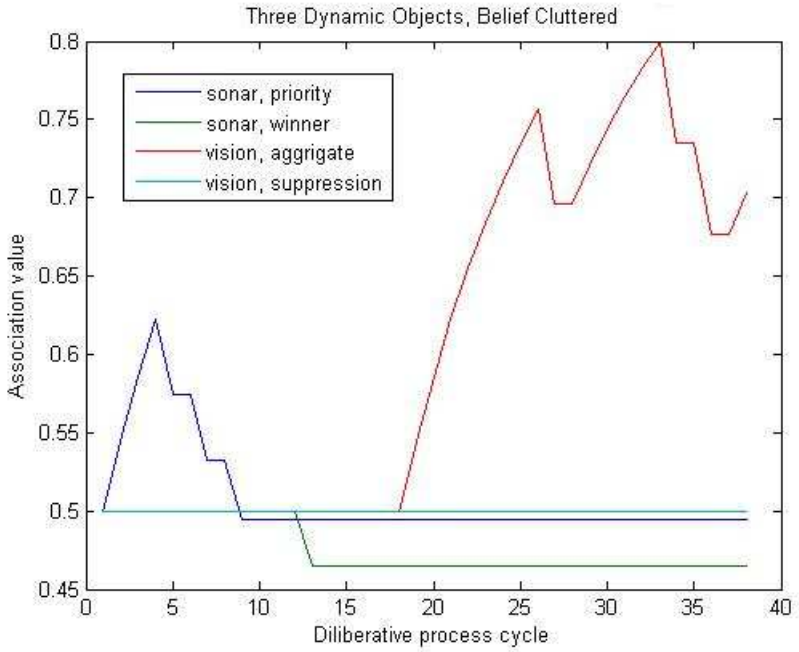

Three Static Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, suppression
vision, aggrigate
vision, winner
both, priority
both, aggrigate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

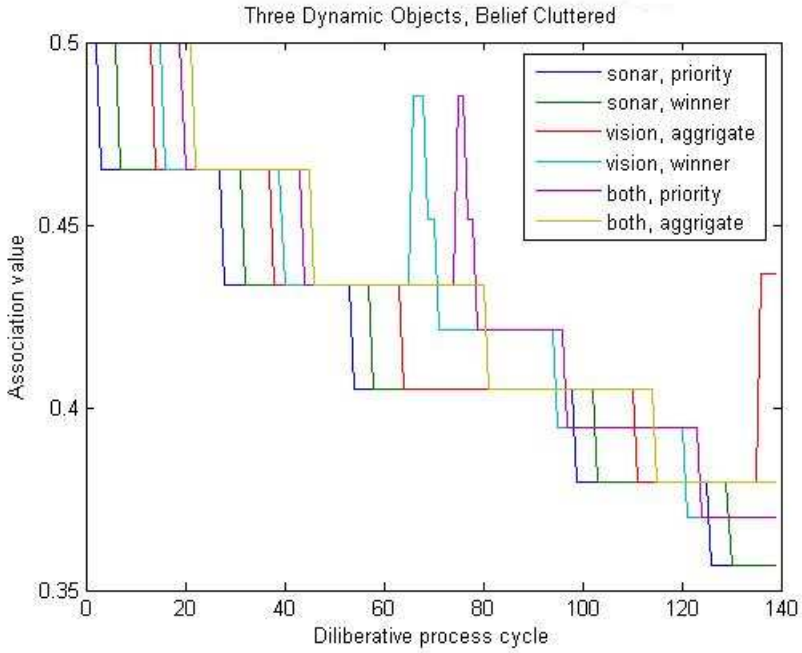

Three Static Objects, Belief Static
sonar, priority
sonar, suppression
vision, priority
vision, aggrigate
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150

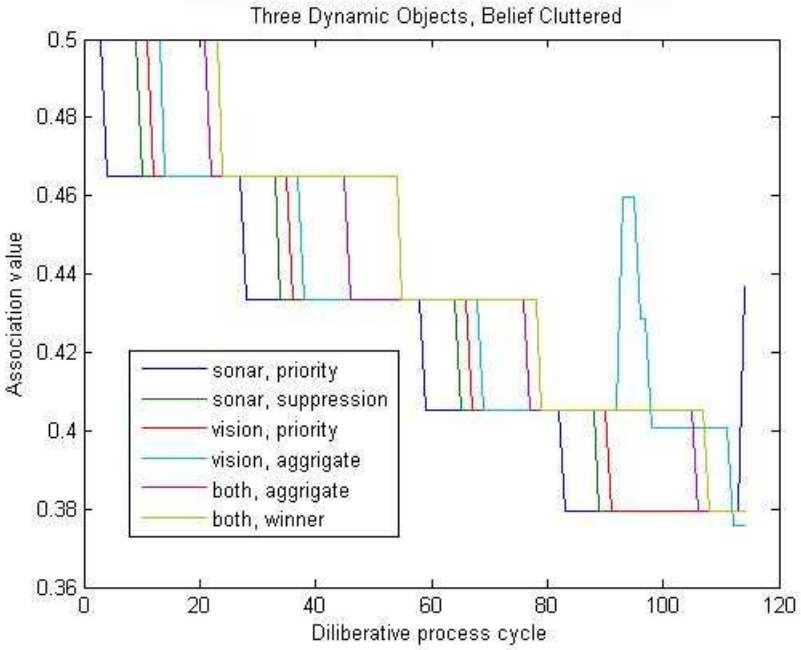
C.3.7 Three Objects, Dynamic Environment

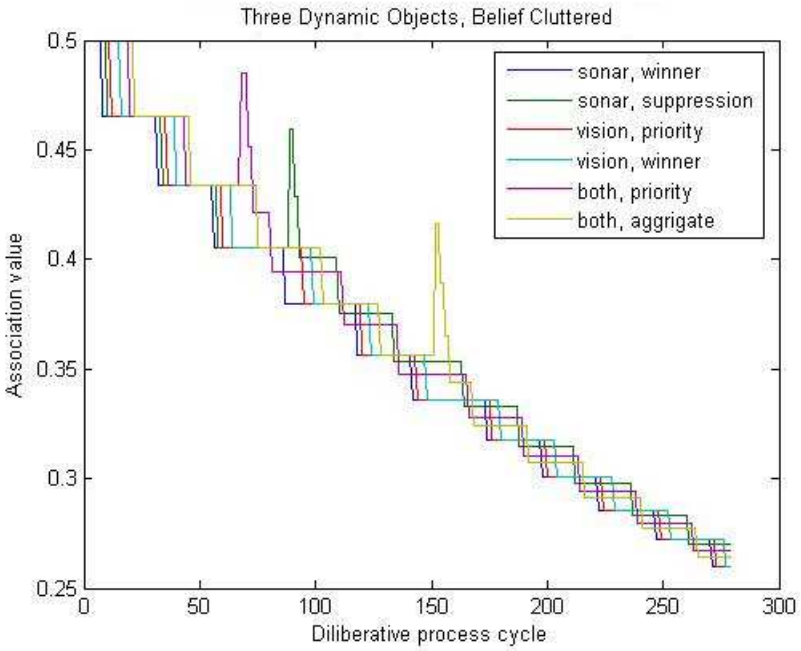


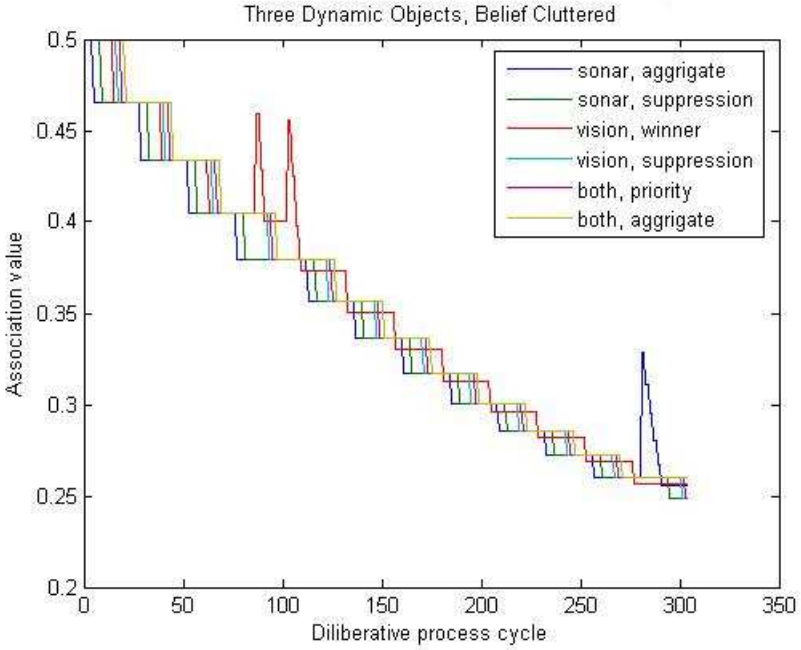


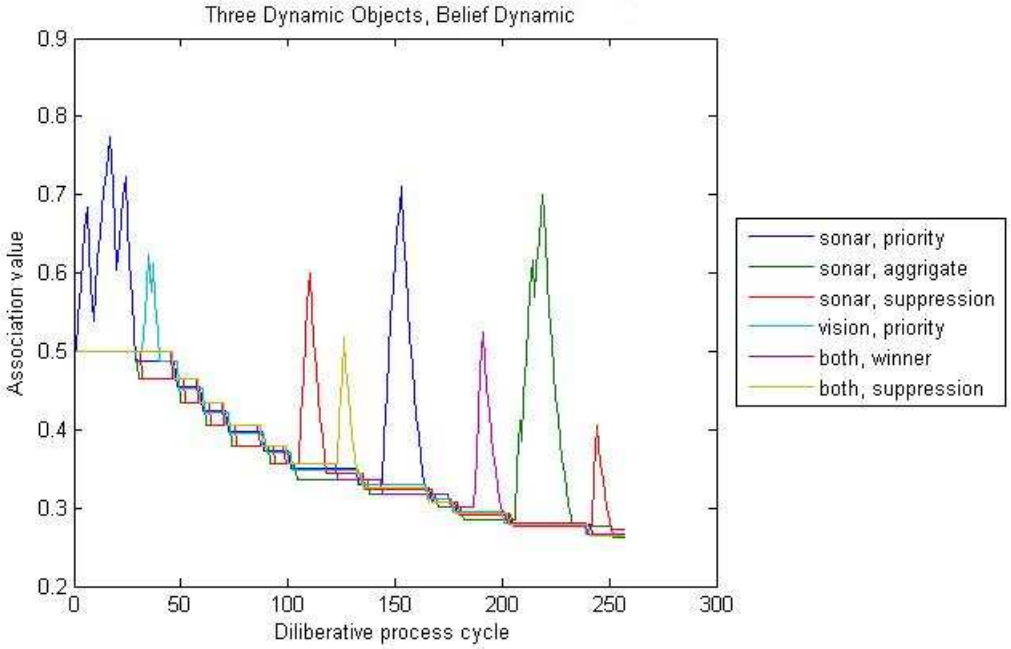

Three Dynamic Objects, Belief Cluttered
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
Association value
Diliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
5
10
15
20
25
30
35
40

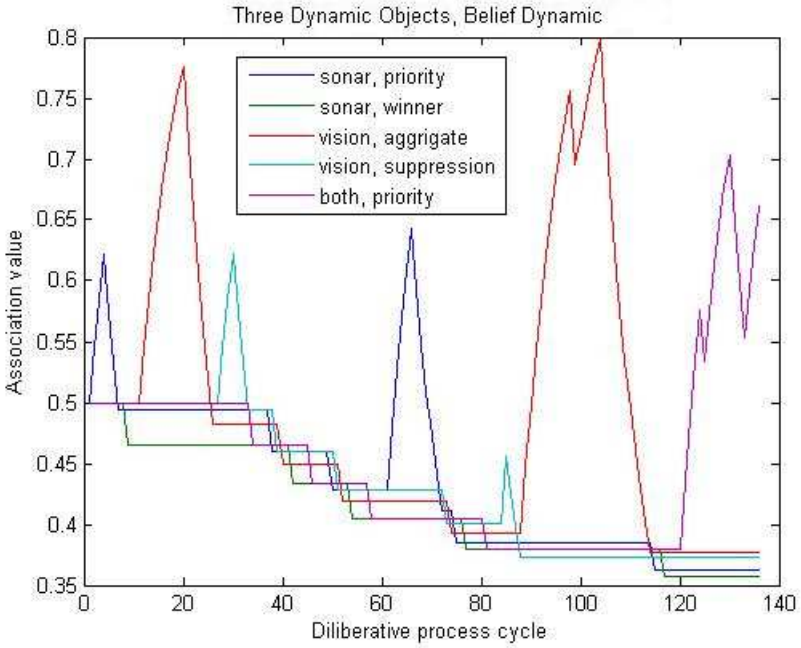

Three Dynamic Objects, Belief Cluttered
sonar, priority
sonar, winner
vision, aggrigate
vision, winner
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140

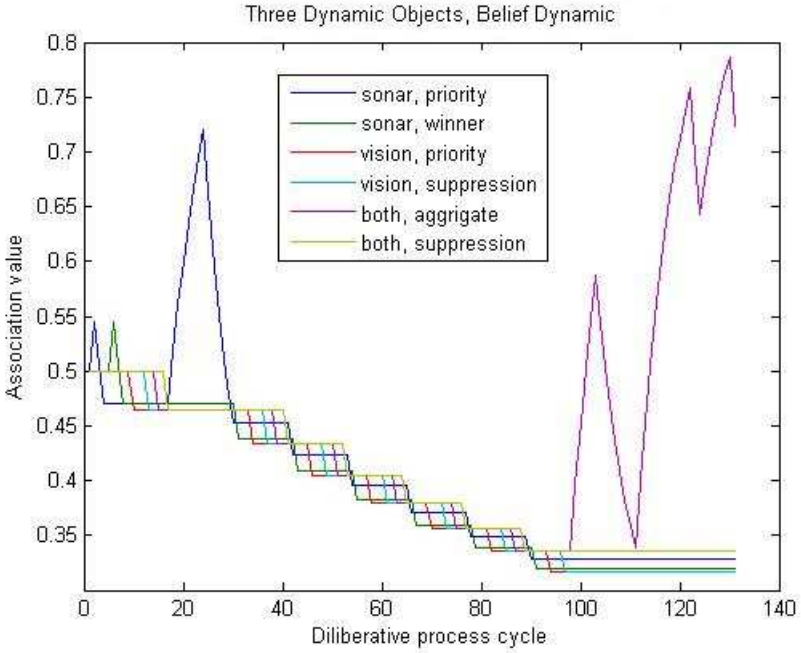

Three Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, priority
sonar, suppression
vision, priority
vision, aggrigate
both, aggrigate
both, winner

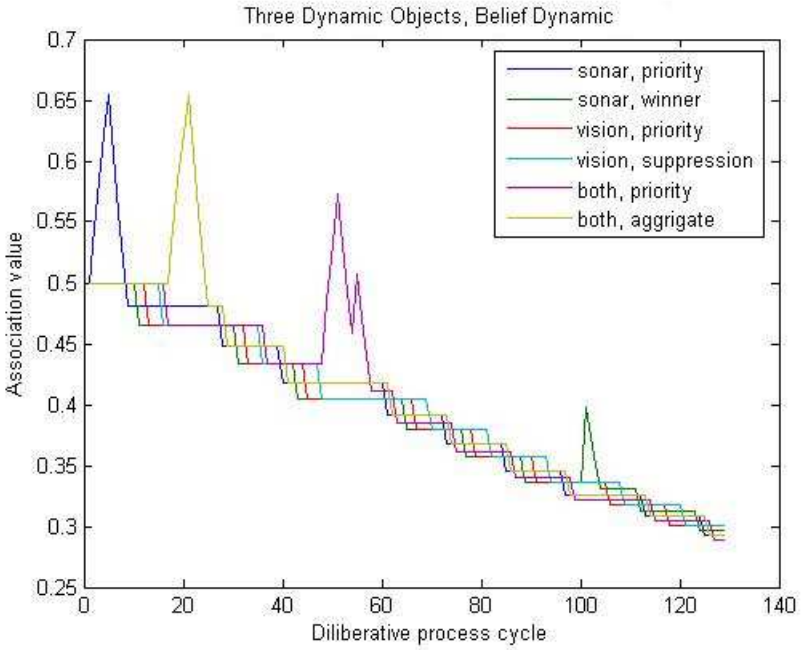

Three Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, winner
sonar, suppression
vision, priority
vision, winner
both, priority
both, aggrigate

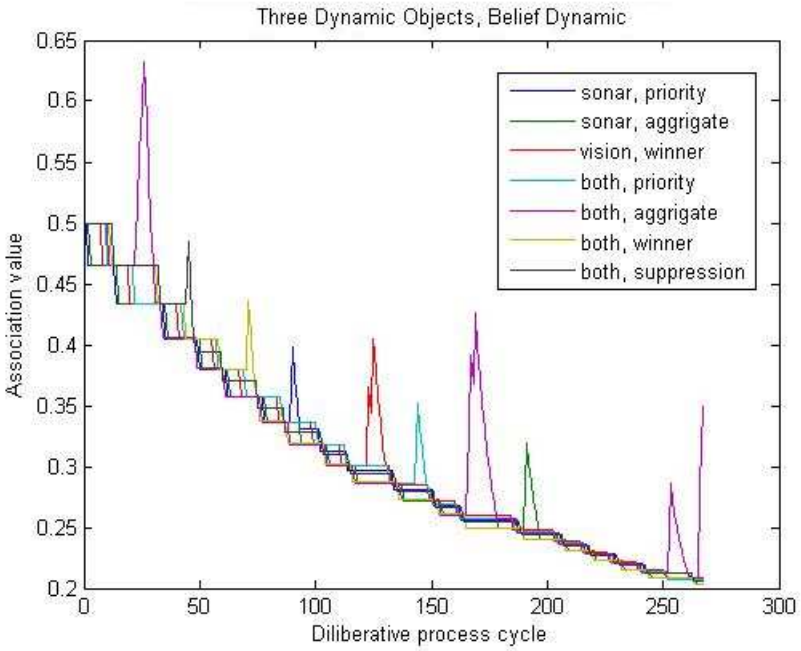

Three Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, aggrigate
sonar, suppression
vision, winner
vision, suppression
both, priority
both, aggrigate

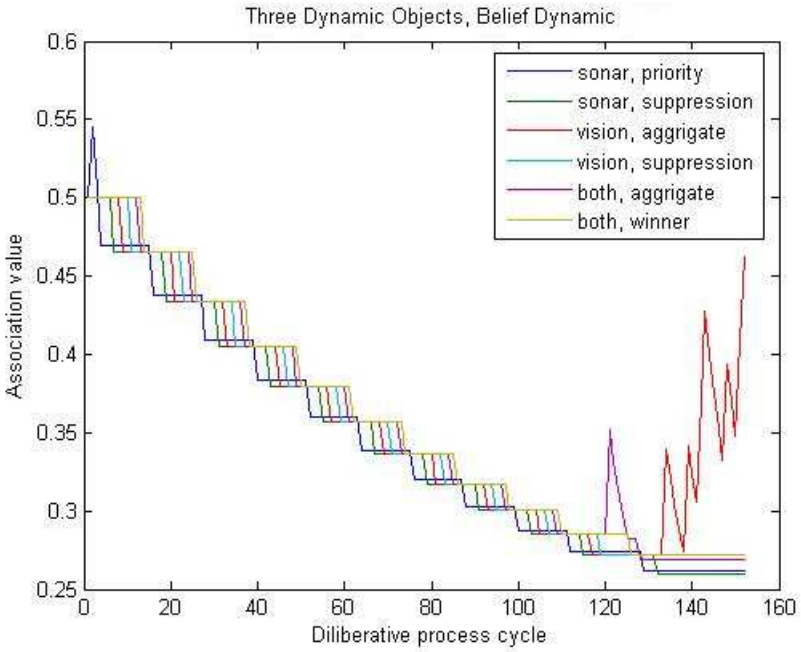

Three Dynamic Objects, Belief Dynamic
Association value
Diliberative process cycle
sonar, priority
sonar, aggrigate
sonar, suppression
vision, priority
both, winner
both, suppression

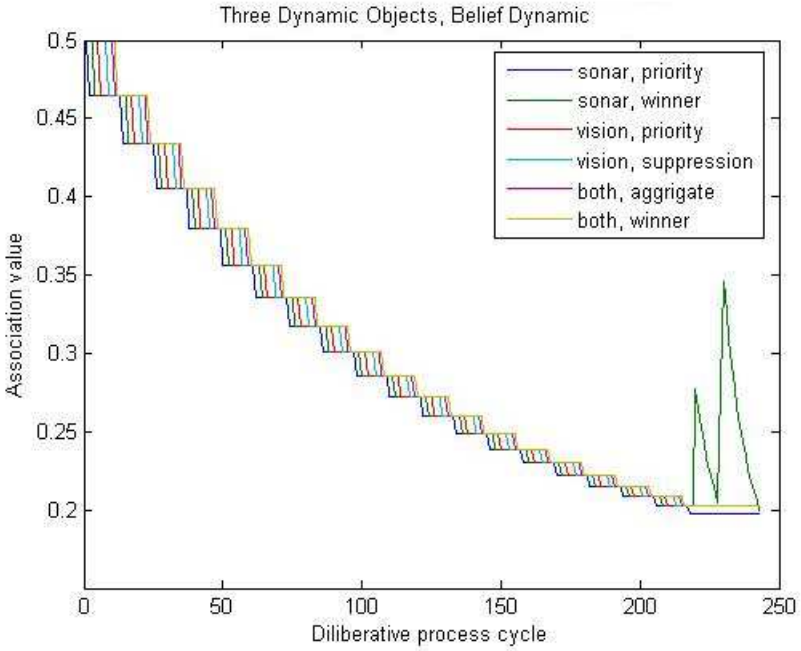

Three Dynamic Objects, Belief Dynamic
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
Association value
Diliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140


