COUNTERING MURPHY'S LAW: THE USE OF ANTICIPATION AND IMPROVISATION VIA AN EPISODIC MEMORY IN SUPPORT OF INTELLIGENT ROBOT BEHAVIOR

A Dissertation
Presented to
the Academic Faculty

by

Yoichiro Endo

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
College of Computing

Georgia Institute of Technology
December 2008

COUNTERING MURPHY'S LAW: THE USE OF ANTICIPATION AND IMPROVISATION VIA AN EPISODIC MEMORY IN SUPPORT OF INTELLIGENT ROBOT BEHAVIOR

Date Approved: August 5, 2008

for my mother in Tokyo

ACKNOWLEDGEMENTS

First and foremost, I owe an enormous debt of gratitude to my advisor, Dr. Ronald Arkin. Without his insightful guidance, encouragements, thorough critiques, and patience, this dissertation would never have been possible. His extensive knowledge of the robotics field is astounding, and his outstanding professionalism is truly admirable. I would also like to thank the rest of my thesis committee members: Dr. Tucker Balch, Dr. Frank Dellaert, Dr. Steve M. Potter, and Dr. Ashwin Ram. Their constructive comments were particularly helpful to shape my dissertation. I would like to thank my former advisor, Dr. Roger Quinn at Case Western Reserve University, for introducing me to the fascinating field of robotics. The 2001 Neuromorphic Engineering Workshop in Telluride was a very educational and inspirational experience for me. At the workshop, Dr. Shih-Chii Liu from the Institute of Neuroinformatics at Zurich introduced me to the concept of the "place cells" in the hippocampus, becoming the starting point for this dissertation. Countless informative discussions I had with my fellow robotics students at Georgia Tech were also extremely valuable to my dissertation. In this regard, I would like to thank Michael Kaess, Zsolt Kira, Keith O'Hara, Dr. Ananth Ranganathan, Dr. Alexander Stoytchev, Patrick Ulam, and Alan Wagner. Finally, I would like to thank Natasha Dahmen for her emotional support through these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF BOXED EXAMPLES

LIST OF SYMBOLS AND ABBREVIATIONS

b	behavior
b^*	optimal behavior
b^τ	a sequence of executed behaviors up to instance τ
C	a set of all episodes in the robot's memory
$d_{j:q}$	the distance between event e_j and event e_q in terms of event numbers
$\overline{d}$	the average number of events being advanced in one computational cycle
E	effector
E	episode
E_P	episode formed via purposive contextualization
E_U	episode formed via utilitarian contextualization
e	event
e_q	querying event
$\hat{e}_{[E]}$	the matched event in E
g	goal
g_{cur}	current goal
g_{int}	intermediate goal
H	entropy
K	the upper limit for the size of M_{rel}
M	model
M_{rel}	a set of relevant episodes being recalled
$\hat{M}_{\text{rel}}$	a subset of M_{rel} whose events are legitimately matched
m	the number of integrated sensor types

n	the number of events in an episode
O	the asymptotic upper bound of computational complexity
o	observation (perception)
o_{cur}	current observation
o_{end}	nodal effect
o_{init}	nodal precondition
$o_{[e]}$	the observation signal in event e
o_{τ}	observation at instance τ
o^{τ}	a sequence of observations up to instance τ
o'	predicted observation
r	reward signal
r_{cur}	current reward
S	object system
T	timestamp
U	utility
w	weight
z	sensor reading
z'	predicted sensor reading
α	the learning rate in TD learning
β	the number of instantiated motor schemata in b
Δ_E	the amount of delay in the current event progress
ε_m	very small number
η	the normalization factor in the recursive Bayesian filter
θ_H	a threshold for selecting relevant episodes

θ	zenith angle
θ_{Δ}	a threshold for validating the event progress
θ_{ρ}	a threshold for selecting relevant episodes
ι	associative rewarding state
κ_g	a factor in the reward function weighting the similarity between o_{cur} and g_{cur}
κ_m	a factor weighting the output of the transition model when behaviors mismatch
κ_o	a factor in the reward function weighting the similarity between o_{cur} and o'
κ_U	a factor in utility computation weighting the influence of other events
κ_ι	a factor in the reward function weighting similarity between o_{cur} and ι
λ	the eligibility trace in TD learning
Π	a set of all possible motivations
π	motivation
ρ_E	similarity value
σ	motor schema
φ	azimuth angle
χ	episodic context
ψ	motivational magnitude
Ω	referent
Ω^*	primary referent
ω	referent node
ω^*	active referent node
2D	two-dimensional
3D	three-dimensional
AAAI	association for the advancement of artificial intelligence

ABC	anticipatory behavioral control
ACS	anticipatory classifier system
AI	artificial intelligence
AIR	anticipatory-improvisational robot
ANOVA	analysis of variance
AuRA	autonomous robot architecture
CA1	cornu ammonis 1
CA3	cornu ammonis 3
CAIT	computer-animated improvisational theater
CASYS	computing anticipatory systems
CBR	case-based reasoning
CPU	central processing unit
CYRUS	computerized Yale retrieval and updating system
EC	entorhinal cortex
E-MOP	episodic memory organization packet
GHz	gigahertz
GPS	global positioning system
HMM	hidden Markov model
HRI	human-robot interaction
HServer	hardware server
IEEE	institute of electrical and electronics engineers
IT	information technology
JIG	jazz improvisation generator
MDP	Markov decision process
MHz	megahertz

MOP	memory organization packet
MRI	magnetic resonance imaging
NIST	national institute of standards and technology
POMDP	partially observable Markov decision process
PSR	predictive state representation
RMS	root mean square
RS	reactive schema
SALT	strategic arms limitation treaty
SLAM	simultaneous localization and mapping
STRIPS	Stanford research institute problem solver
TD	temporal difference
TEC	theory of event coding
URL	uniform resource locator
USARSim	unified system for automation and robot simulation

SUMMARY

Recently in robotics, substantial efforts have been invested on critical applications such as military, nursing, and search-and-rescue. These applications are critical in a sense that the robots may directly deal with human lives in life-or-death situations, and they are therefore required to make highly intelligent decisions as rapidly as possible. The intelligence we are looking for in this type of situations is proactiveness: the ability to anticipate and improvise.

Anticipation here means that the robot can assess the current situation, predict the future consequence of the situation, and execute an action to have desired outcome based on the determined assessment and prediction. On the other hand, improvisation is performed when the consequence of the situation is not fully known. In other words, it is the ability to deal with a novel situation based on knowledge or skill being acquired before.

In this dissertation, we introduce a biologically inspired computational model of proactive intelligent behavior for robots. Integrating multiple levels of machine learning techniques such as temporal difference learning, instance-based learning, and partially observable Markov decision process, aggregated episodic memories are processed in order to accomplish anticipation as well as improvisation. How this model can be implemented within a software architectural framework and integrated into a physically realized robotic system is also explained. The experimental results using a real robot and high fidelity 3D simulators are then presented in order to help us understand how extended experience of a robot influences its ability to behave proactively.

CHAPTER 1

INTRODUCTION

1.1 Motivation

1.1.1 Anticipation

"If there is any way to do it wrong, he'll find it"; in 1949, an engineer working on the USAF MX981 Project cursed at a technician after discovering his wrong wiring in their mechanical sled, according to United States Air Force historian Ted Bear [22]. The engineer was Capt. Edward A. Murphy. Bear further reports that, by using the sled, the team was investigating the effect of massive force applied on a human body during abrupt braking. Despite the dangerous nature of the project, by constantly reminding themselves about Capt. Murphy's curse, the team was able to complete the project successfully without a major accident. This curse is known today as Murphy's Law: *anything that can go wrong will go wrong* [1].

The moral of this story is not so much about the pessimism, but it is about *anticipation*. Anticipation is *to foresee and deal with in advance* [1]. The success of MX981 was due to the team's ability to anticipate: assessing the current situation, predicting the future consequence of the situation, and executing an action to have a desired outcome based on the assessment and prediction.

Anticipation is also a key to the intelligence of robots. To clarify, the *robot* here refers to a self-contained autonomous mobile robot that is capable of making its own decisions in order to achieve an overall goal that has been assigned to it. It is true that robots today do not yet play a major role in our daily life. However, they are being viewed as the latest technology that could provide convenience to our lives. Like washing machines, automobiles, microwaves, personal computers, and cellular phones, if robots become affordable to consumers, they should be soon working in our living space. As a starter, over two million vacuum cleaning robots, Roomba from iRobot, have been already sold worldwide [47]. As robots start working together with humans, it is unavoidable that some of them would occasionally have to make crucial decisions that affect the safety of people (e.g., in nursing, passenger transportation, search-and-rescue, etc.). As we learned from Capt. Murphy's case, the ability to anticipate is indeed vital in these types of situations.

Since Pavlov [125] conducted a study on classical conditioning (animals' anticipatory physiological response to certain environmental stimuli) more than a hundred years ago, animals' ability to anticipate has fascinated countless researchers. An annual conference called *Computing Anticipatory Systems* or *CASYS* has been discussing the topic of anticipation exclusively for more than a decade. It has attracted researchers from a variety of fields such as biology, psychology, physiology, engineering, artificial intelligence, robotics, economy, and music. The European Commission has recently funded a three-year research project (*Mind RACES: from Reactive to Anticipatory Cognitive Embodied Systems* [56]) to investigate the role of anticipation in cognition; some of the results were presented at *the AAAI Fall Symposium on Anticipatory Cognitive Embodied Systems* in 2005. (Note that our paper [53], which introduced the basic framework of our computational model described in Chapter 3, was also published in this conference.) There is also a workshop called *Anticipatory Behavior in Adaptive Learning Systems* or *ABiALS*, in which the behavioral aspect of anticipation is discussed annually. In other words, there is a substantially large community interested in anticipation today.

Why are these researchers interested in anticipation? From a scientific point of view, not only is it a fascinating task to uncover the underlying principle of how an animal's prescience shapes its behavior, it also seems to have been understood as a key to explain why we behave the way we behave. From an engineering point of view, developing systems that incorporate such a principle seems to be considered useful to advance our everyday lives. More specifically, Rosen [143], one of the most prominent researchers in anticipatory systems, described that "an anticipatory behavior is one in which a change of state in the present occurs as a function of some predicted future state," and "the agency through which the prediction is made must be, in the broadest sense, a model." For example, suppose that a GPS navigation system is currently instructing a driver to continue the current road, Peachtree Street, for another 200 meters and make a left turn at North Avenue; the final destination is the Georgia Tech campus. In this case, *turning to the left* (at North Avenue) can be considered an anticipatory behavior if we view the route specified in a map as a model and the Georgia Tech campus as a predicted future state. In other words, the change of state (from Peachtree Street to North Avenue) occurs in response to the future state (the Georgia Tech campus). Furthermore, if the driver takes a certain preparatory action for the left turn before even seeing North Avenue (e.g., starting the left blinker, changing the lane to the leftmost one, slowing down the speed, etc.), this can be also considered an anticipatory behavior; the predicted future state is, in this case, North Avenue, and the model is a sequence of actions developed from the driver's own experience. Note that our computational model developed in this dissertation (Chapter 3) deals with this experience-based anticipation. Experience-based anticipation also allows the driver to print out a paper copy of the route (from Google Maps, MapQuest, etc.) before starting the trip just in case the GPS signal becomes disrupted (as it perhaps did before).