Three Dynamic Objects, Belief Dynamic
sonar, priority
sonar, winner
vision, priority
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140

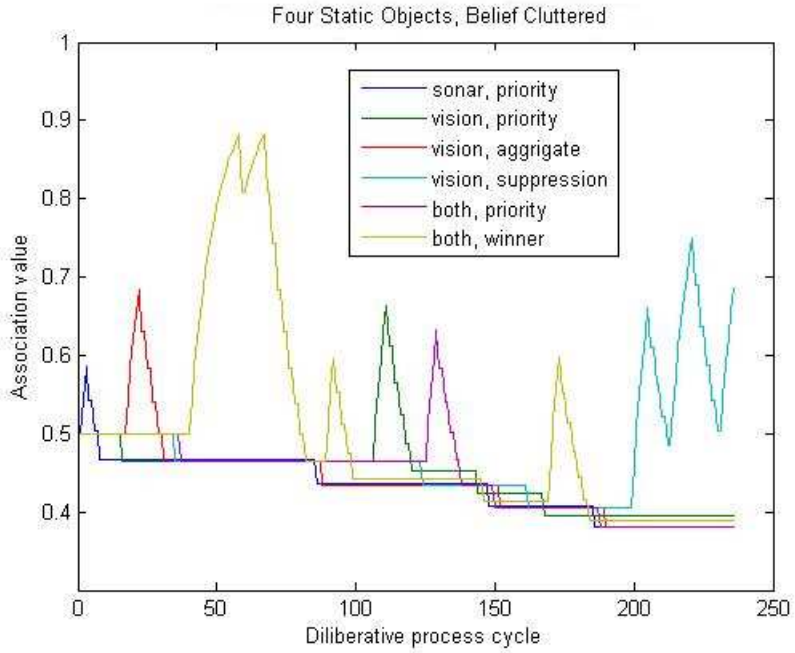

Three Dynamic Objects, Belief Dynamic
sonar, priority
sonar, winner
vision, priority
vision, suppression
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140

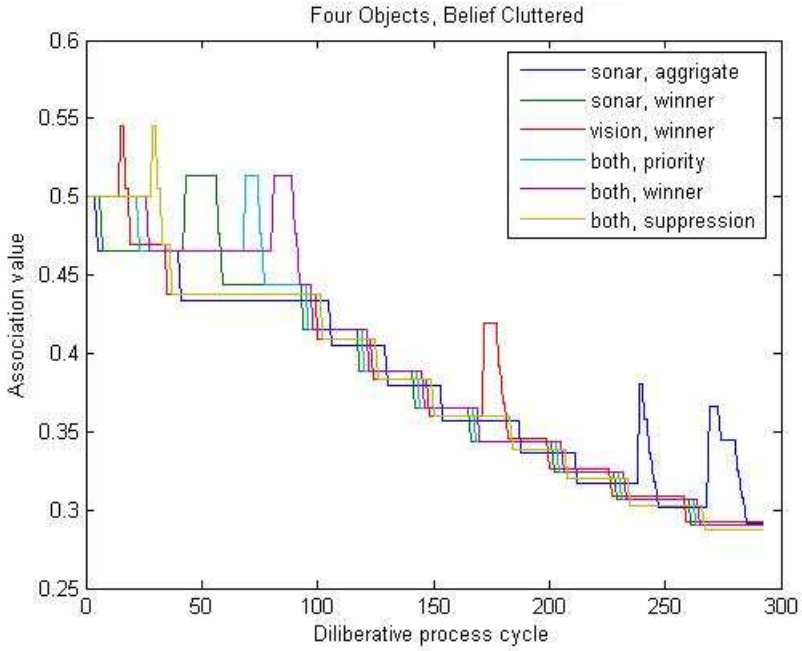

Three Dynamic Objects, Belief Dynamic
sonar, priority
sonar, aggrigate
vision, winner
both, priority
both, aggrigate
both, winner
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300

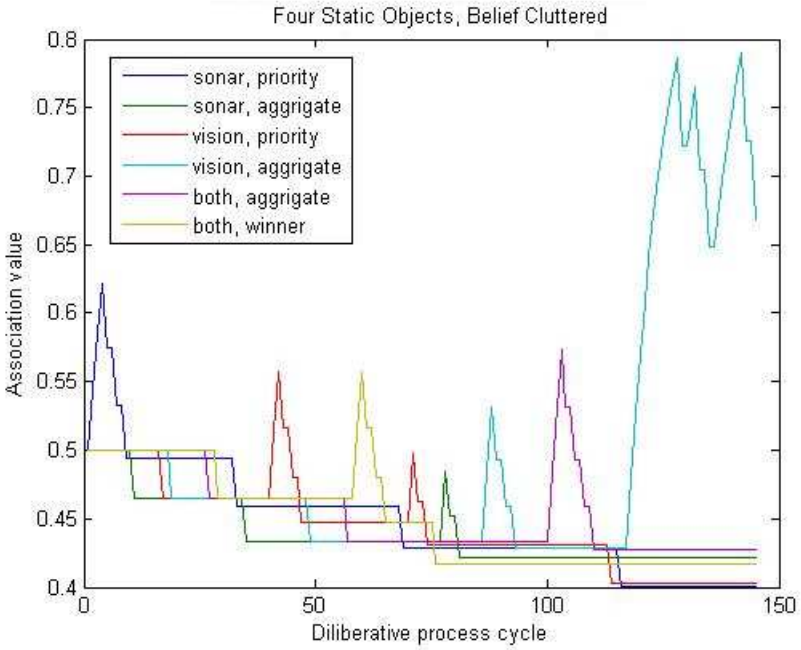

Three Dynamic Objects, Belief Dynamic
sonar, priority
sonar, suppression
vision, aggrigate
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160

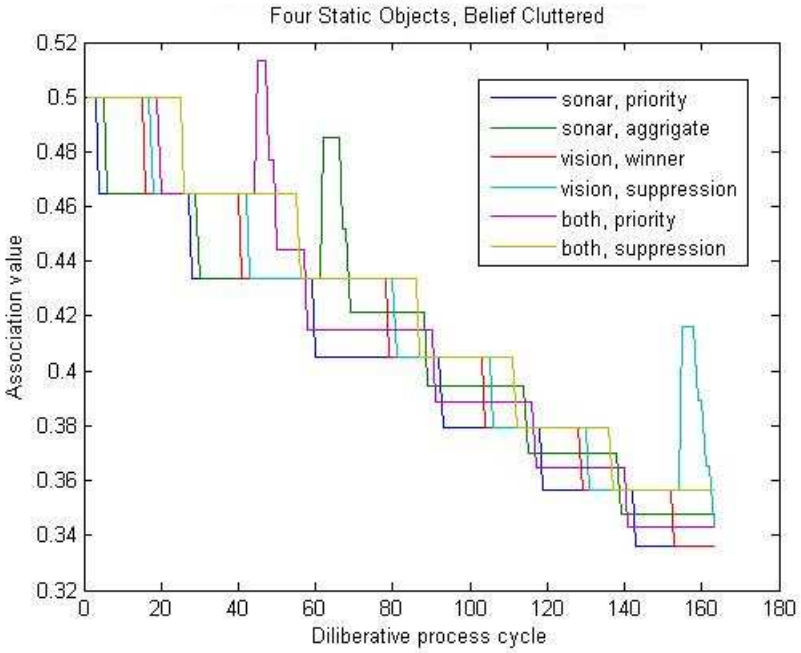

Three Dynamic Objects, Belief Dynamic
sonar, priority
sonar, winner
vision, priority
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

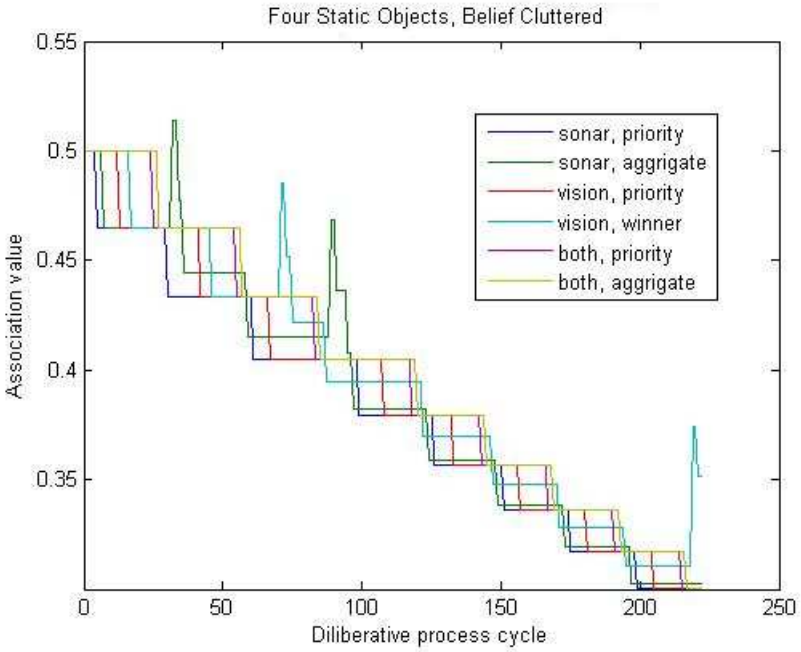
C.3.8 Four Objects, Static Environment

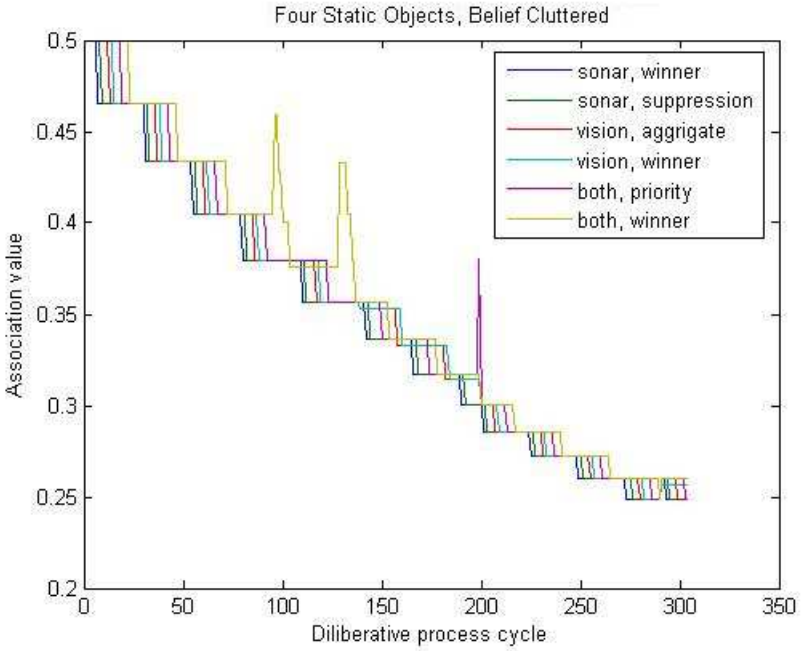


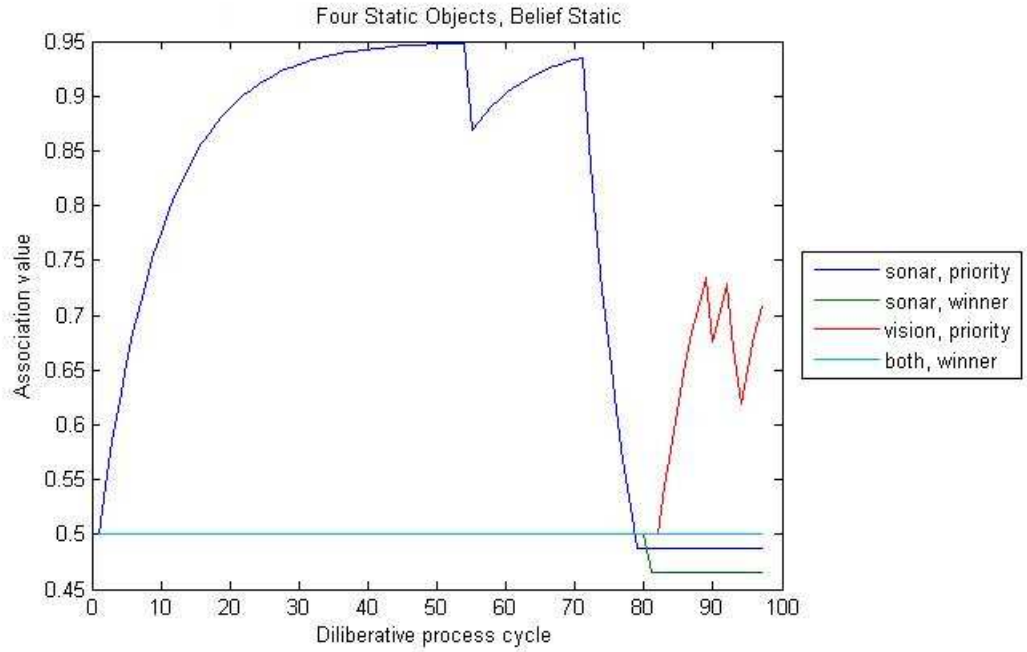


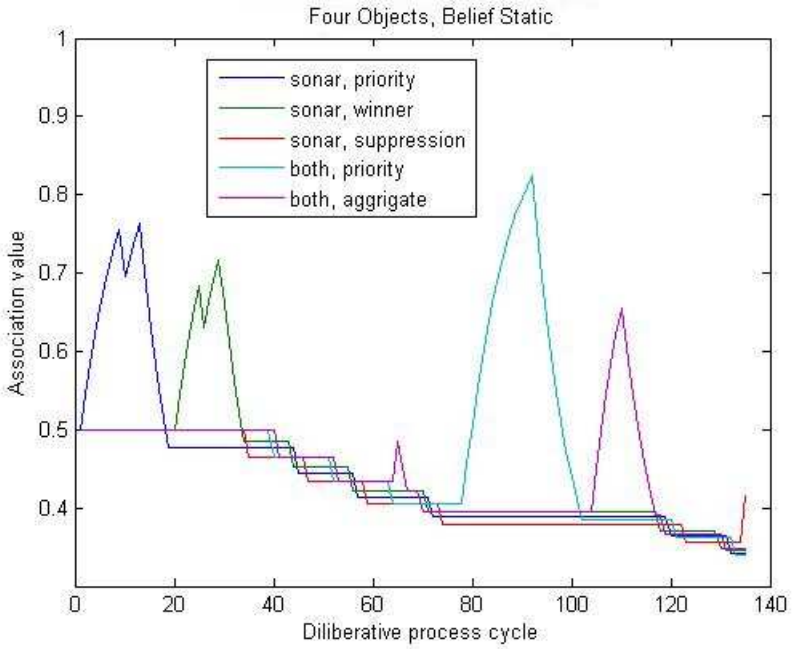

Four Static Objects, Belief Cluttered
sonar, priority
sonar, aggrigate
vision, priority
vision, aggrigate
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
50
100
150

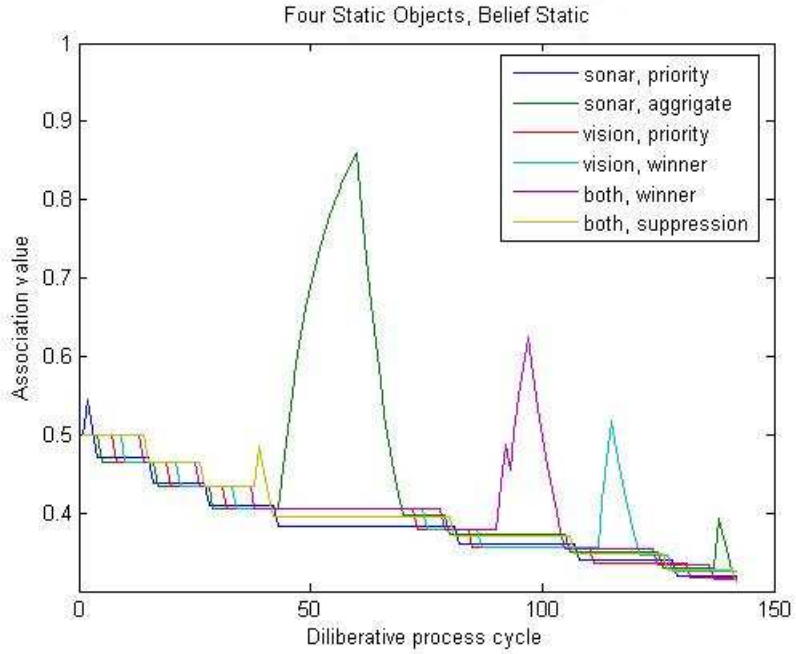

Four Static Objects, Belief Cluttered
sonar, priority
sonar, aggrigate
vision, winner
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0.32
0
20
40
60
80
100
120
140
160
180

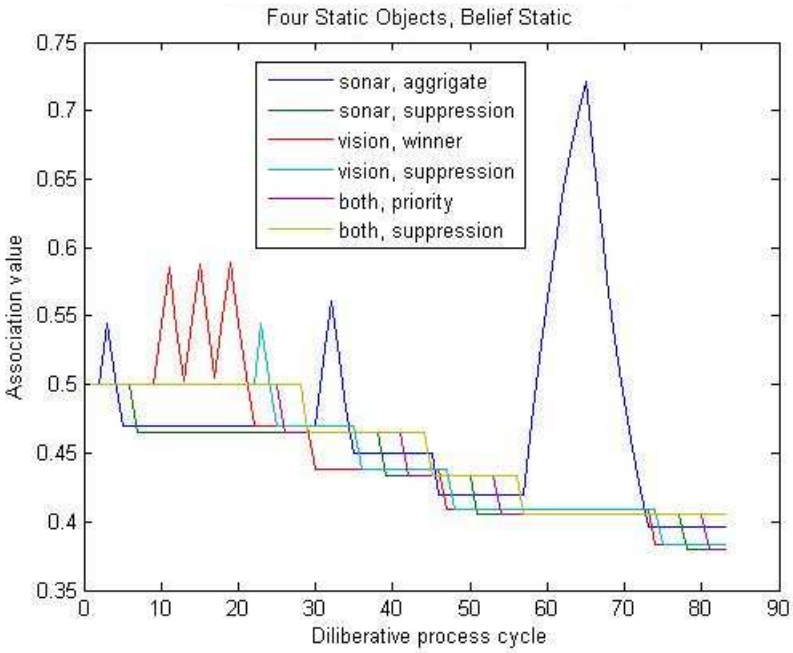

Four Static Objects, Belief Cluttered
sonar, priority
sonar, aggrigate
vision, priority
vision, winner
both, priority
both, aggrigate
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0
50
100
150
200
250

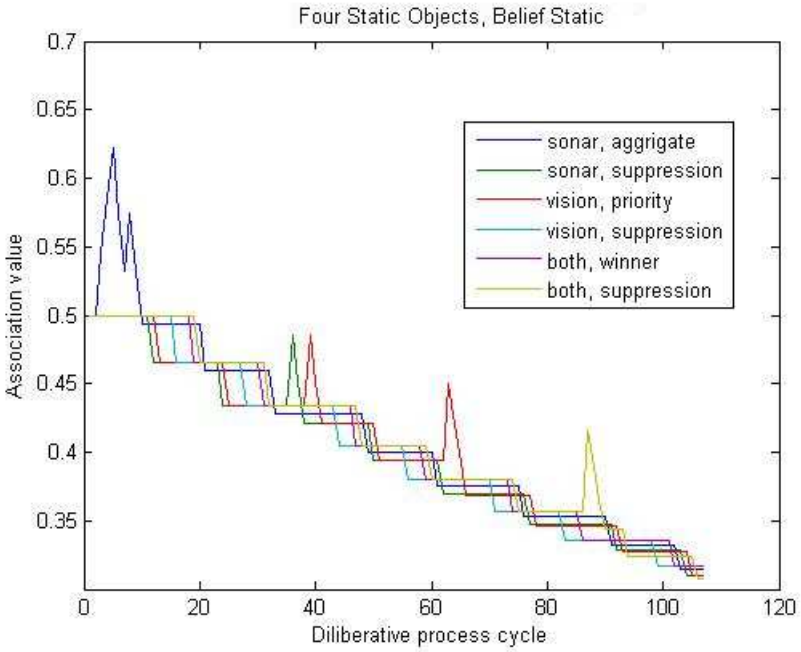

Four Static Objects, Belief Cluttered
sonar, winner
sonar, suppression
vision, aggrigate
vision, winner
both, priority
both, winner
Association value
Diliberative process cycle
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350