Instead of reacting to an ongoing event, by starting to respond to an event that has not yet happened, anticipation provides us with more time to prepare for that event. Naturally, such a response should be more advantageous than simply reacting as it could save time to complete the overall task and/or yield a more advantageous outcome.

Furthermore, the advantage of *experience-based* anticipation is that, unlike map-based anticipation, it can handle non-navigational behaviors (e.g., turning on a blinker, printing out a route, etc.). Hence, if it is appropriately applied, a robot with experience-based anticipation can effectively complete tasks that are typically beyond the scope of the conventional navigational domain.

For example, suppose that an experience-based anticipatory robot is working in a restaurant as a server, and a regular customer just came in. Based on the previous interactions with the customer, the robot can prepare the customer's favorite beverage even before he/she is seated. Thus, the customer would not have to wait for the drink to arrive, and the robot can save an extra trip to take the drink order. Similarly, based on the experience in the past, an elderly assistance robot at home can prepare the elder's favorite breakfast before he/she wakes up, bring a daily newspaper to the table before he/she arrives, turn on his/her favorite TV show before he/she starts walking towards the living room, and so on. Furthermore, to find survivors in a collapsed building, a search-and-rescue robot can start inspecting the part of the rubble where survivors are most likely buried based on the similar experience in the past.

It should be also noted that experience-based anticipation is applicable to non-robotic intelligent systems as well. For example, a company printer may automatically send an email to the IT department before running out the paper, so that it can be restocked before inconveniencing the users. Similarly, an inventory system in a retail store may automatically order merchandises from factories before they become out of stock. A similar concept can be also applied to a much larger system. For example, if a severe heat wave is forecasted in a city, a local power company may automatically request regional or even distant power plants to increase their capacities before an electrical power outage occurs in the city (i.e., a "smart grid" [106]).

1.1.2 Improvisation

Another important issue in terms of robots dealing with the real world is the question of what they should do if their anticipation fails. On April 13, 1970, one of two oxygen tanks exploded when the Apollo 13 spacecraft was on its way to the moon. The mission to land on the moon was aborted, and all of the three crewmembers had to evacuate into the cockpit of the lunar module at that time. The lunar module was primarily designed to land on the lunar surface, but what the NASA engineers did not anticipate was to use the lunar module as a lifeboat when only half of the oxygen in the spacecraft was available. The crew was facing CO_2 poisoning in the cockpit. Nevertheless, the NASA engineers were able to save the lives of the crew by coming up with an improvised solution; an ad-hoc air purifier was constructed using materials available in the spacecraft (oversized lithium hydroxide canisters, duct tape, cardboard paper taken out from a manual, and plastic sheet removed from thermal undergarments) [103]. *Improvisation*, which is *to make, invent, or arrange offhand* [1], is also crucial to the intelligence of robots perhaps especially when their anticipation fails.

More specifically, anticipation is useful for a robot when it is performing some routine task; a model developed though the experience in the past can be thus straightforwardly utilized to suggest appropriate current actions. On the other hand, improvisation becomes useful when the robot is in a situation that is not fully covered by the model (i.e., outside the normal routines). Even with this incomplete knowledge, however, improvisation allows the robot to come up with a solution, so that it can still complete the original task without substantial delay. Note that improvisation is different from planning/re-planning. As noted by Agre [4], planning is performed when the information regarding the world is already known; on the other hand, improvisation is performed when the characteristics of the world are not necessarily fully understood. For example, recall the GPS-based navigation scenario in the previous section. Suppose that Peachtree Street is totally blocked. Based on the latest traffic update, the GPS device may automatically reroute the path to detour around Peachtree Street. This *rerouting* is, however, not an example of improvisation. It is an example of re-planning, or it can be considered a form of anticipation since the model still had a complete knowledge (map) of the world to suggest how to reach the predicted future state (the Georgia Tech campus). On the other hand, suppose that Peachtree Street is again totally blocked, the GPS signal is now disrupted, and the paper copy of the route only shows the original path (via Peachtree Street). Suppose also that the driver knew the pencil-shaped tallest building in the midtown is the Bank of America Plaza located on North Avenue. Without taking Peachtree Street, if the driver could reach North Avenue by constantly steering the car towards this tallest building and, from there, arrive at the Georgia Tech campus by referring to the paper copy of the route, the driver is considered to have performed improvisation. Note that our computational model developed in this dissertation (Chapter 3) also deals with improvisation. Furthermore, similar to experience-based anticipation, improvisational behavior is determined based on the knowledge of the world developed from the robot's own experience (i.e., experience-based improvisation).

Improvisation is thus important for a robot because it enables the robot to solve a time-sensitive problem without complete knowledge of the world. Experience-based improvisation is important for a robot because, like experience-based anticipation, the tasks it can perform are not limited to just navigation. For example, in case of the server robot in a restaurant, suppose that the robot is serving a table with a group of four. Suppose also that a friend of the group just walked into the restaurant and decided to join the group even though the table can seat only four. Based on the experience of rearranging tables in the past, the robot with experience-based improvisation should be able to bring a nearby empty chair to accommodate this fifth person. In the case of elderly assistance, if the elder expresses an excruciating pain in his/her stomach while watching TV, the experience-based improvisational robot should be able to call a doctor (or alarm neighbors) just as it did when, for example, the same or another elder had a severe chest pain previously. Similarly, if the search-and-rescue robot found itself stuck between obstacles, it should be able to automatically call for assistance from a human rescue worker or another rescue robot (or even disassemble its own body part that is responsible for its being stuck).

Furthermore, as in experience-based anticipation, experience-based improvisation is also applicable to non-robotic intelligent systems. In case of the office printer, suppose that the printer found itself being out of commission due to jammed paper. Experience-based improvisation allows the printer to forward the unfinished print queue to a nearby printer automatically, so that it can be printed regardless without delay. Similarly, suppose that a web server is experiencing severe overload after, for instance, its managed hostname appeared on several popular online newspapers or diaries (blogs). Via experience-based improvisation, the server should automatically be able to transfer the web contents to some high-end server, request temporary web hosting, and pay a fee to the host if applicable. In case of an inventory system with an automatic restocking (anticipatory) capability, if a certain brand of merchandise was found to be discontinued, based on past experience, the system should place an order for a similar product for a different brand automatically.

1.1.3 From Experience to Proactive Behavior

As mentioned above, anticipation is for acting advantageously in familiar (routine) situations whereas improvisation is for dealing with novel situations. While it is certainly useful to have a robot that is capable of exploiting both means, the question is how to realize such a robot. In other words, how can we make a robot behave proactively, that is, to act in an anticipatory and/or improvisational manner? In this research, we seek clues from how our own brains work. Like a human infant, a brand-new robot, unwrapped from a shipping box, may not be ready yet to perform anticipation or improvisation. However, after having interactions with the real world for a certain period of time, we conjecture that the robot should eventually be able to figure out how to anticipate and/or improvise by reasoning about the current situation based on relevant episodes[1] that it has experienced in the past. Naturally, in order for the robot to recall relevant episodes, they have to be stored in some form of memory. In particular, we are interested in an *episodic memory*, a form of memory that contains information associated with a particular episode of experience, and it is stored in a way that the episode can be traced back and recalled later in time [184]. Given a sufficient framework to process a current episode of experience, store it in an episodic memory, and recall and utilize relevant past episodes for an ongoing situation, our primary hypothesis is that long exposure to the real world and interactions with it should help a robot improve its ability to anticipate. In other words, it provides better assessments of the current situation, formulates better predictions of the future consequences of the situation, and executes better actions based on the assessment and prediction. Furthermore, even if anticipation fails, the accumulated experiences should also help the robot perform a better improvisation. Of course, at first, we would have to identify what common denominator the processes of

[1] Here, *episode* is loosely defined as an *event that is distinctive and separate although part of a larger series* [1]. This will be further elaborated in Chapters 2 and 3.

anticipation and improvisation share in terms of recollection and exploitation of past experience. Nevertheless, our main objective in this research is to study how the lifelong experience of a robot influences its ability to anticipate as well as the way it improvises its actions. We pay special attention to episodic memory, since we view that it is an essential mediator between experience and such intelligent behavior.

In order for a robot to encode a current episode of experience in an episodic memory, we would first need to determine what information should be extracted and be remembered in order for it to be utilized for anticipation and improvisation in the future. One of the factors that have to be considered is its storage space. In jazz music, for example, there seems to be "a lifetime of preparation and knowledge behind every idea that an improviser performs" [26]. When robots become part of our daily life in the future, they could be activated for an extended period (perhaps for years). Accordingly, the volume of information a robot goes through in its lifetime would be massive. On the other hand, the amount of information the robot can store is always limited because of hardware constraints (e.g., capacity of a memory chip, physical space available for installing memory chips, etc.). Thus, the information to be extracted from a current episode of experience should be very small while it still retains knowledge that is essential for future anticipation/improvisation.

Moreover, in addition to *what* to store, we also need to consider *how* to store, recall, and utilize episodes since it would likely affect the search time upon anticipation/improvisation. For example, in order to recall relevant episodes promptly for a current situation, how should past episodes be organized in the memory of a robot? Should similar episodes be grouped together? If so, what does it mean for episodes to be *similar*? Would it mean to have a similar goal or a similar outcome? Integration of emotion and/or motivation could also be exploited in this context. In our brains, for example, the information being extracted from an episode of experience appears to be labeled with an emotionally induced marker (referred to as a *somatic marker*) upon storage, and such markers seem to help us make advantageous decisions [45]. Of course, the obvious question is whether this model can or should be applied to a *robot*. Thus, whether a somatic marker could help a robot achieve better anticipation and/or improvisation is also examined in this research.

Finally, whether using a somatic marker or not, one particular scheme of recollection and exploitation of episodic memories would likely produce faster anticipation and/or improvisation than other schemes. In this case, however, it is possible to compromise the quality of the anticipation/improvisation by rushing into a quick solution. Thus, the possible trade-off between promptness and quality of anticipation/improvisation is also explored in this research.

1.2 Research Questions

The research problems discussed in the previous section are here restated as the primary research question and the subsidiary questions of this dissertation. Our research is carried out in a way that it can help us determine these questions.

1.2.1 Primary Research Question

How does lifelong experience of a robot influence its ability to anticipate as well as the way it improvises its actions?

In other words, this research seeks to determine how extended experience of a robot affects its ability to behave proactively. In this dissertation, this primary research question is investigated by exploring a series of subsidiary questions presented below.