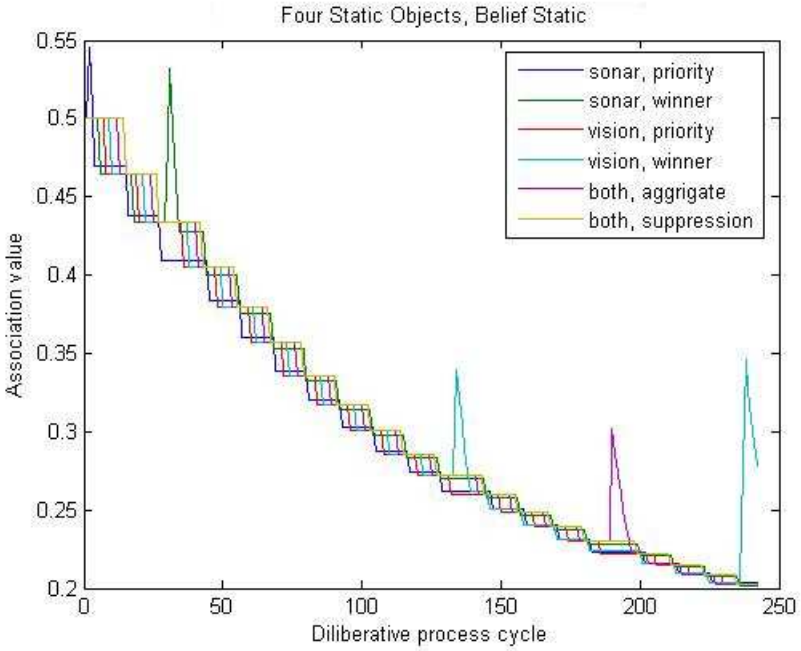

Four Static Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, winner
vision, priority
both, winner

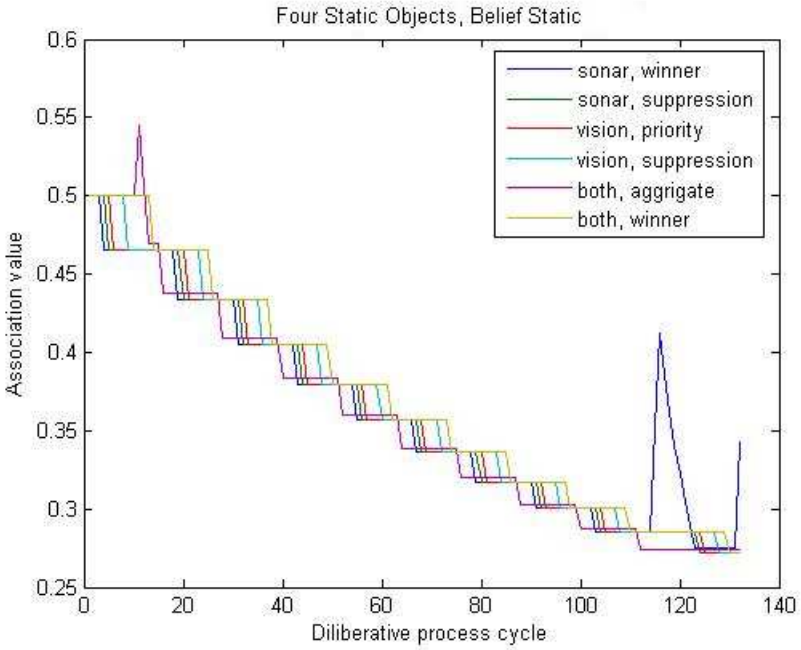

Four Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, winner
sonar, suppression
both, priority
both, aggrigate


Four Static Objects, Belief Static
sonar, priority
sonar, aggrigate
vision, priority
vision, winner
both, winner
both, suppression
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150

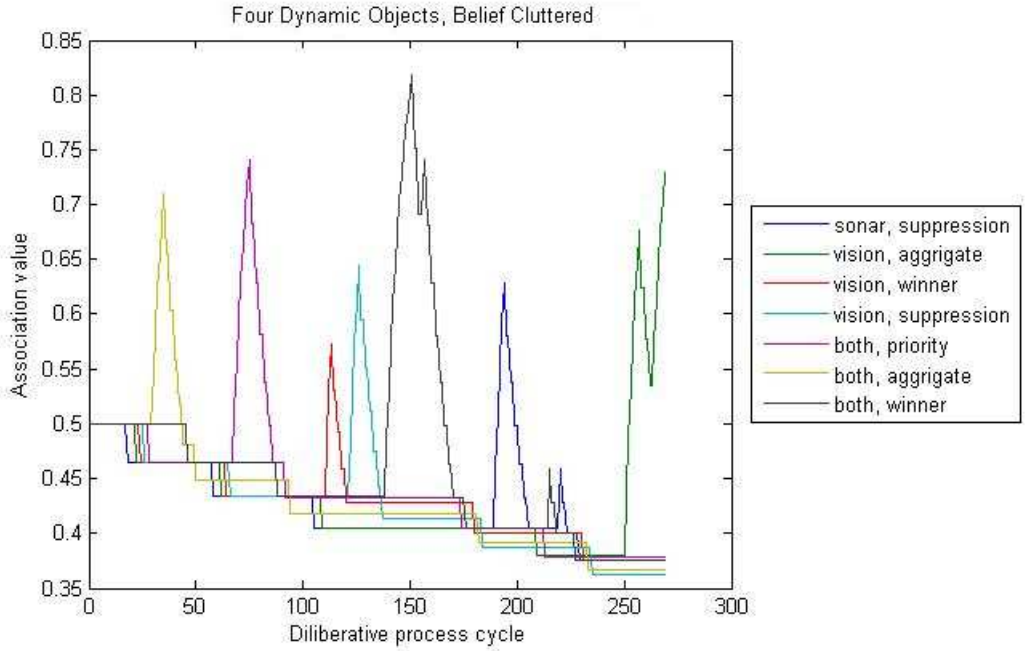

Four Static Objects, Belief Static
sonar, aggrigate
sonar, suppression
vision, winner
vision, suppression
both, priority
both, suppression
Association value
Diliberative process cycle
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
10
20
30
40
50
60
70
80
90

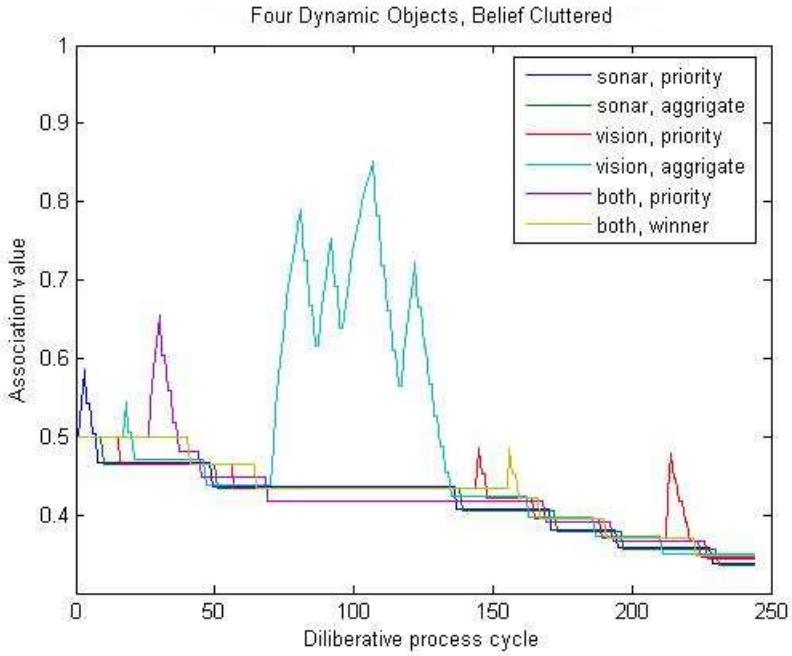

Four Static Objects, Belief Static
Association value
Diliberative process cycle
sonar, aggrigate
sonar, suppression
vision, priority
vision, suppression
both, winner
both, suppression
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120

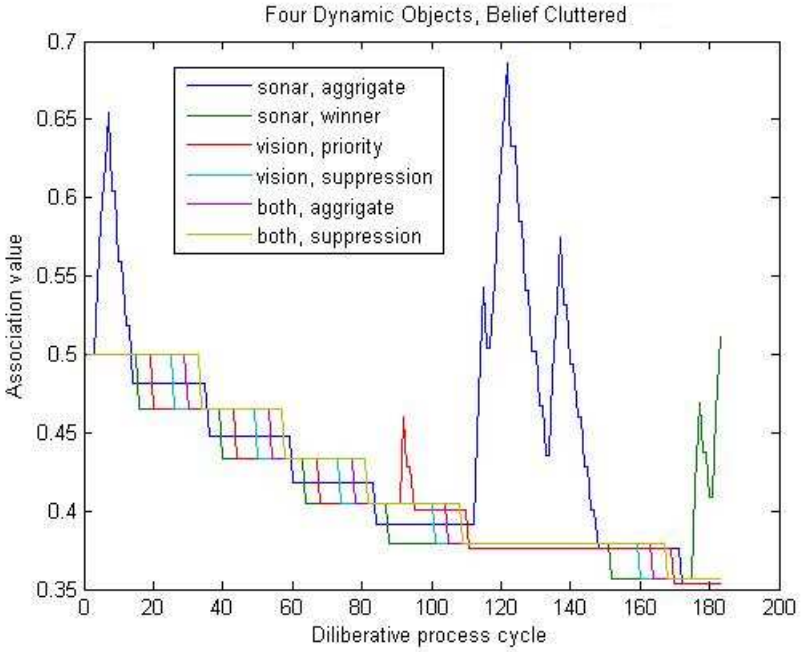

Four Static Objects, Belief Static
Association value
Diliberative process cycle
sonar, priority
sonar, winner
vision, priority
vision, winner
both, aggrigate
both, suppression
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

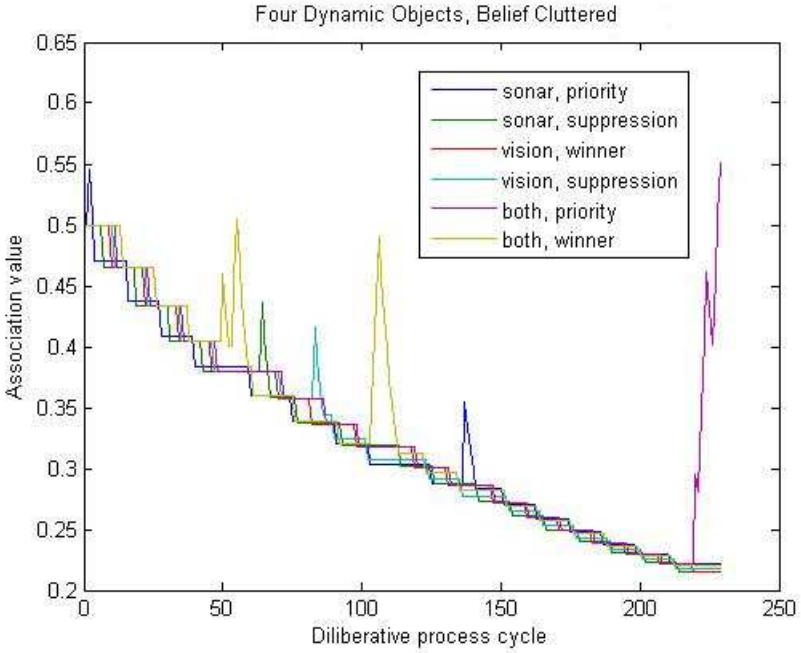

Four Static Objects, Belief Static
sonar, winner
sonar, suppression
vision, priority
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140

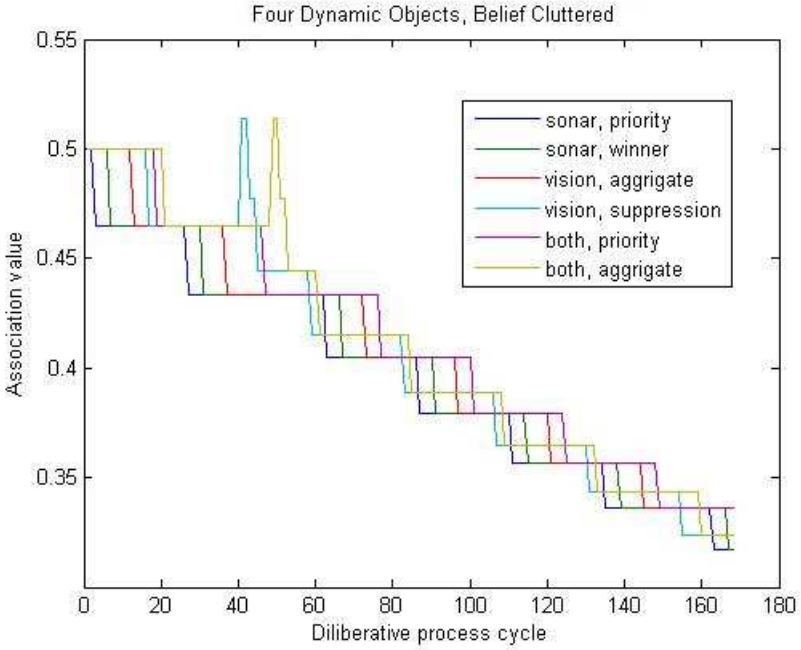
C.3.9 Four Objects, Dynamic Environment

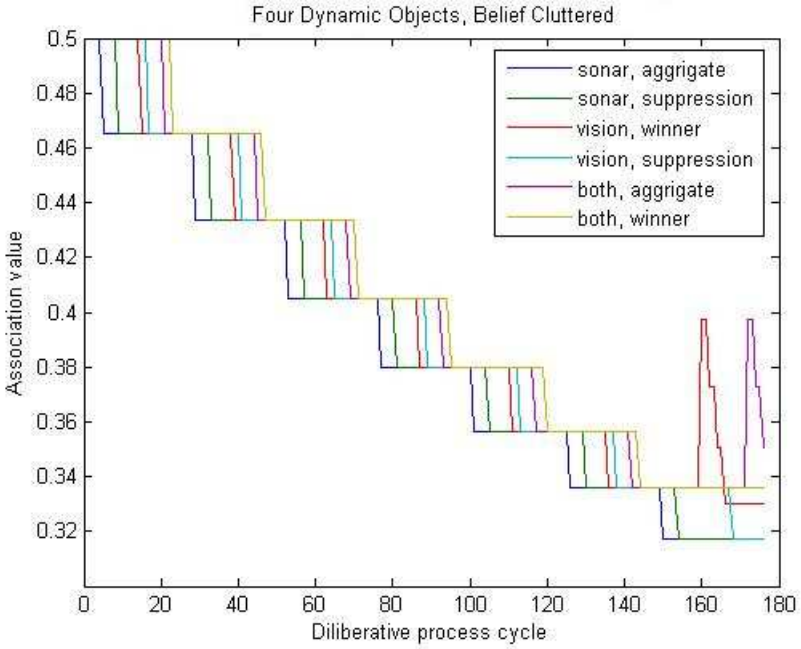


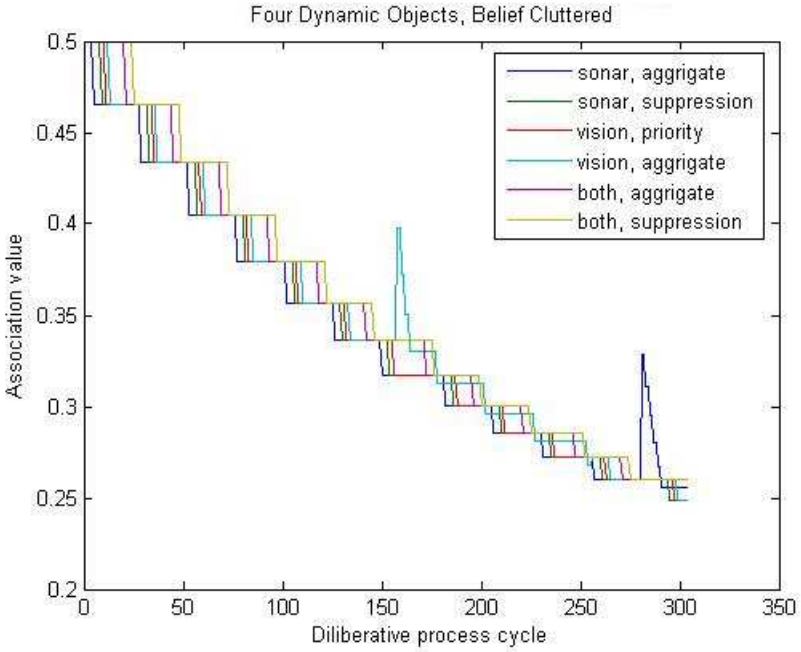


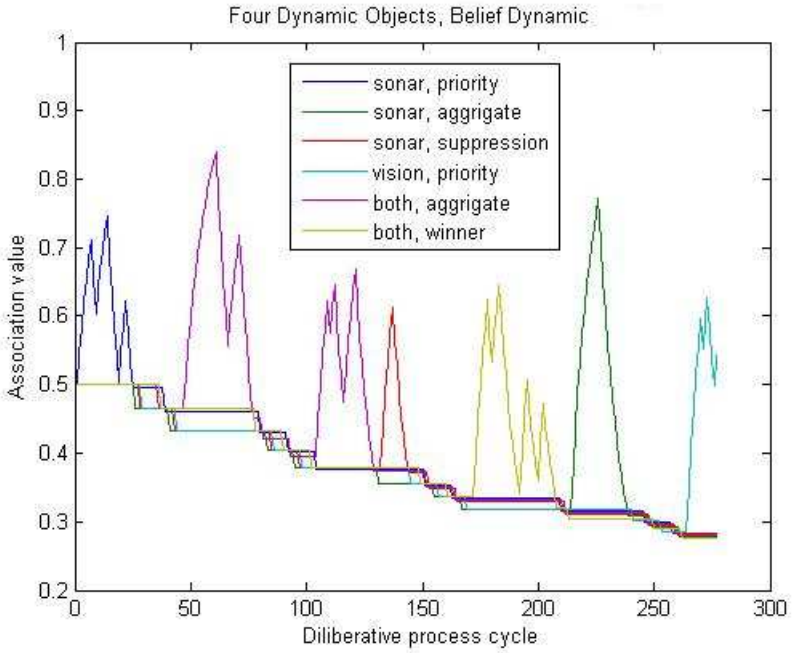

Four Dynamic Objects, Belief Cluttered
sonar, aggrigate
sonar, winner
vision, priority
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180
200

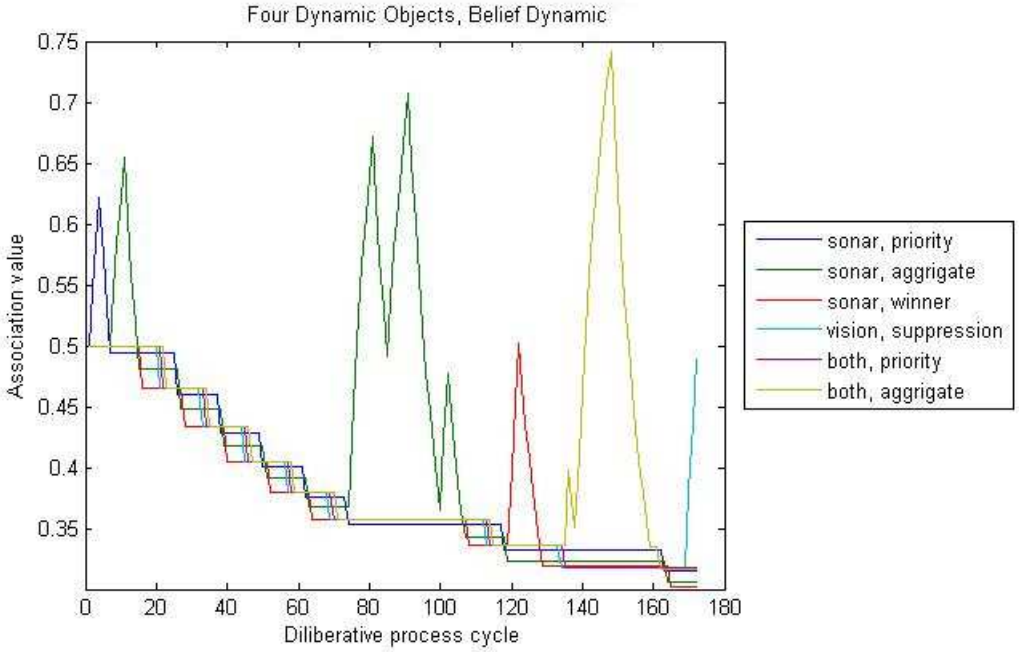

Four Dynamic Objects, Belief Cluttered
sonar, priority
sonar, suppression
vision, winner
vision, suppression
both, priority
both, winner
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250