1.2.2 Subsidiary Questions

To help address the issues posed by the primary research question, the following five subsidiary questions are forwarded. This research is carried out in a way that it leads to answers to these questions.

<u>Subsidiary Question 1</u>

What common denominator do the processes of anticipation and improvisation for a robot share in terms of recollection and exploitation of past experience?

An assumption here is that the processes of computing anticipation and improvisation do share certain commonalities. In this research, we seek to identify specifically what these are. In particular, this issue is addressed in our computational model of proactive intelligent behavior for robots (Chapter 3), in which the algorithmic processes involved in computation of both anticipation and improvisation are explained.

<u>Subsidiary Question 2</u>

What information should a robot extract from a current episode of experience to be remembered in order for it to be utilized upon anticipation and improvisation in the future?

To compute proactive behavior for a robot, we are interested in utilizing episodic memory that stores a particular episode of experience, which can be later retrieved and recounted in accordance with how it unfolded [184, 185]. This question is posed to identify exactly what should be stored in an episodic memory. This question is first explored in the literature review in Chapter 2 (Section 2.1.1) in which related work on episodic memory is examined. It is further addressed in our computational model (Chapter 3) in which the role of episodic memory in computing proactive behavior is discussed.

Subsidiary Question 3

How should past episodes of experience be organized in the memory of a robot in order for them to be utilized upon anticipation and improvisation?

While the previous question addresses *what* information should be stored in episodic memory, this question addresses *how* it should be stored. We are interested in the organizational structure of the memory. In the literature review, after examining related work on episodic memory (Section 2.1.1), existing memory-based problem-solving techniques (case-based reasoning and instance-based learning) and their memory structures are discussed (Sections 2.4.2 and 2.4.3). This question is further addressed in our computational model (Chapter 3) into which episodic memory is incorporated as its foundational data structure. The efficiency of this data structure is examined in one of the experiments in Chapter 5 (Section 5.1).

Subsidiary Question 4

Does a memory with integrated somatic markers help a robot achieve better anticipation and/or improvisation than without them?

As mentioned above, the human brain, there seems to be a certain neural mechanism to integrate emotionally induced signals into episodic memories [45]. We hypothesize that the notion of somatic markers indeed helps compute better proactive behavior. In this research, the concept of somatic markers is first examined in the related literature review in Chapter 2 (Section 2.1.2). The role of somatic markers in terms of proactive behavior computation is then discussed in our computational model (Chapter 3), in which a model of somatic markers is integrated into the foundational data structure. Finally, the effectiveness of somatic markers in terms of proactive behavior computation is examined in Chapter 5 (Section 5.2).

Subsidiary Question 5

What is the trade-off between promptness and the quality of anticipation/improvisation that a robot performs?

While the previous questions attempt to uncover the constitution of the mechanism that allows transformation of extended experience of a robot into its proactive behavior, this question seeks to determine the characteristics of the behavior produced by the mechanism. In particular, the relationship between promptness of the proactive behavior computation and the quality of the behavior is examined in Chapter 5 (Section 5.3).

1.3 Contributions

As noted above, by exploring the primary research question, this research helps us understand the nature of the relationship between the extended experience of a robot and its ability to anticipate as well as improvise. In particular, the computational model of proactive intelligent behavior for robots presented in Chapter 3 unveils the commonalities between the processes of anticipation and improvisation (Subsidiary Question 1), identifies *what* and *how* information should be stored in episodic memories (Subsidiary Questions 2 and 3), and clarifies the role of somatic markers (Subsidiary Question 4). Furthermore, the experiments in Chapter 5 elucidate the characteristics of proactive behavior with respect to the organizational structure of the episodic memory (Subsidiary Question 3), somatic markers (Subsidiary Question 4), and the promptness of computing (Subsidiary Question 5).

In addition, other specific contributions include:

- A computational model of proactive intelligent behavior for robots (Chapter 3).
- An experimental result verifying the computational model (Chapter 5).
- An efficient world representation that reduces the POMDP (partially observable Markov

decision process) computation load (Section 3.1).

- A robotic system capable of performing proactive navigational tasks without pose sensors (Chapter 5).
- The first robotic implementation of the new hippocampal hypothesis by Eichenbaum et al. [51] (Section 3.1).
- A novel evaluation method for robotic somatic markers (Section 5.2).
- A novel robotic system that performs practical (non-artistic) improvisation (Section 5.3).
- A novel way to hybridize CBR (case-based reasoning) and POMDP (Section 3.2).

These are all further elaborated in Chapter 6 (Section 6.2).

1.4 Dissertation Overview

This dissertation is comprised of six chapters. Background and related research are first reviewed in Chapter 2. In particular, relevant work in the biological sciences, studies on anticipation and improvisation, and relevant machine learning techniques are reviewed. The main thesis of this dissertation, a computational model of proactive intelligent behavior for robots, is introduced in Chapter 3. The implementation and evaluation of the computational model are then described in Chapters 4 and 5, respectively. Finally, conclusions and future work are discussed in Chapter 6.

CHAPTER 2

BACKGROUND AND RELATED WORK

This dissertation appertains to a variety of academic disciplines. Those researches that are relevant to the main computational model, described in Chapter 3, are reviewed in this chapter. At first, inspirational research in the field of biology that serves as the foundation of this computational model is discussed. The second and third sections examine related work on anticipatory and improvisational robots, respectively. Relevant machine learning techniques that are employed or mentioned in Chapter 3 are also reviewed at the final section of this chapter.

2.1 Biological Perspectives

The main computational model of this dissertation was inspired by how a mammalian brain works. The biological findings that became foundation of this model are reviewed in this section. In particular, the studies related to episodic memories, hippocampal functions, and the somatic marker hypothesis are discussed.

2.1.1 Episodic Memory and Hippocampal Functions

As discussed in Chapter 1, the primary goal of this dissertation is to understand how extended experience of a robot affects its ability to behave proactively. Hence, we pay special attention to *episodic memory* as it is expected to provide a robot with a means to store its experience. More specifically, the term "episodic memory" was first coined by Tulving [184] to describe a type of memory storing one's firsthand experience. This memory may retain spatial or non-spatial cues [141]. For example, a distance to a coffee mug can be a spatial cue while an aroma of coffee can be a non-spatial one. Once stored in memory, the information can be subsequently retrieved, and the experience can be recounted in accordance with how it actually proceeded [185]. This property can be then utilized to assess the current situation and anticipate what may happen next [197].

Attributing to Tulving's work [184], episodic memory is often contrasted with *semantic memory*. While both are considered declarative memory [50], which is the type of memory that allows recollection of "everyday facts and events" [48], semantic memory refers to the type of memory retaining general knowledge of the world, stored independently from the animal's experience (e.g., "Tokyo is the capital of Japan", etc.).

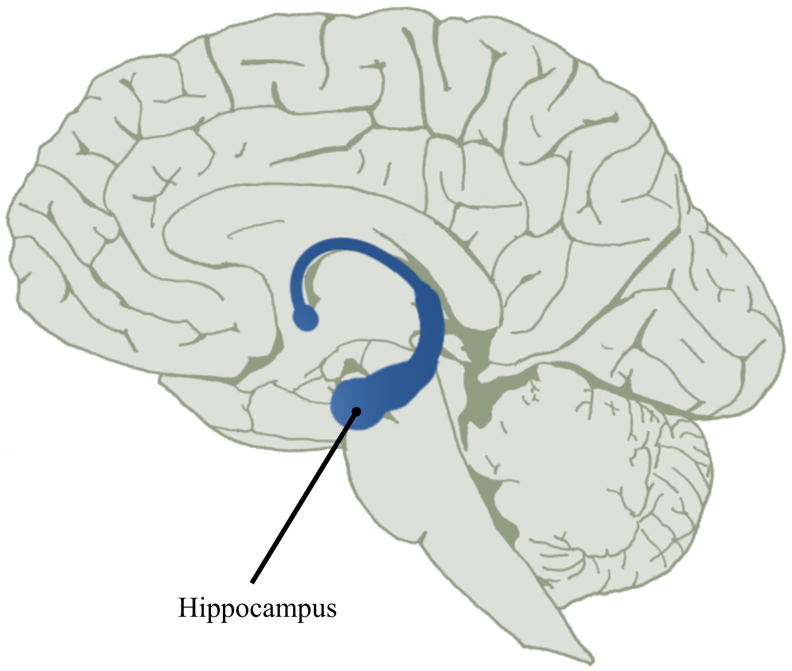


Figure 1: The hippocampus in a human brain.

It is commonly acknowledged that formation of episodic memories is primarily carried out by the hippocampus [161, 165, 200], an element that exists in the interior of a mammalian brain (Figure 1). Each mammalian brain holds a pair of hippocampi (left and right) in its medial temporal lobe. The shape of the human hippocampus resembles a seahorse, after which this part of the brain was named. Hippocampi have been studied by numerous researchers due to their distinctive functions associated with memory and spatial navigation. In addition, it is also speculated that this part of the brain is responsible for detecting novelty in arriving sensory information.

Memory Function

Our main computational model presented in Chapter 3 deals with formation of episodic memories inspired by how the mammalian hippocampi accomplish this process. One of the most notable studies on the hippocampal memory function was conducted by Scoville and Milner [156] on patient H. M., whose hippocampal formation and its surroundings were mostly removed by a surgical procedure in order to treat his chronic neurological disorder. The removal of the region resulted in H. M.'s inability to convert short-term memory into long-term memory. In addition, H. M. lost two years' worth of the memory prior to his surgery. In other words, even though his old memories acquired before two-year prior to the surgery were intact, and he was able to form a new memory for a brief moment (due to functional short-term memory), H. M. was no longer able to remember any of interactions with the world since the removal of the hippocampal region. Hence, attributable to Scoville and Milner's work on H. M. and subsequent studies including scans of monkeys' brains using MRI [166], the hippocampus is widely regarded as a region that converts short-term memory into long-term memory as well as aiding the recall of episodic memories.