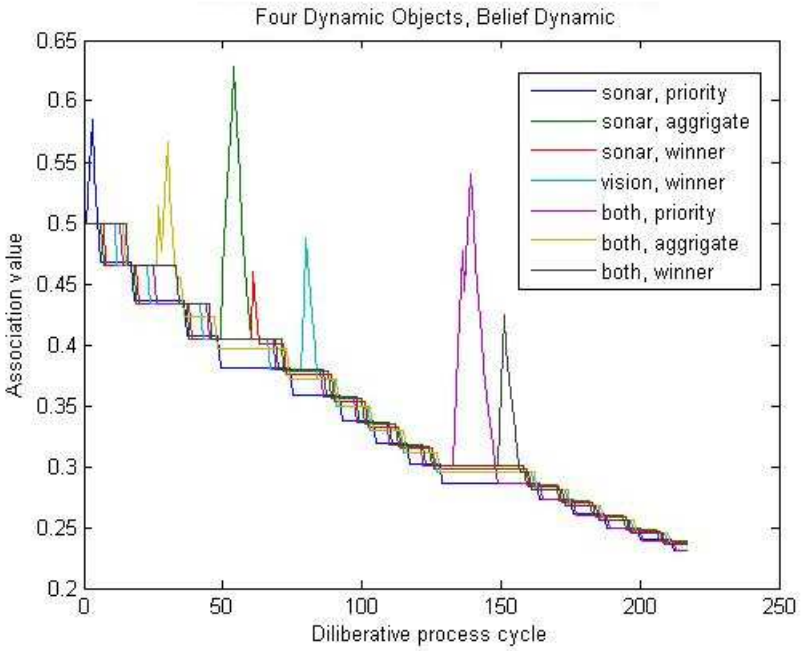

Four Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, priority
sonar, winner
vision, aggrigate
vision, suppression
both, priority
both, aggrigate
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180

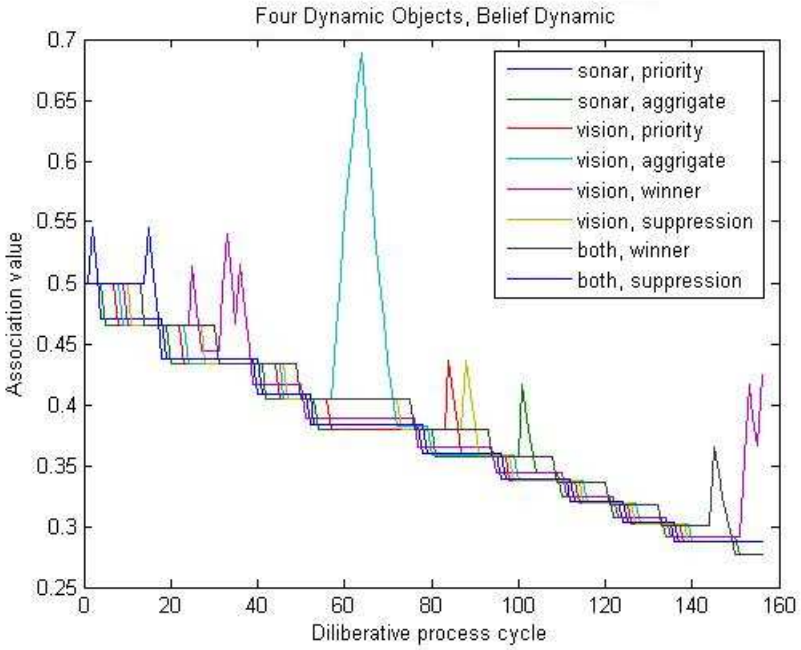

Four Dynamic Objects, Belief Cluttered
Association value
Diliberative process cycle
sonar, aggrigate
sonar, suppression
vision, winner
vision, suppression
both, aggrigate
both, winner
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0.32
0
20
40
60
80
100
120
140
160
180

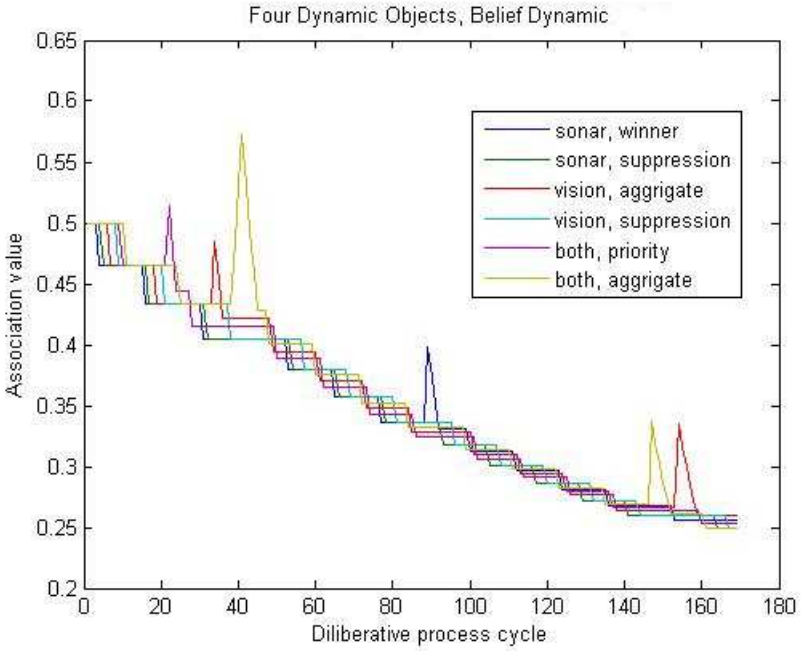

Four Dynamic Objects, Belief Cluttered
sonar, aggrigate
sonar, suppression
vision, priority
vision, aggrigate
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250
300
350

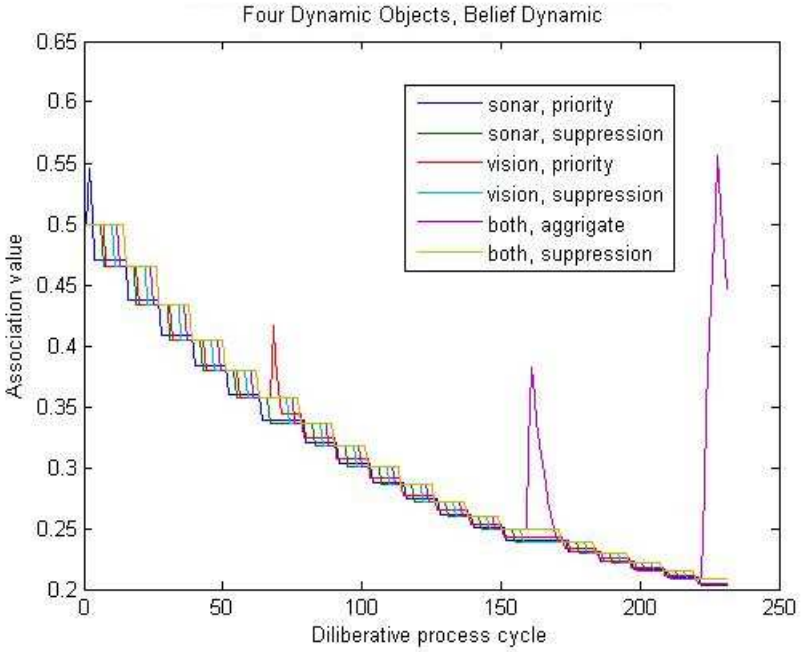

Four Dynamic Objects, Belief Dynamic
sonar, priority
sonar, aggrigate
sonar, suppression
vision, priority
both, aggrigate
both, winner
Association value
Diliberative process cycle
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
50
100
150
200
250
300

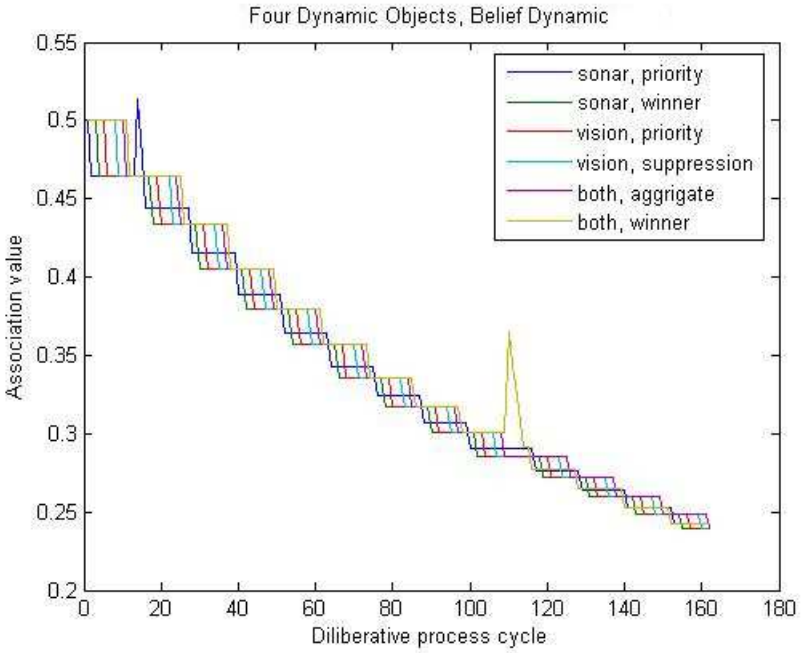

Four Dynamic Objects, Belief Dynamic
Association value
Diliberative process cycle
sonar, priority
sonar, aggrigate
sonar, winner
vision, suppression
both, priority
both, aggrigate


Four Dynamic Objects, Belief Dynamic
Association value
Diliberative process cycle
sonar, priority
sonar, aggrigate
sonar, winner
vision, winner
both, priority
both, aggrigate
both, winner


Four Dynamic Objects, Belief Dynamic
Association value
Diliberative process cycle
sonar, priority
sonar, aggrigate
vision, priority
vision, aggrigate
vision, winner
vision, suppression
both, winner
both, suppression
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0
20
40
60
80
100
120
140
160


Four Dynamic Objects, Belief Dynamic
Association value
Diliberative process cycle
sonar, winner
sonar, suppression
vision, aggrigate
vision, suppression
both, priority
both, aggrigate
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
20
40
60
80
100
120
140
160
180


Four Dynamic Objects, Belief Dynamic
sonar, priority
sonar, suppression
vision, priority
vision, suppression
both, aggrigate
both, suppression
Association value
Diliberative process cycle
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
50
100
150
200
250


Four Dynamic Objects, Belief Dynamic
sonar, priority
sonar, winner
vision, priority
vision, suppression
both, aggrigate
both, winner
Association value
Diliberative process cycle
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0
20
40
60
80
100
120
140
160
180

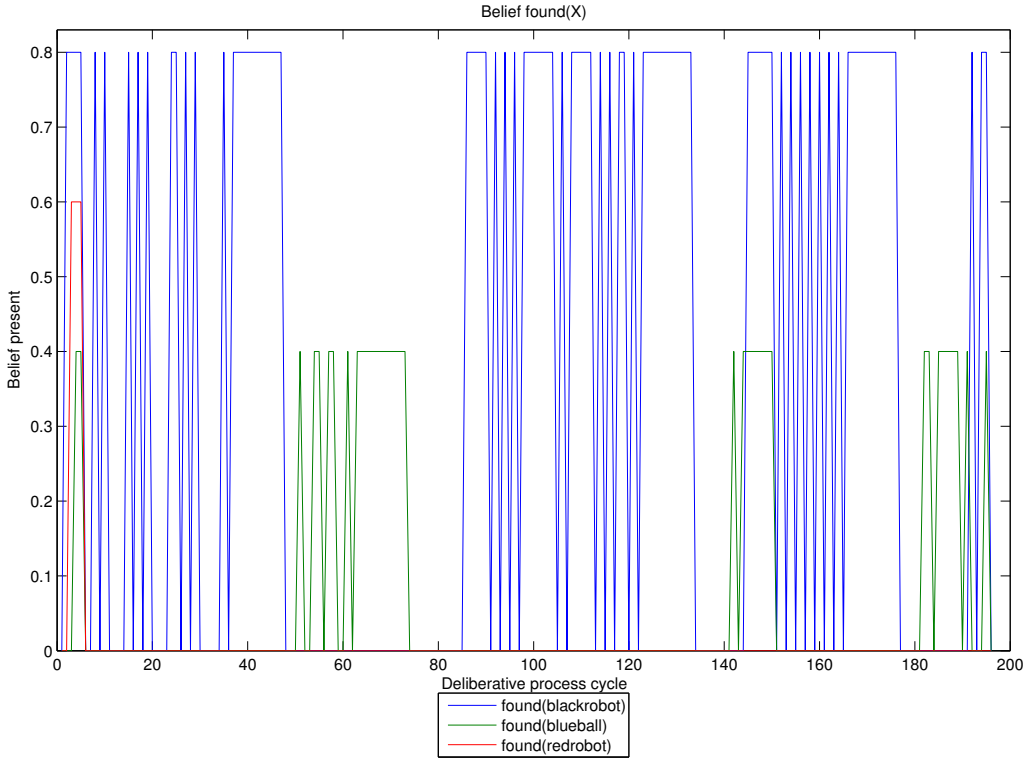
C.4 Adaptation Results

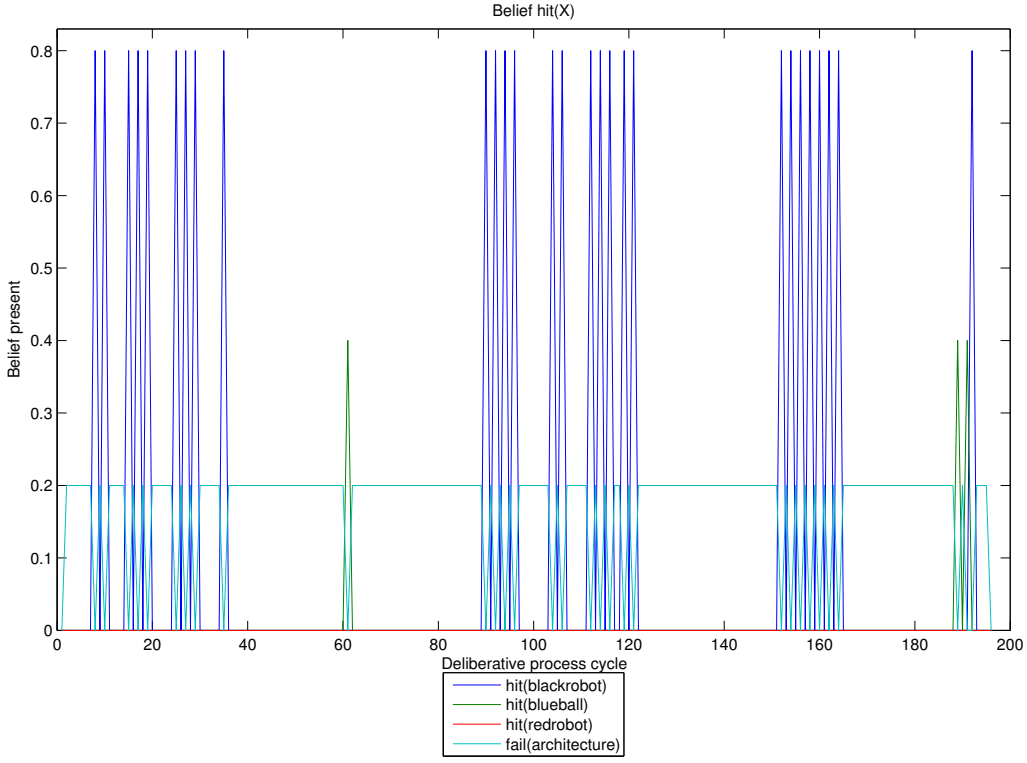
This section presents the results gathered during the adaptation experiment. This is where robo-CAMAL was allowed to roam in a variable environment with multiple goals and know good associations. The results can be read as follows. The graphs representing the agent's beliefs shows when a particular belief is present. If the line representing a belief has a value, then the belief is present at that deliberative processing cycle. The value chosen is arbitrary and has no meaning other than to differentiate between beliefs. The goal importance graphs represent how each goal importance value varied over time, like wise with the association value graphs. The initial paragraph of each subsection details the nature of the environment for that particular experiment.

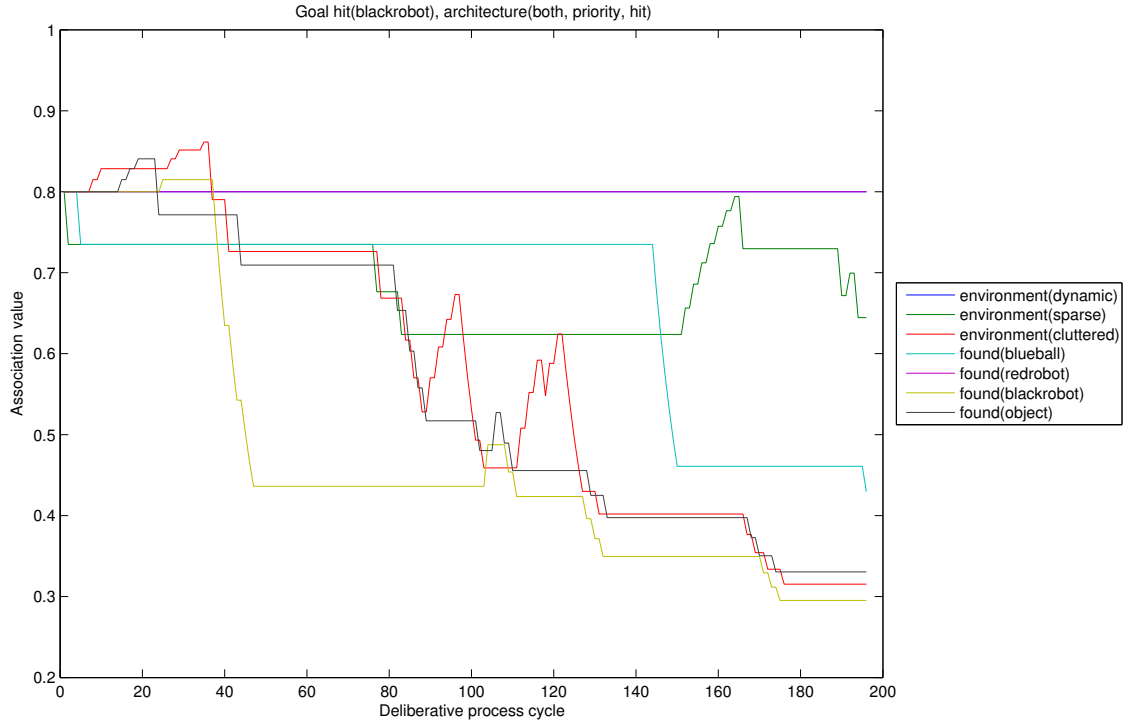
C.4.1 Adaptation Experiment 1

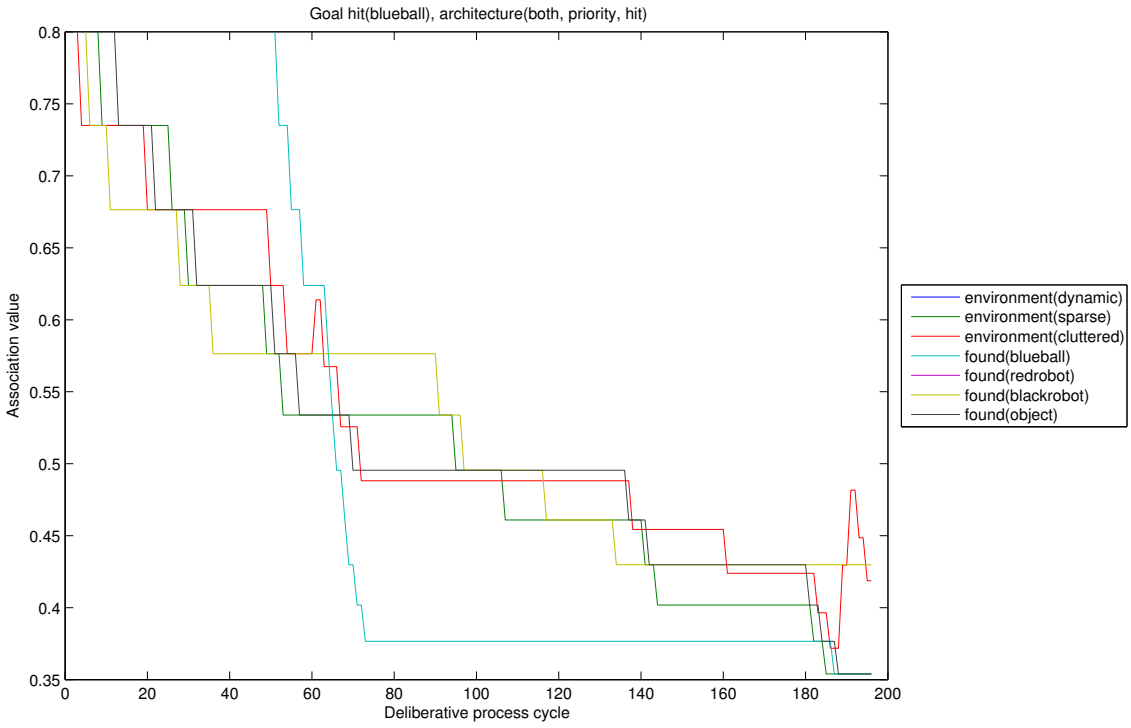
Environment sequence

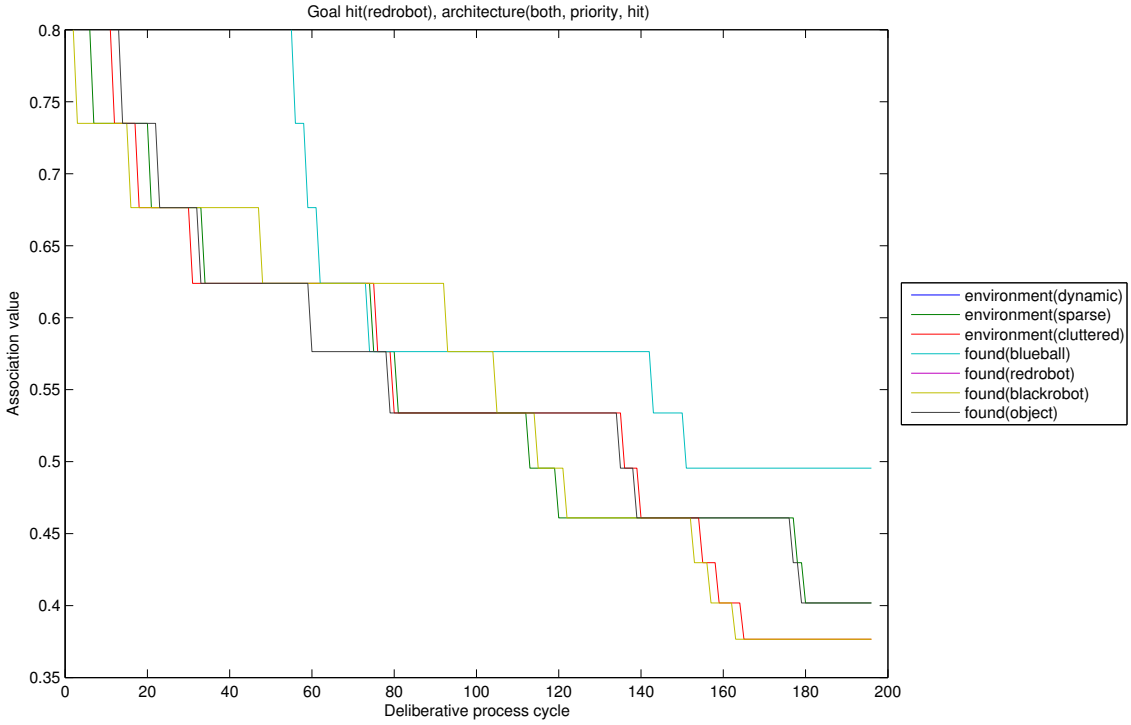
redrobot and *blackrobot*
blueball
blueball