Cognitive Map

Another recognized functionality of the hippocampus is regarding spatial navigation. For example, London taxi drivers, who routinely endure extensive navigational tasks, were found to possess significantly larger posterior hippocampi than average people [105]. A seminal study on the hippocampal role on navigation was conducted by O'Keefe and Nadel [123], who discovered that certain cells in a rat's hippocampus excite whenever the animal is in a familiar environment. The discovery of these cells, known as *place cells*, has led O'Keefe and Nadel to speculate that a cognitive map is constructed in a hippocampus. *Cognitive map* is a term first coined by Tolman [180] who claimed that animals are capable of remembering spatial information of the environment regardless of the presence of a reinforcer (e.g., food). Tolman's view was controversial at that time as it challenged the en vogue school of psychology (behaviorists) attempting to understand animal behaviors only through the connections between stimuli and responses, rejecting any internal representation such as a "map" [27]. While Tolman did not elaborate how a cognitive map is constructed in an animal's brain (except to say that some "nervous system" in the brain had to be responsible for it [180]), O'Keefe and Nadel [123] conjectured that a cognitive map is expressed in a two-dimensional Euclidean space. To examine their supposition, a computational model has been developed by Burgess et al. (including O'Keefe himself) [34]. In this model, one of the most prominent components is a path-integrator. In other words, a dead-reckoning mechanism is employed in order to convert the egocentric view that an animal perceives into the unified two-dimensional (geocentric) framework with which the cognitive map supposedly works. Burgess et al. [36] further hypothesized that, between the two hippocampi in a human brain, the right hippocampus in particular could be dealing with the processing of geocentric spatial information. Many other researchers have also proposed hippocampal computational models that utilize path-integration (e.g., [66, 111, 138, 139, 144, 160]), and some have even tested it on real mobile robots successfully [35, 114]. A few of the models [60, 140] were shown to solve the Morris water-maze problem. Here, the Morris water-maze problem [117] is a type of experiment in which an animal (typically a rat) released in an opaque water pool attempts to arrive at a target platform hidden under the water. Although the target is invisible from the animal's view above the surface, if suitably trained, the animal can find the goal faster than untrained ones. The hippocampus is indeed considered responsible for dealing with such a task [118]. According to the experiment with real rats [61, 73], the place cells of those animals do fire actively when they reach the underwater target.

Note that Endo and Arkin [54] has also developed a computational model of a hippocampally-inspired cognitive map, in which the SLAM (simultaneous localization and mapping) aspect of the computation was investigated. This computational model is a precursor to the main computational model presented in Chapter 3. Note that our computational models do not convert the egocentric view that a robot perceives into a two-dimensional geocentric map because they are founded on an alternative hypothesis that rejects the notion of a two-dimensional geocentric cognitive map. The details of this alternative hypothesis, viewing the hippocampus as where the spatial information is represented in term of discrete episodic memories, are explained in the next subsection.

Memory Space

As discussed above, many have hypothesized that the hippocampus is able to construct two-dimensional geocentric cognitive maps. However, this premise has been challenged by several scientists who further studied the hippocampi of rats [121]. For example, while O'Keefe [122] attested that the sole function of a rat's hippocampus has everything to do with "processing and storage of spatial information," the experiment by Bunsey and Eichenbaum [33] showed that the hippocampus also responds to non-spatial cues such as odor. Eichenbaum and his colleagues [49, 51, 158] further argued that the hippocampus does not project spatial information in an unified world framework. Instead, they proposed that it constructs a "memory space" where the spatial information is represented in term of discrete episodic memories, which are further interconnected to each other. Our main computational model presented in Chapter 3 is founded upon this notion of data structure where an *episodic memory* is considered "a sequence of event representations, with each event characterized by a particular combination of spatial and nonspatial stimuli and behavioral actions" [51]. Incidentally, this concept of event is in accordance with the theory of event coding (TEC) proposed by cognitive psychologists Hommel et al. [74]. According to TEC, when perceptual information is stored in the memory, corresponding behavioral information is also integrated into the framework, forming a joint representation. Such a representational structure, according to the theory, facilitates the animal to behave in an anticipatory fashion. Indeed, in our computational model of proactive intelligent behavior for robots (Chapter 3), this representation of event that encapsulates both perception and behavior is considered one of the constituents of an episode, which is utilized to compute anticipatory (as well as improvisational) behavior.

To examine the notion of hippocampus constructing a "memory space", Wood et al. [198] conducted a T-maze experiment using rats in which the hippocampal activities of the rats were monitored through attached electrodes. There, the firing patterns were found to correlate more with the context of episodes (e.g., whether to turn left or right at the junction) than with the geographical location itself, contradicting the assertion of the other hippocampal school of thought that a "self" position is localized relative to a 2D geographical map. Furthermore, based on an experiment using rats with hippocampal lesions, Fortin et al. [58] concluded that the hippocampus indeed plays a crucial role in forming an episodic memory comprised of a sequence of events.

Novelty Detection

Novelty detection is also a prominent function that the hippocampus seems to possess [50], and it is also one of the functionalities implemented in our computational model of proactive intelligent behavior for robots (Chapter 3). For example, monkeys with hippocampal lesions were found not to be able to distinguish between a familiar object and a novel object when presented sequentially with a fixed delay [165]. Similar conclusions have been also drawn from experiments using rats [75] and humans [131, 187].

Exactly which component of hippocampus is responsible for detecting novelty in the sensory information is still debated among scientists even though it is generally agreed that areas CA1 and CA3 (Figure 2) bear such a responsibility [68]. For example, some researchers [69, 119] hypothesized that CA1 acts in the role of a comparator. More specifically, based on a recalled memory arrived from CA3, in CA1, the current sensory signals are predicted and compared against the actual signals arrived from the entorhinal cortex (Figure 3). On the other hand, other researchers [97, 192] suggested that such a comparator is in fact implemented in CA3. It should be noted that, while the empirical data collected from rat experiments [59] suggest that recollection of past experience is administered largely by the hippocampus, for humans, other regions in the brain such as the frontal and parietal lobes are also believed to be involved in the memory recollection process [37, 186]. When human infants deal with novel situations, it has been speculated that amygdalae play a significant role as well [155].