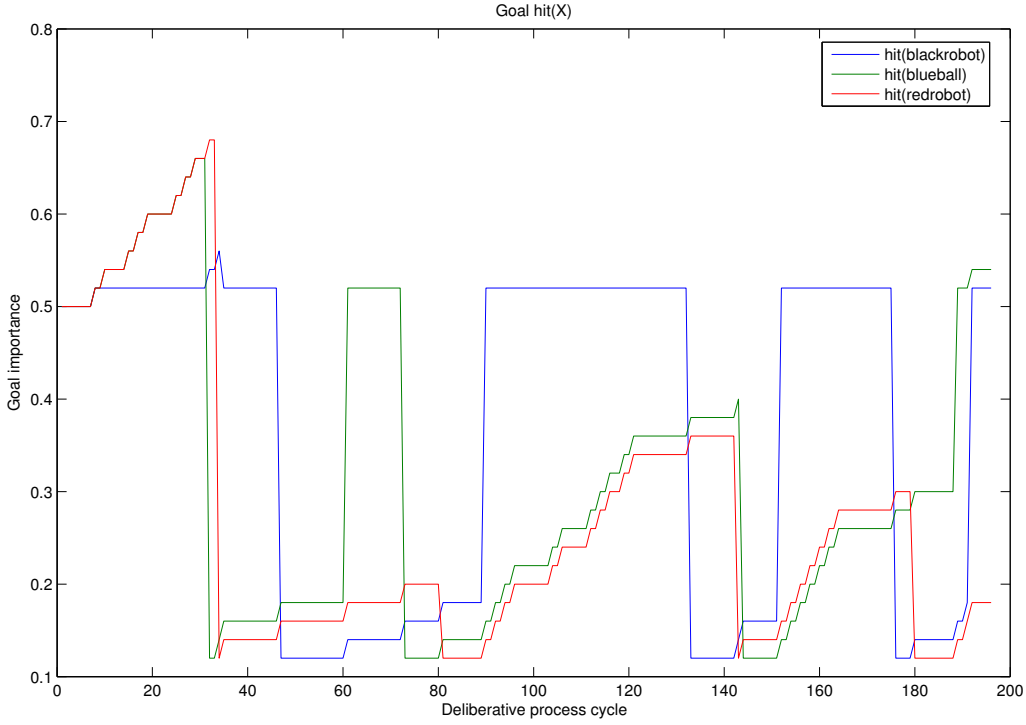

Belief found(X)
Belief present
Deliberative process cycle
found(blackrobot)
found(blueball)
found(redrobot)


Belief hit(X)
Belief present
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)
fail(architecture)


Goal hit(blackrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(blueball), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(redrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(X)
Goal importance
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)

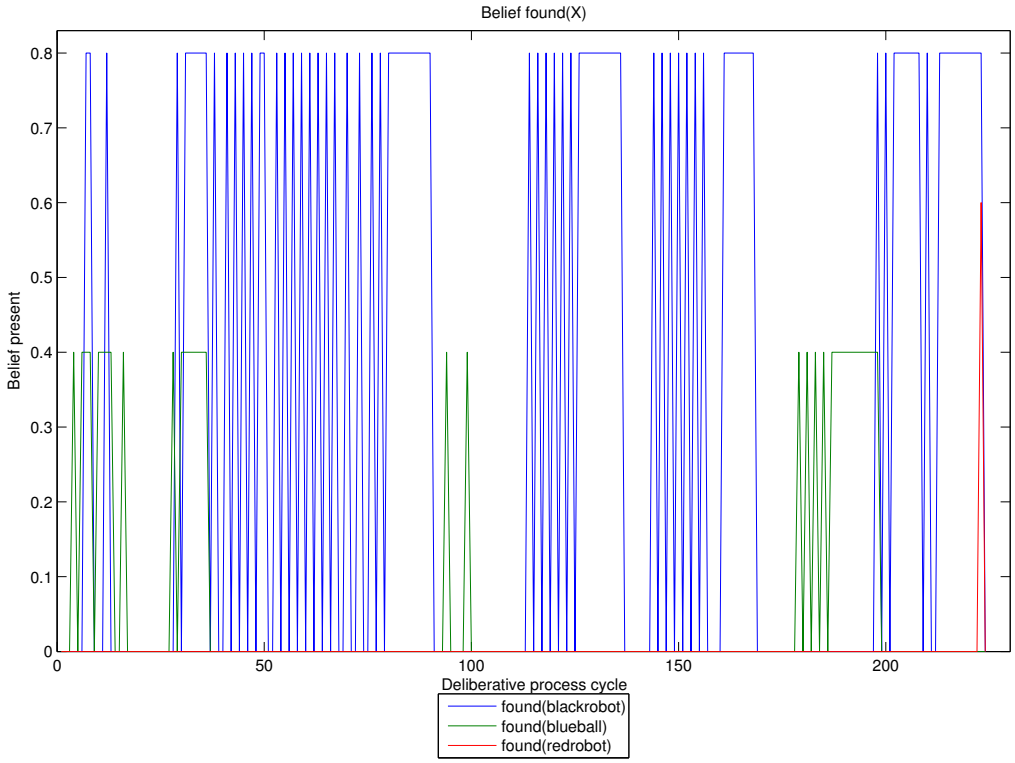
C.4.2 Adaptation Experiment 2

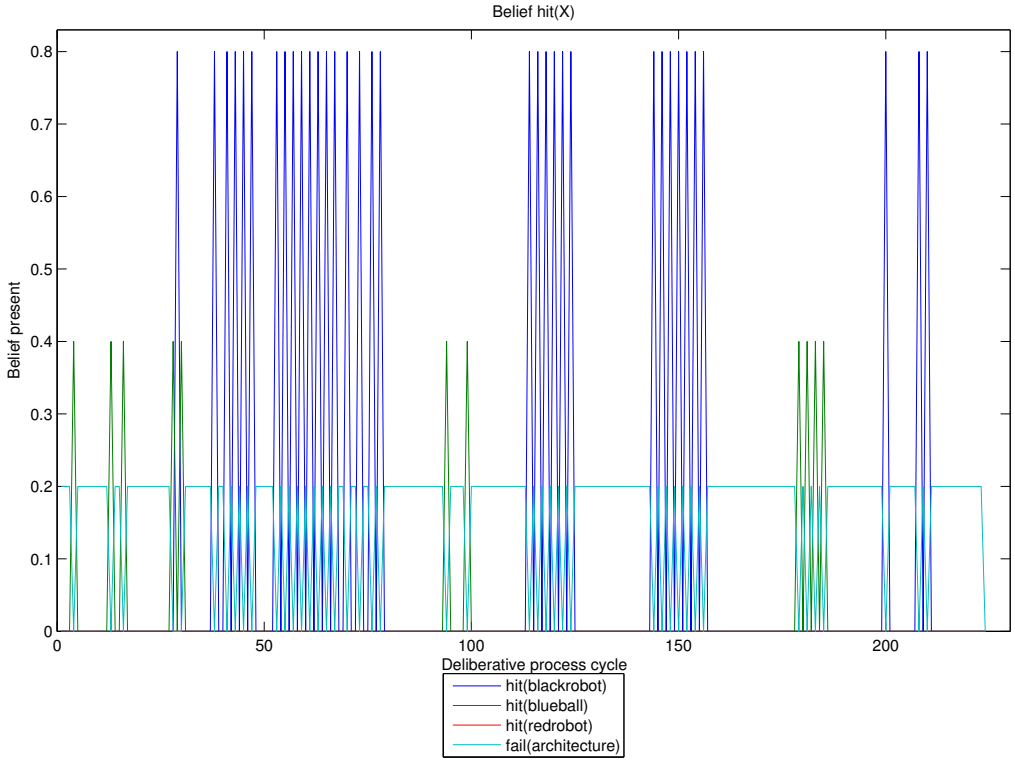
Environment sequence

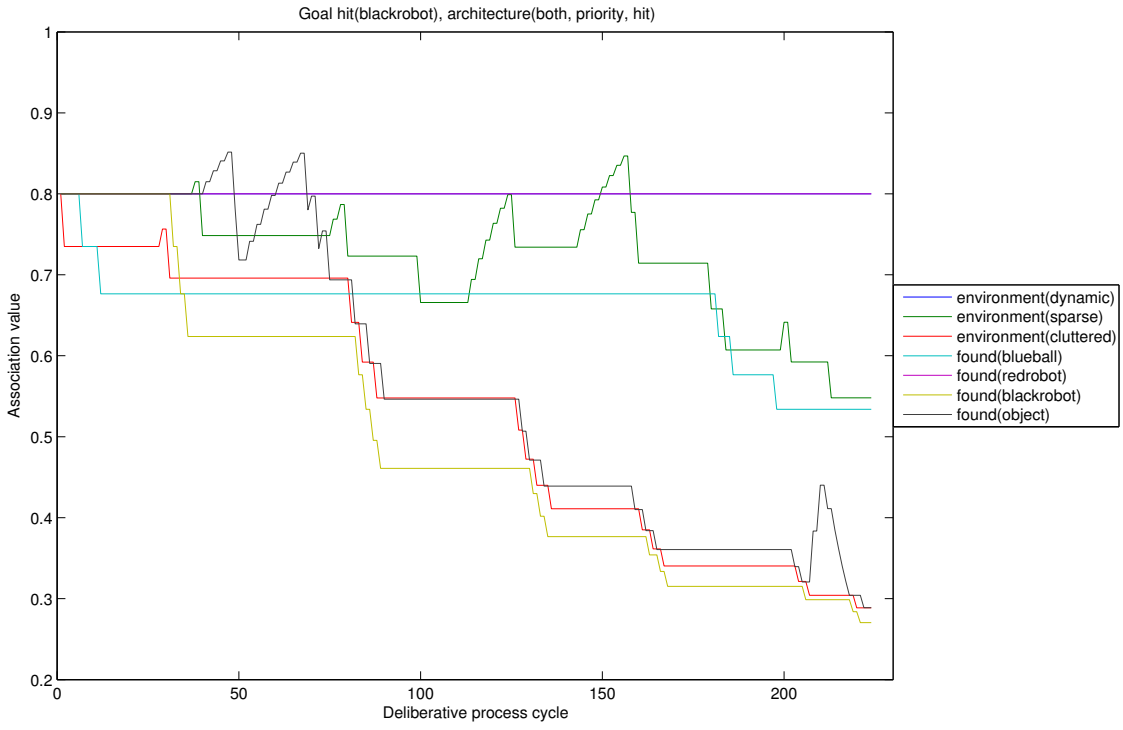
redrobot and *blackrobot*

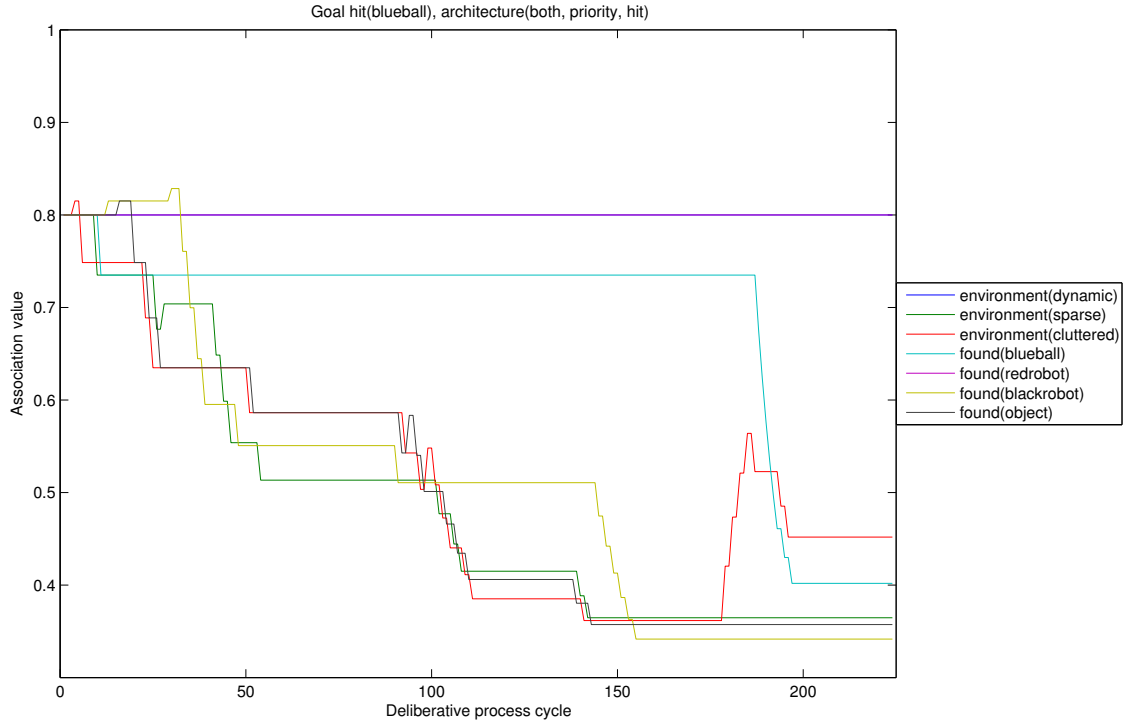
blueball and *blackrobot*

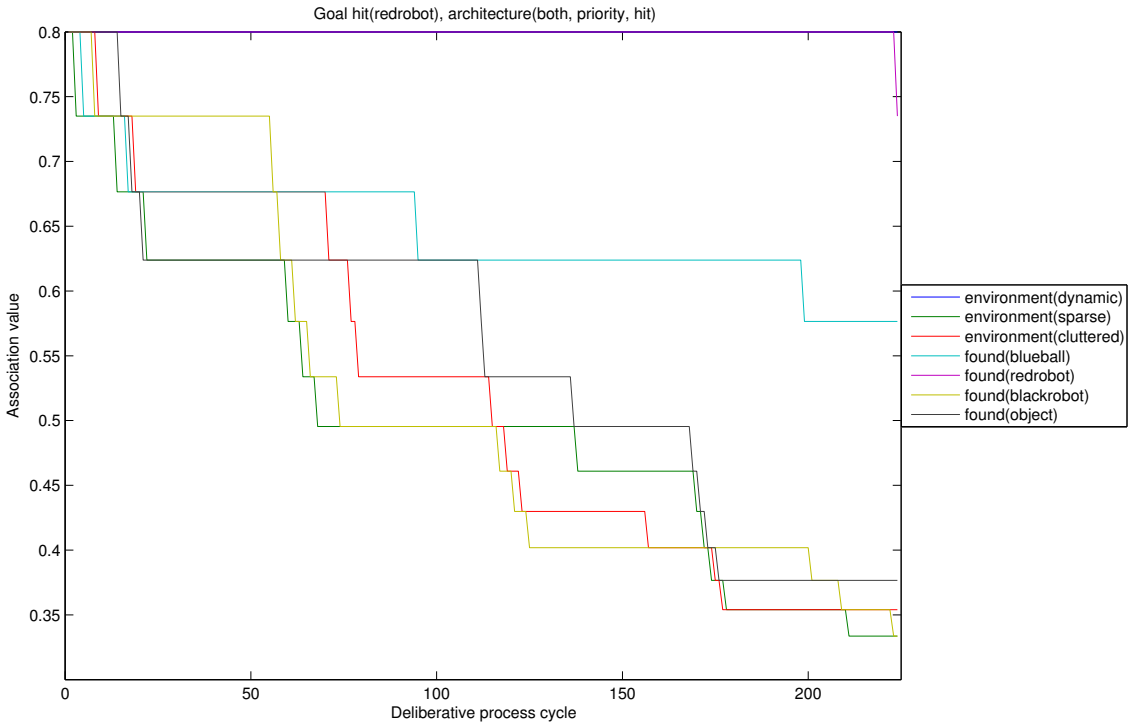
blackrobot

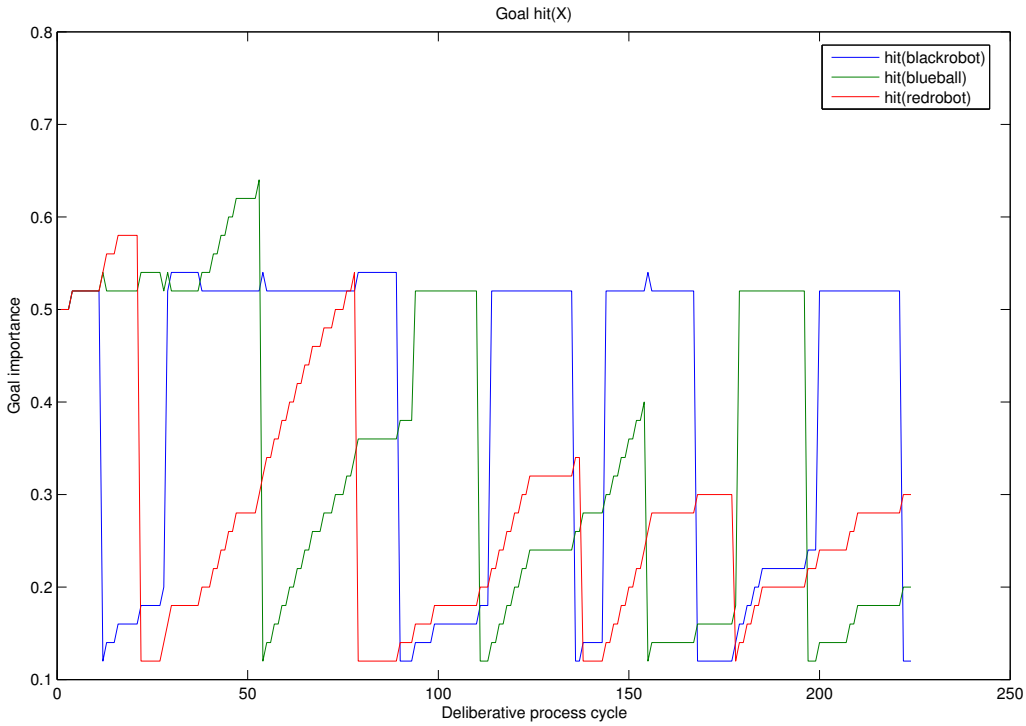



Belief hit(X)
Belief present
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)
fail(architecture)


Goal hit(blackrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(blueball), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(redrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(X)
hit(blackrobot)
hit(blueball)
hit(redrobot)
Goal importance
Deliberative process cycle
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250

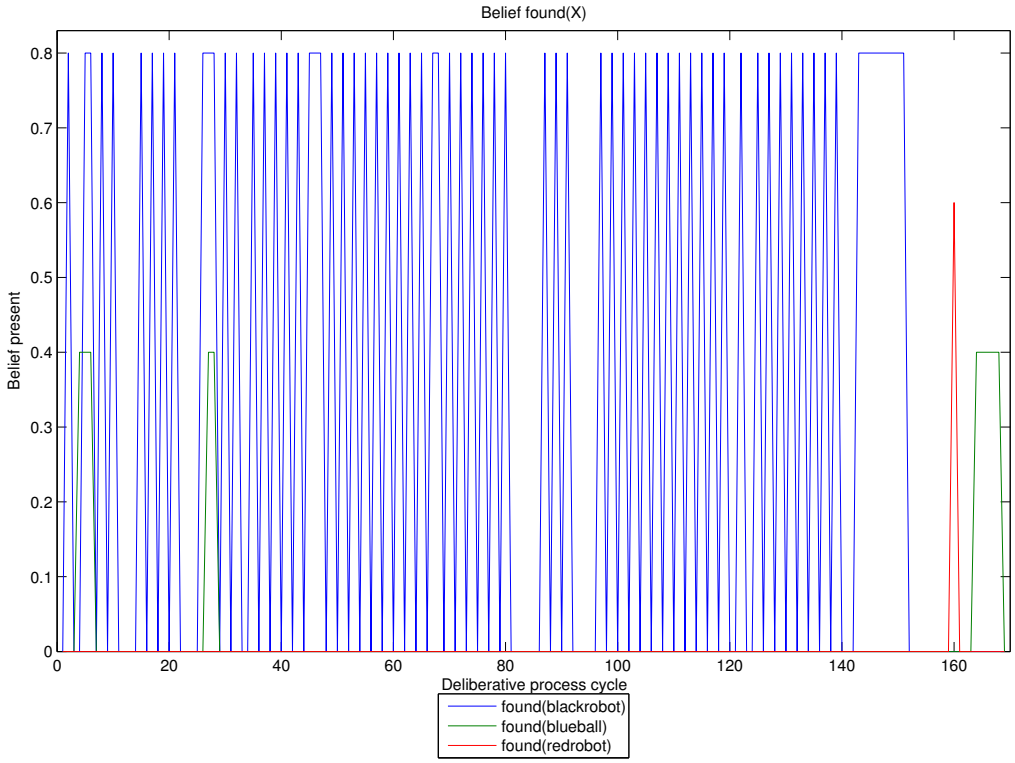
C.4.3 Adaptation Experiment 3

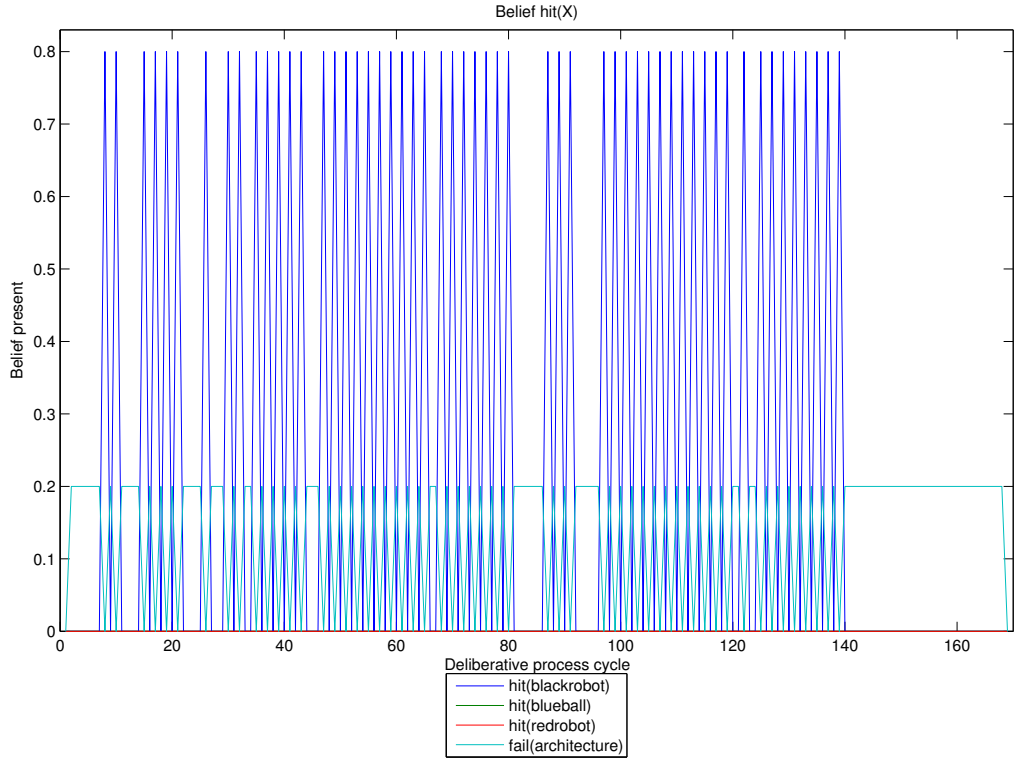
Environment sequence

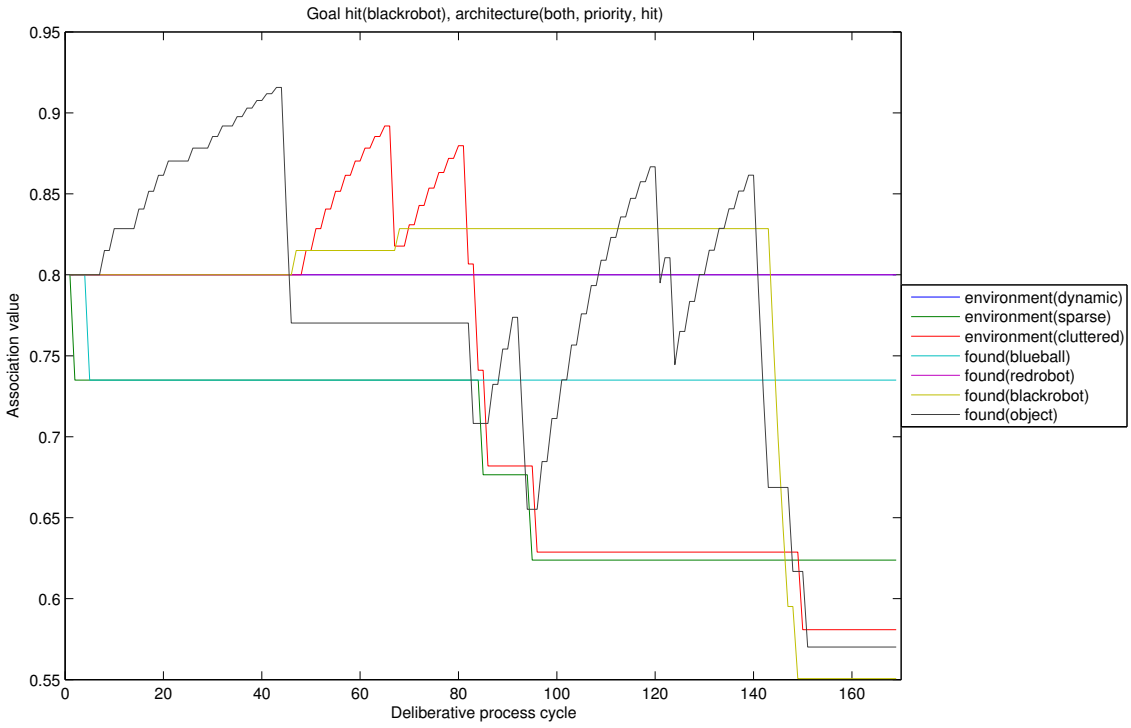
blueball and *blackrobot*

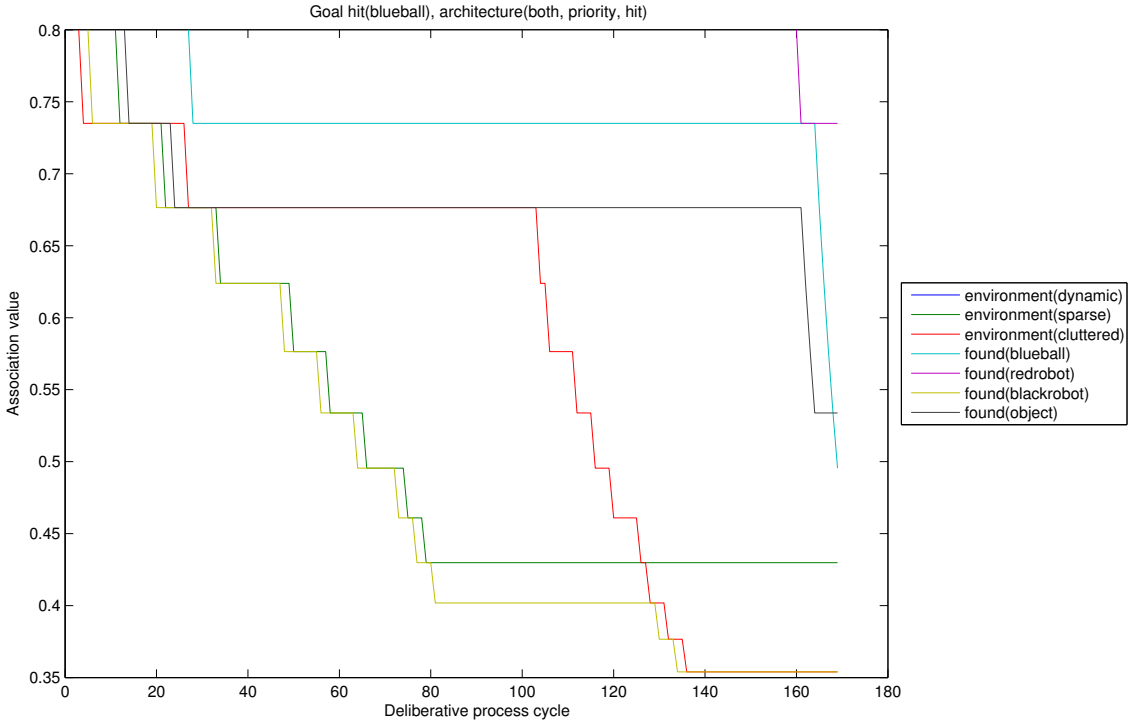
blackrobot

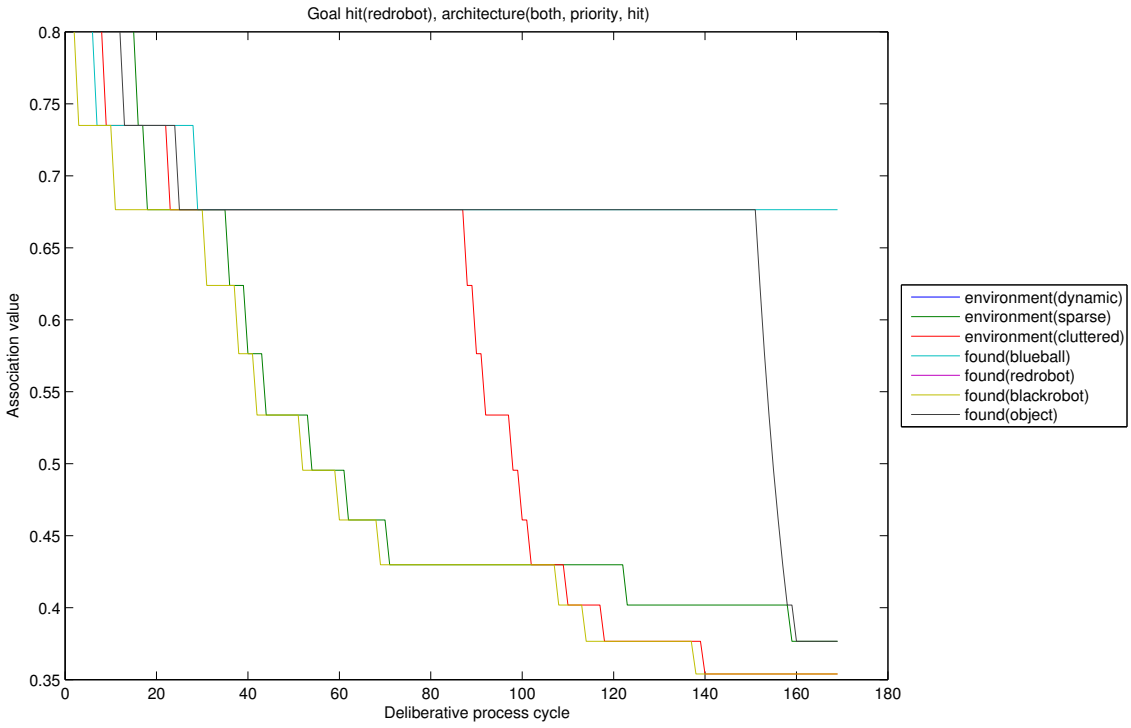
redrobot

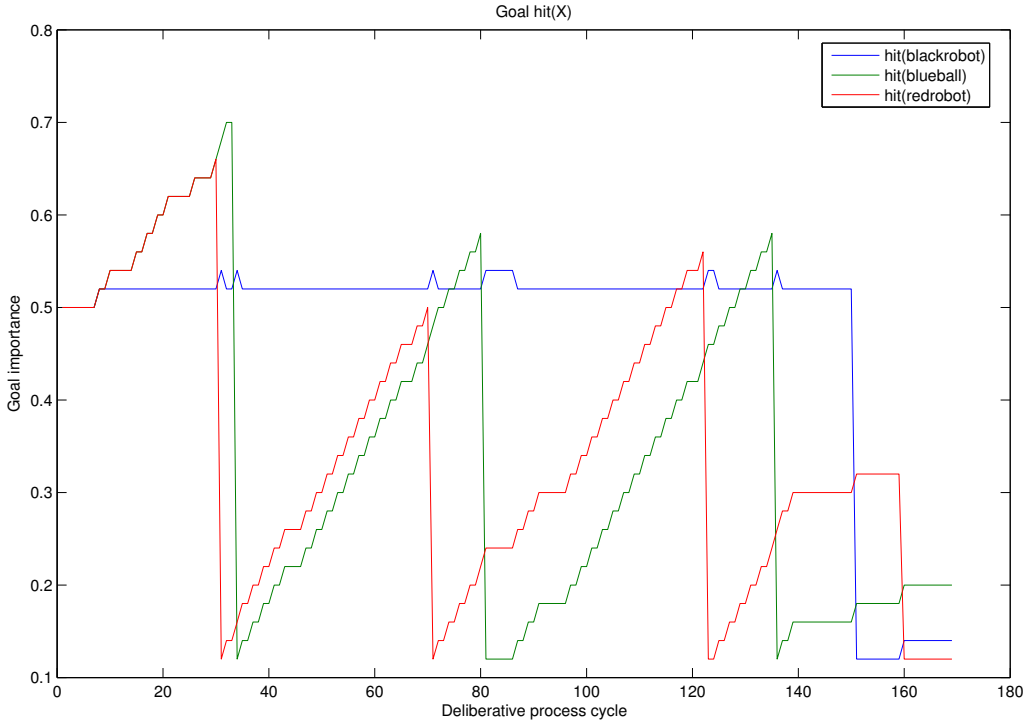



Belief hit(X)
Belief present
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)
fail(architecture)


Goal hit(blackrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(blueball), architecture(both, priority, hit)
Association value
Deliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(redrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0
20
40
60
80
100
120
140
160
180
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(X)
hit(blackrobot)
hit(blueball)
hit(redrobot)
Goal importance
Deliberative process cycle
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
120
140
160
180

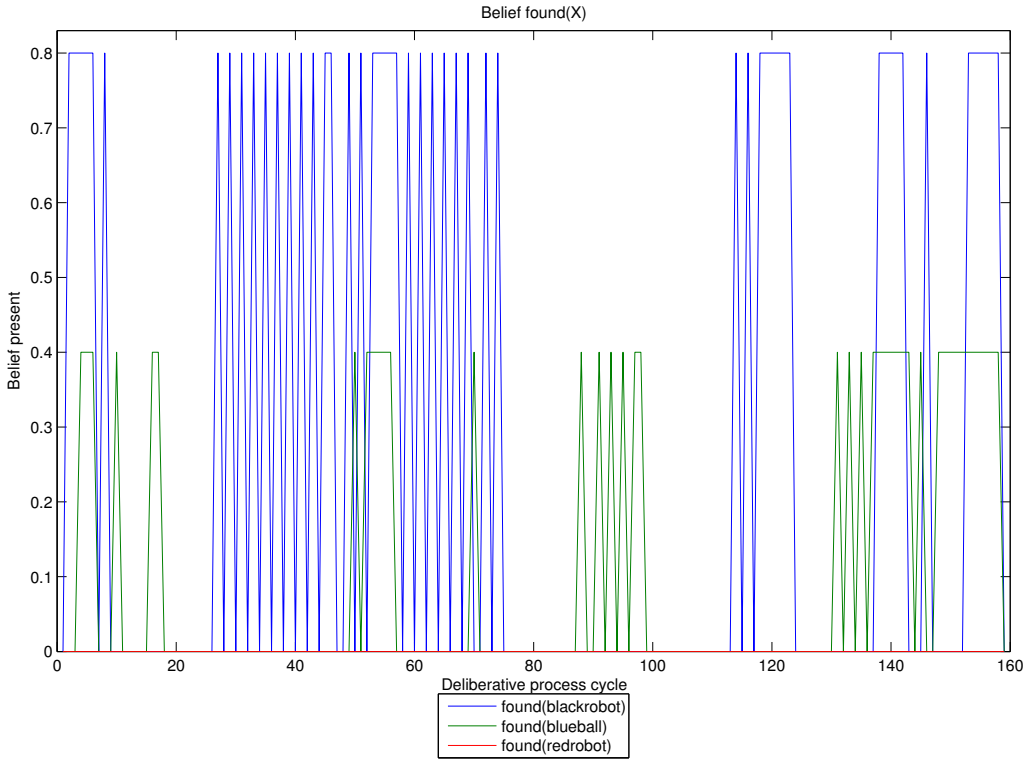
C.4.4 Adaptation Experiment 4

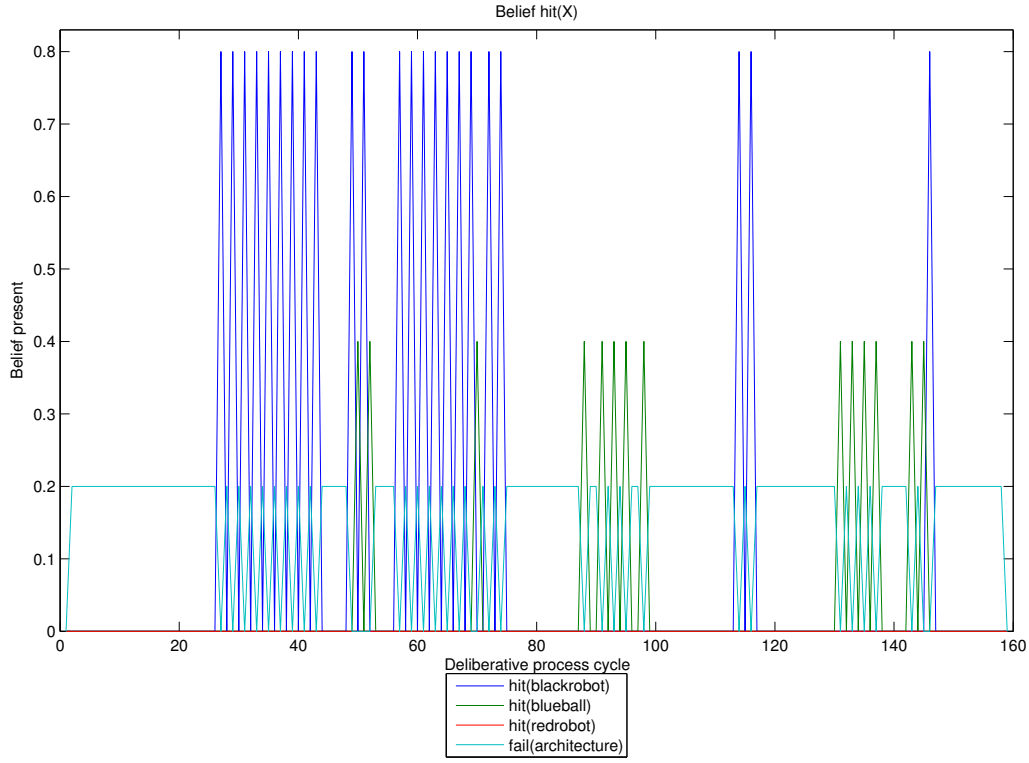
Environment sequence

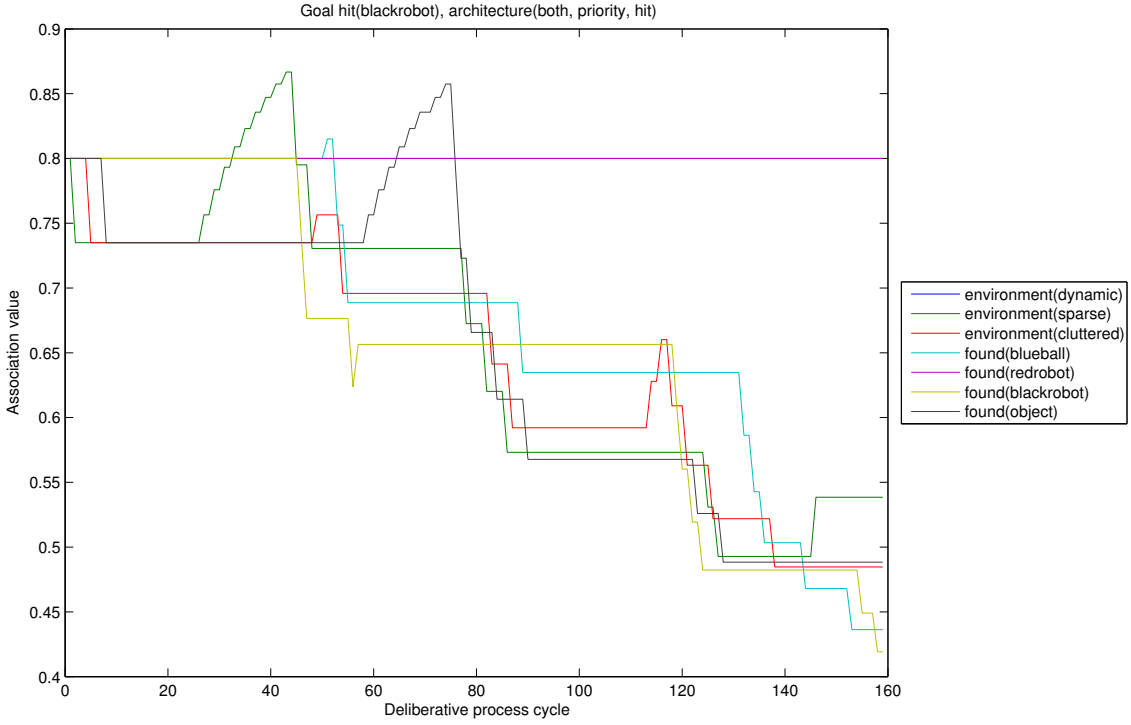
blueball and *blackrobot*

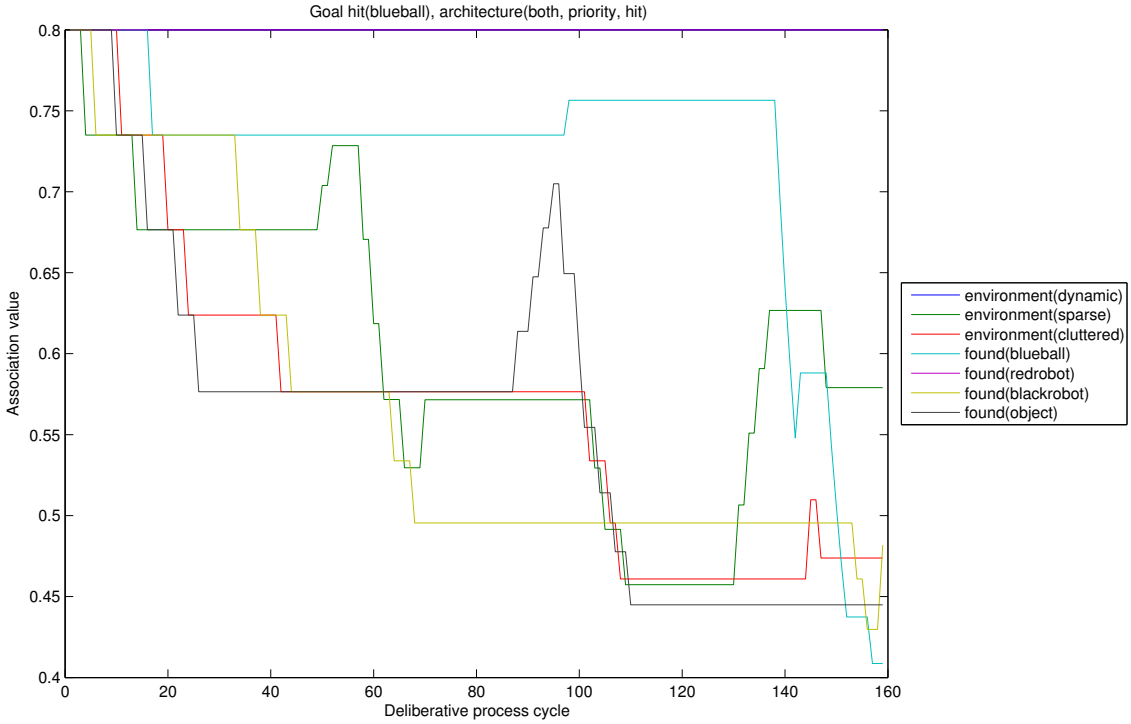
blackrobot

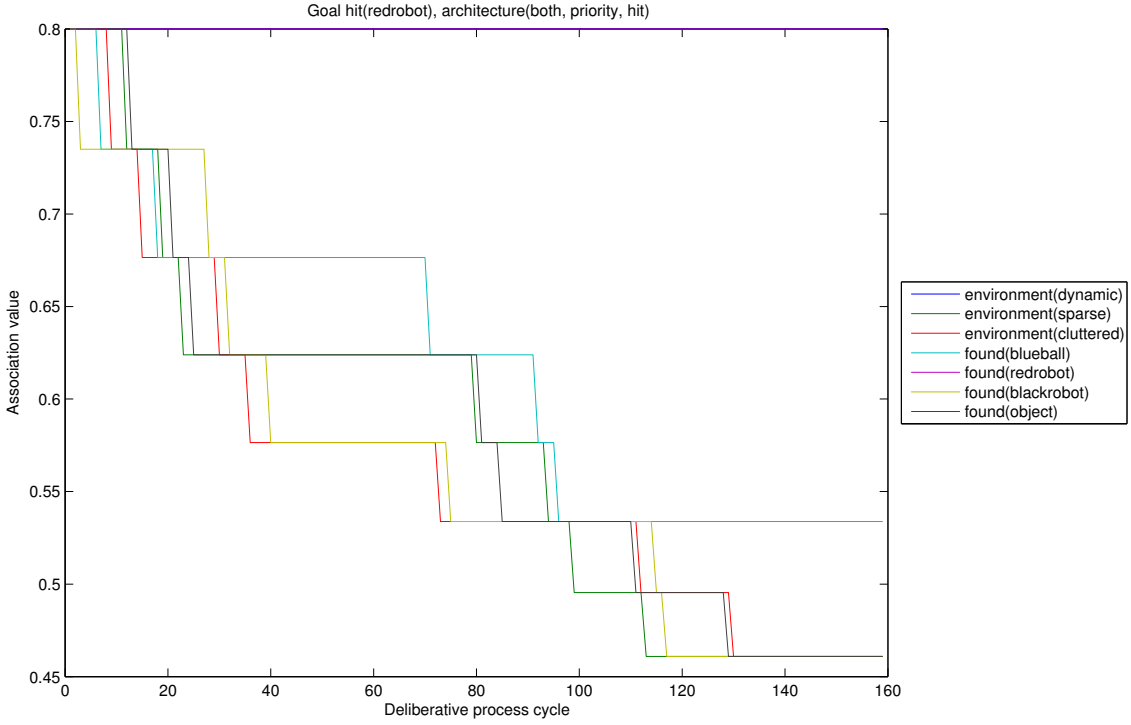
blueball

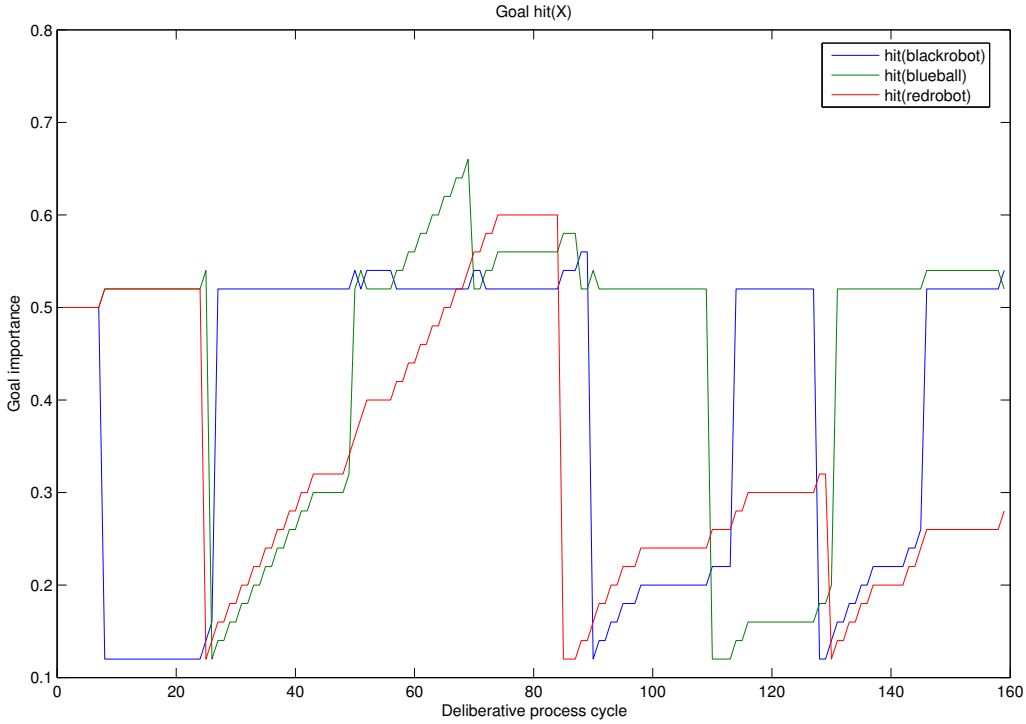



Belief hit(X)
Belief present
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)
fail(architecture)


Goal hit(blackrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(blueball), architecture(both, priority, hit)
Association value
Deliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0
20
40
60
80
100
120
140
160
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(redrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
20
40
60
80
100
120
140
160
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(X)
hit(blackrobot)
hit(blueball)
hit(redrobot)
Goal importance
Deliberative process cycle
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
120
140
160

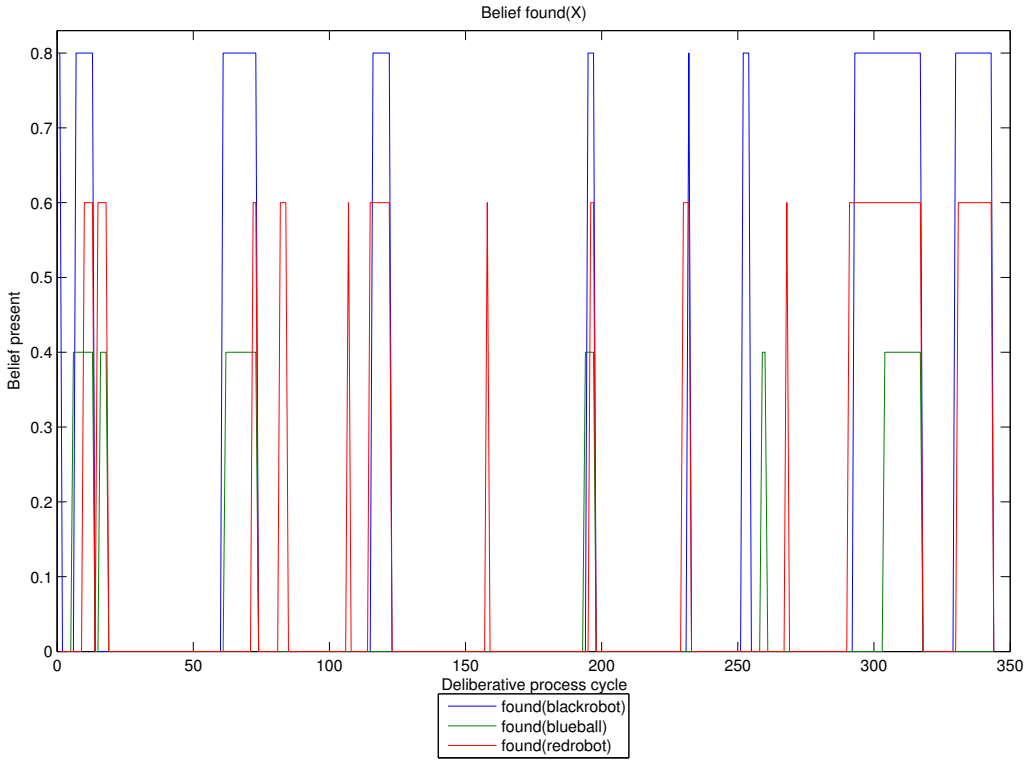
C.4.5 Adaptation Experiment 5

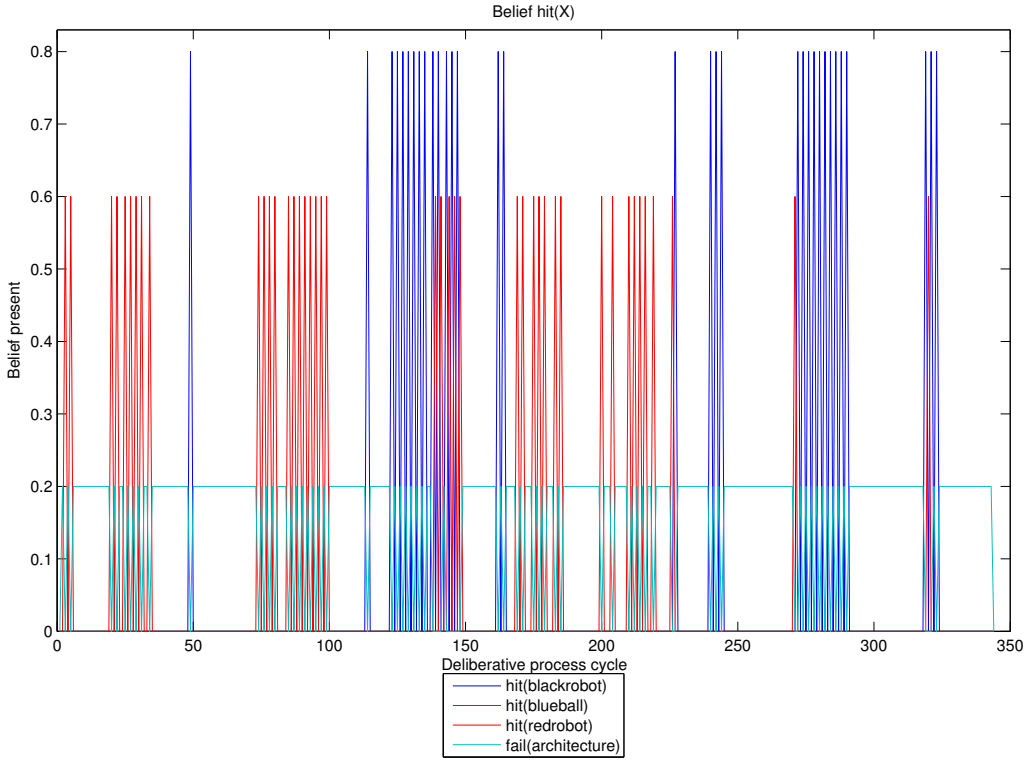
Environment sequence

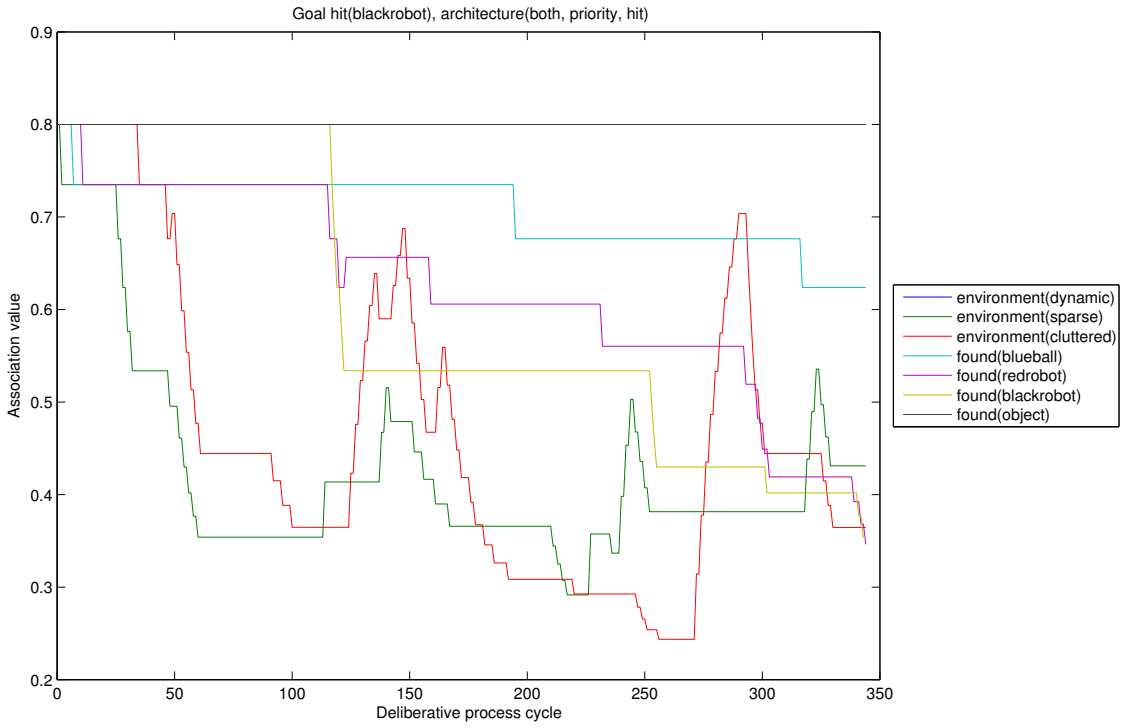
redrobot and *blueball*

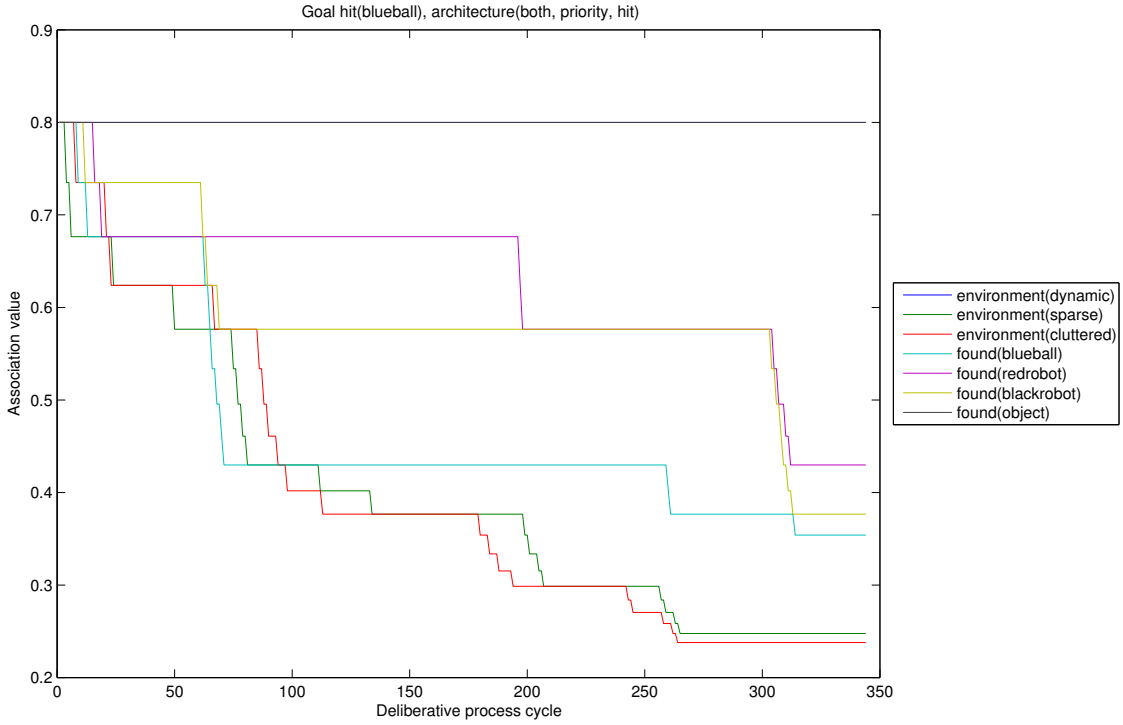
redrobot and *blackrobot*

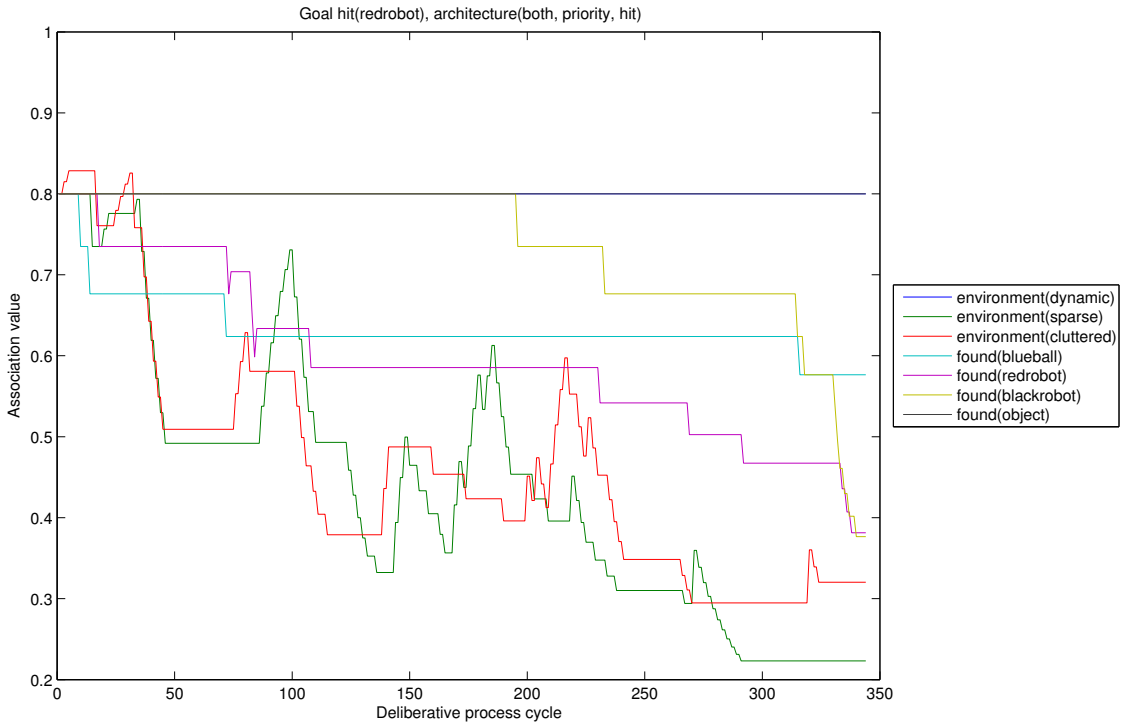
redrobot and *blueball*

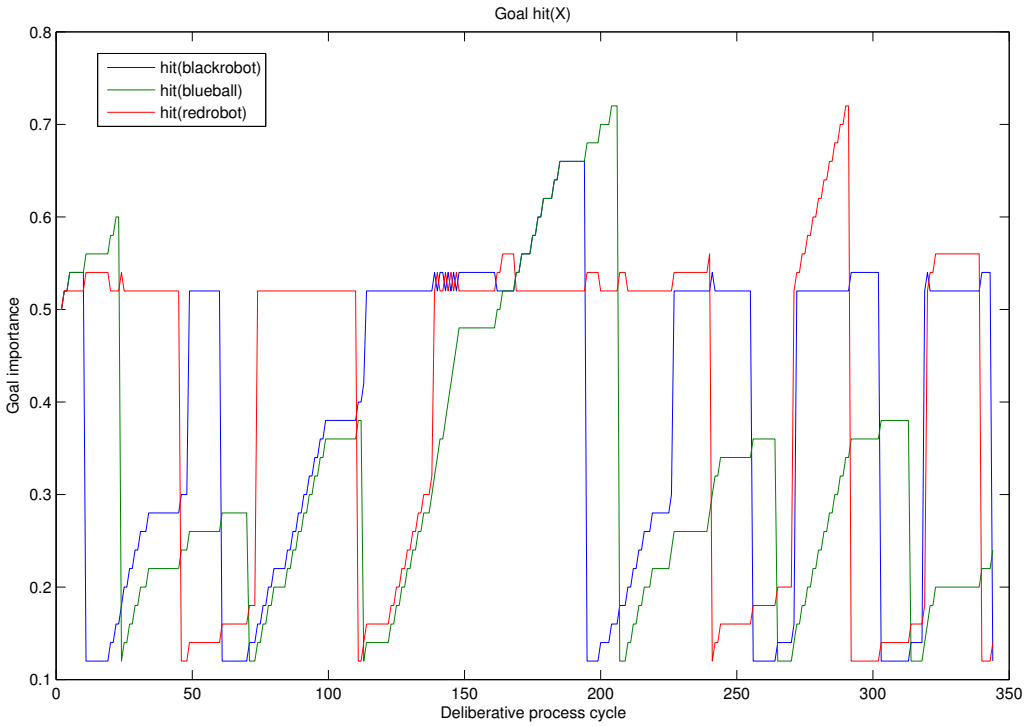



Belief hit(X)
Belief present
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)
fail(architecture)


Goal hit(blackrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(blueball), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(redrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(X)
hit(blackrobot)
hit(blueball)
hit(redrobot)
Goal importance
Deliberative process cycle
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
350

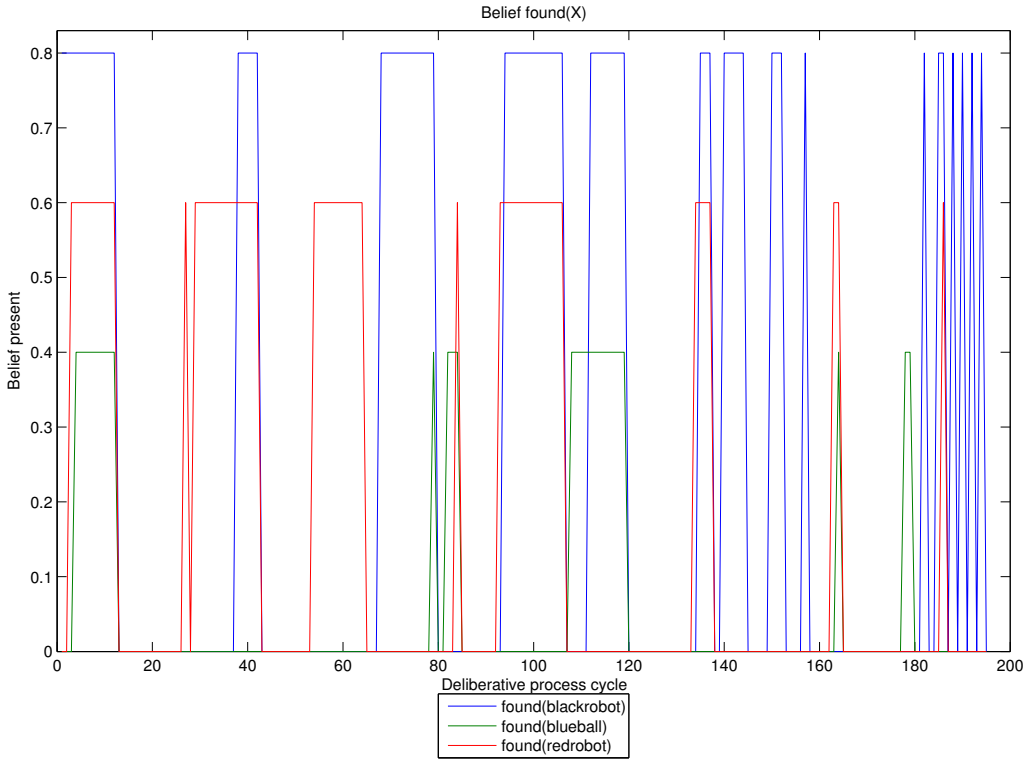
C.4.6 Adaptation Experiment 6

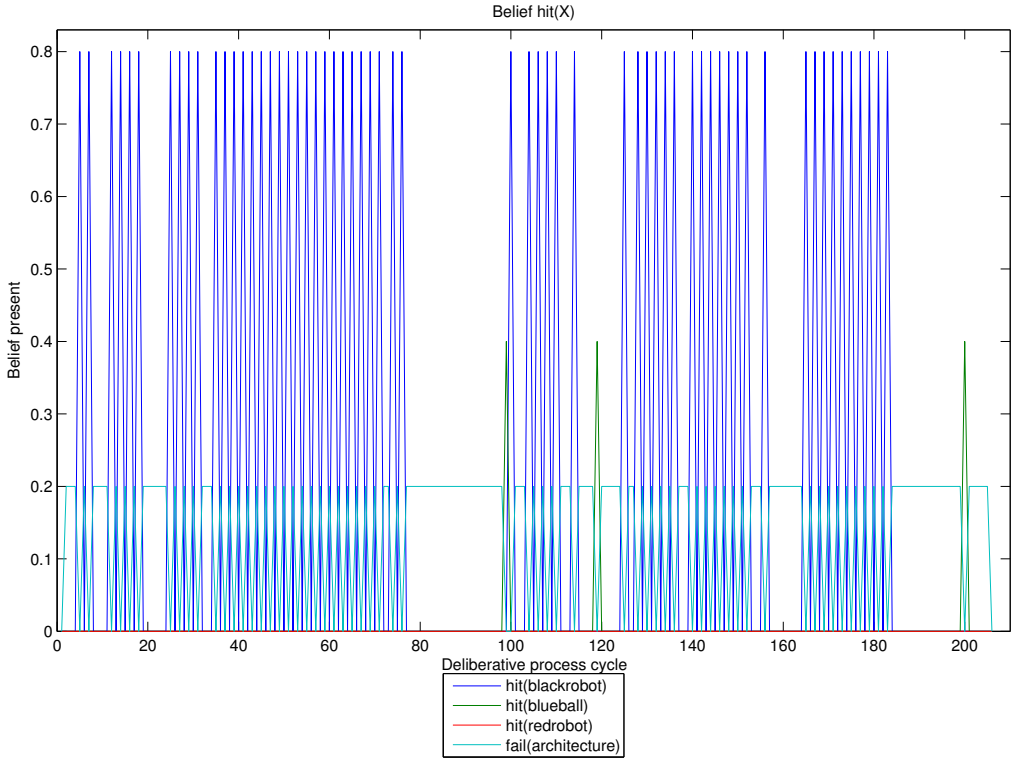
Environment sequence

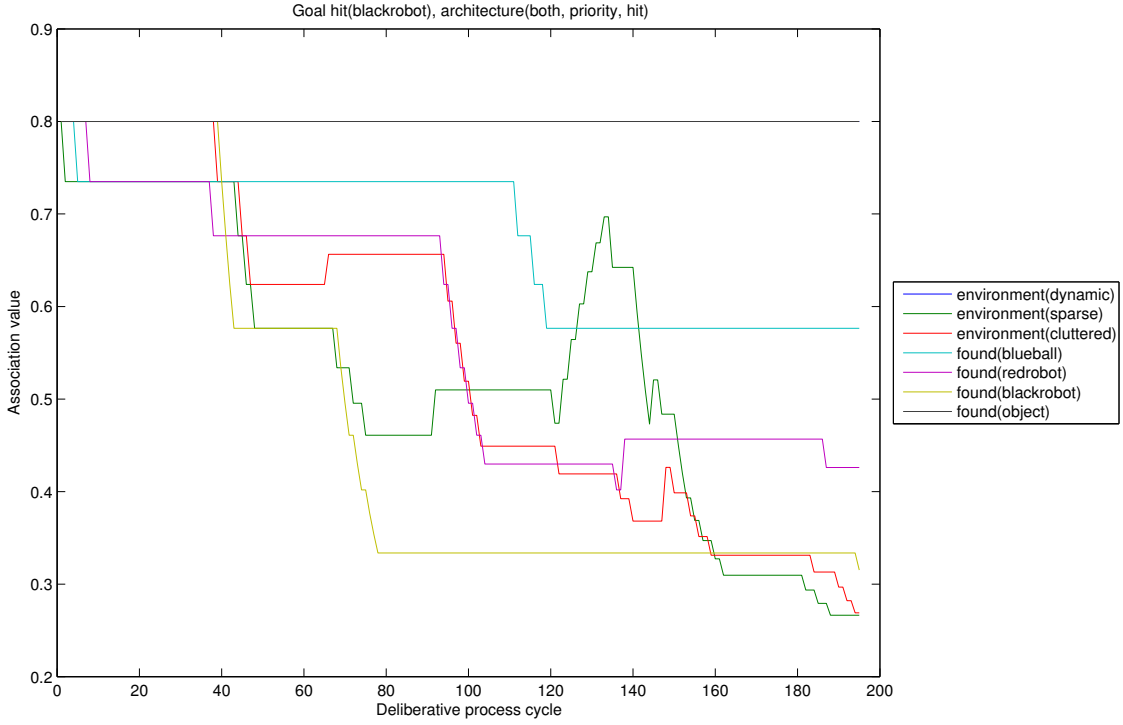
redrobot and *blackrobot*

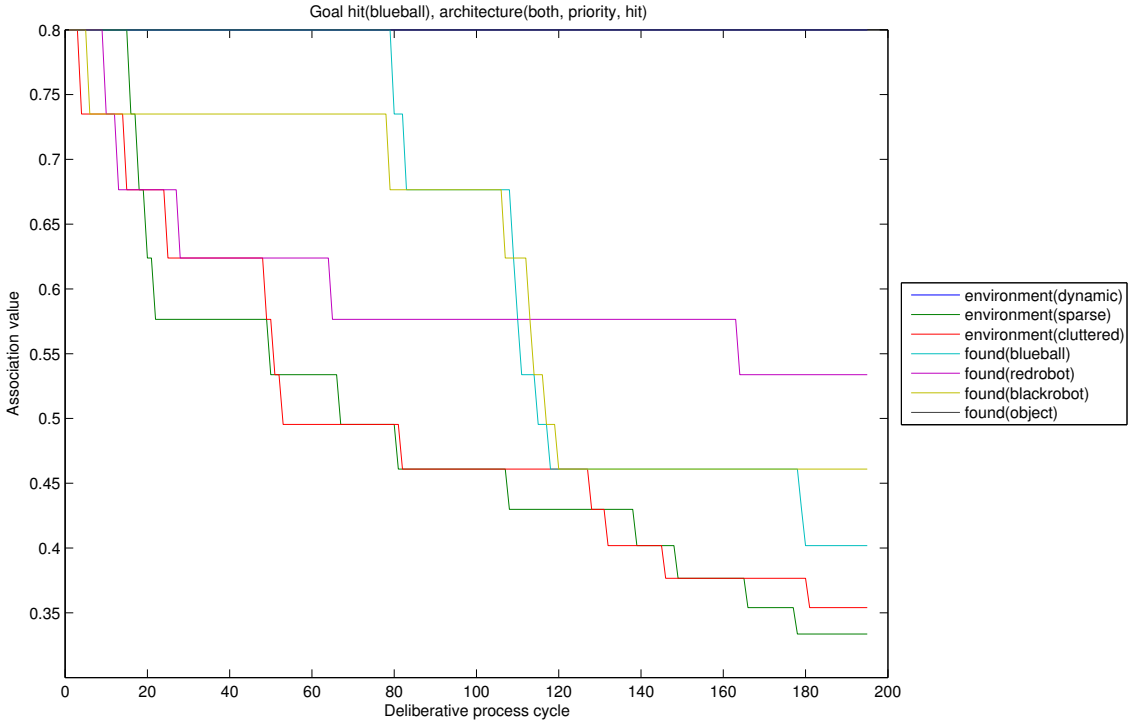
redrobot and *blackrobot*

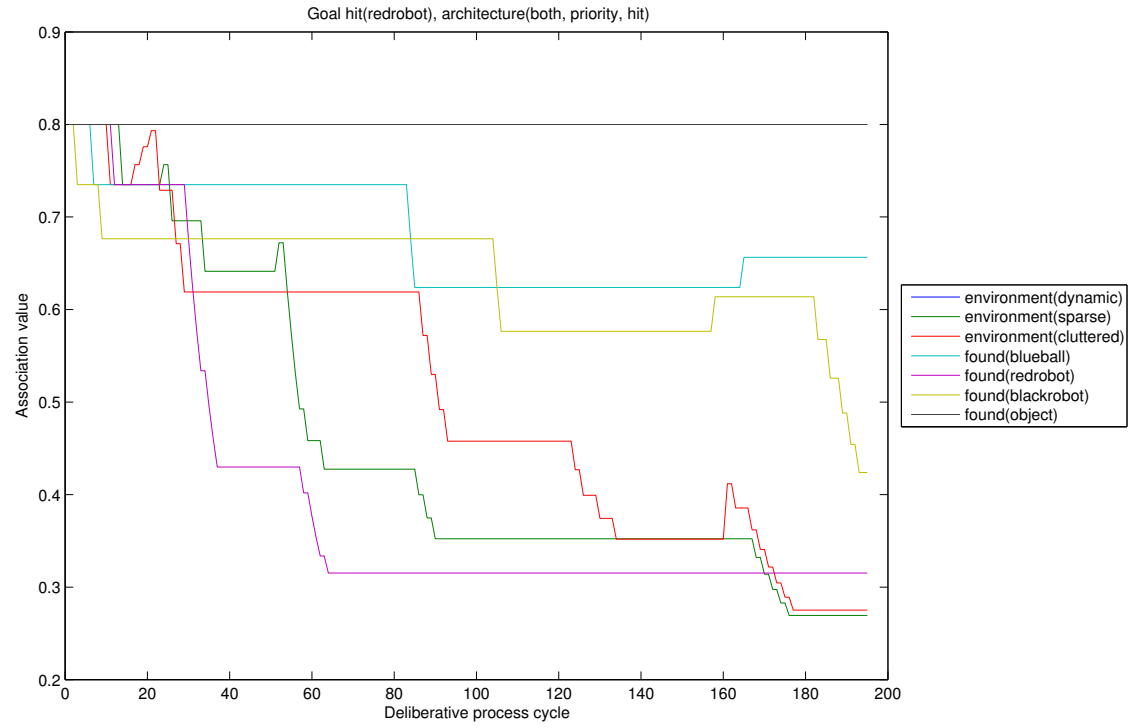
blackrobot

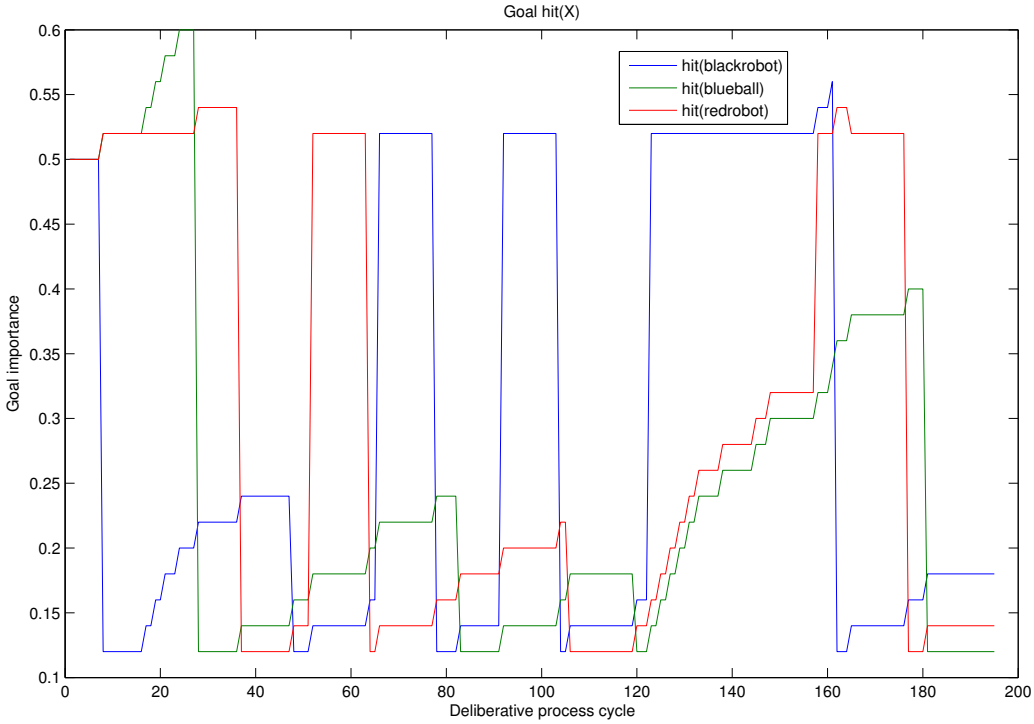



Belief hit(X)
Belief present
Deliberative process cycle
hit(blackrobot)
hit(blueball)
hit(redrobot)
fail(architecture)


Goal hit(blackrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(blueball), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(redrobot), architecture(both, priority, hit)
Association value
Deliberative process cycle
environment(dynamic)
environment(sparse)
environment(cluttered)
found(blueball)
found(redrobot)
found(blackrobot)
found(object)


Goal hit(X)
hit(blackrobot)
hit(blueball)
hit(redrobot)
Goal importance
Deliberative process cycle
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0
20
40
60
80
100
120
140
160
180
200