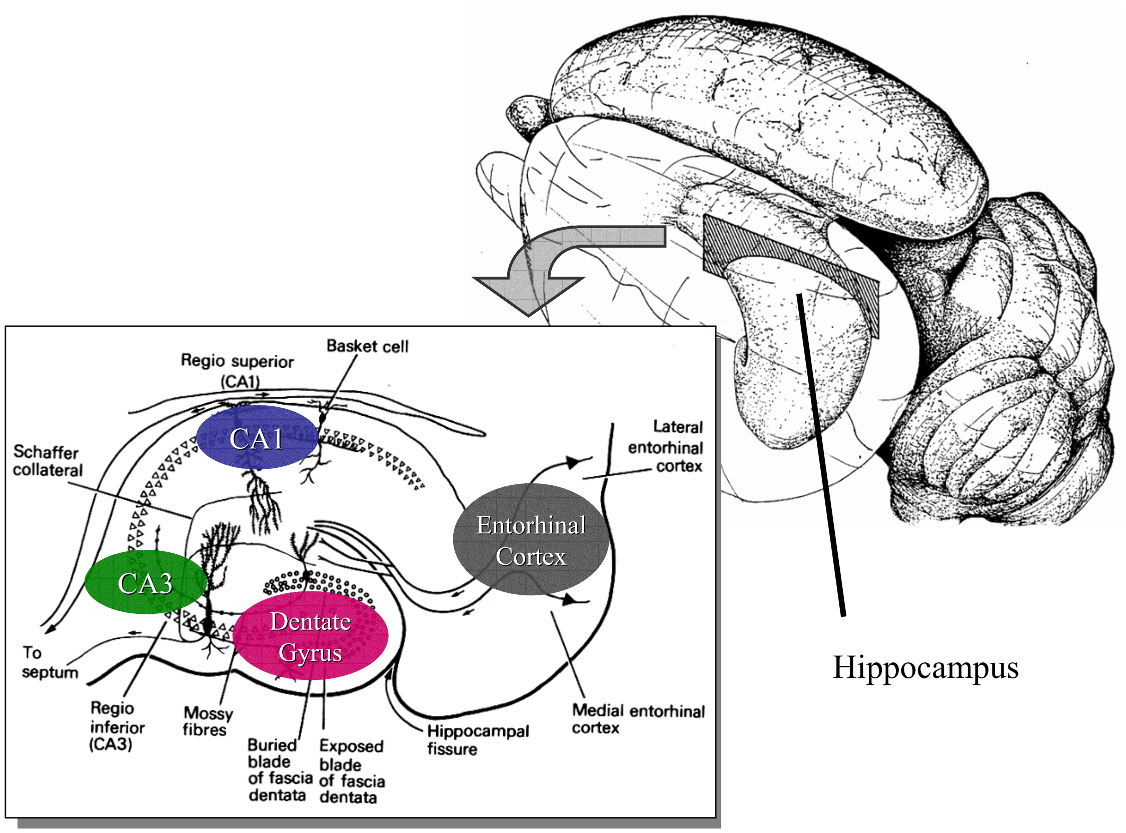


Figure 2: A rat brain (upper right) and the cross-section of the hippocampus with its subdivisions (lower left). (Diagram reproduced from [188] and [123].)

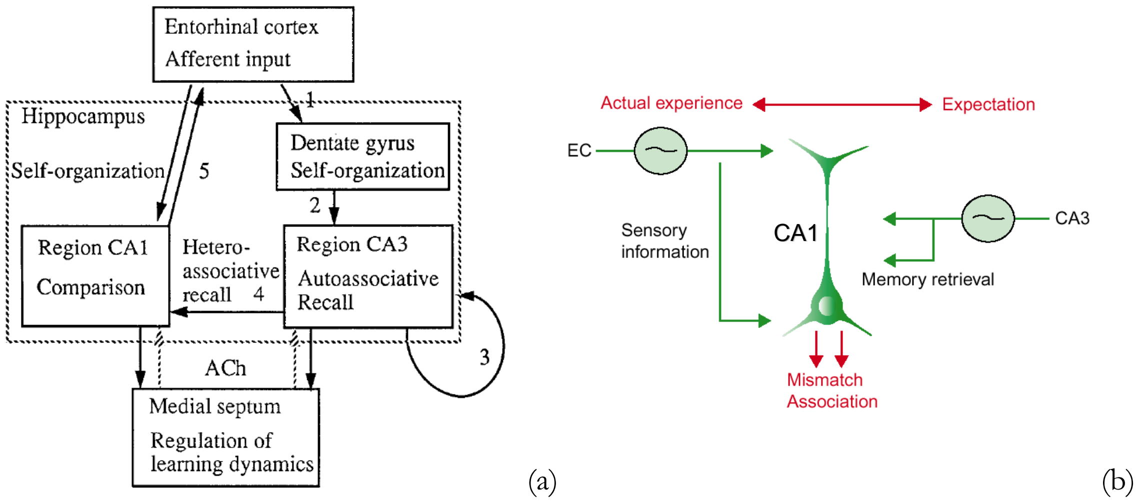


Figure 3: Hippocampal area CA1 as a comparator: (a) Hasselmo's model, (b) a simplified diagram of the same concept. A mismatch between expected signals from hippocampus area CA3 and the present sensory information from the entorhinal cortex (EC) is detected at CA1. (Diagram reproduced from [69] and [119].)

Regardless of whether such a comparator exists in CA1 or CA3, the concept of comparing the actual sensory signals and expected signals produced from recalled memory has been studied for several decades. For example, Held [71] proposed that, when an animal attempts to make an action, the actual sensory signals induced by the self-movement (referred to as *re-afferent signals*, first coined by von Holst [194]) are compared against the predicted ones, which are internally produced based on previous experiences (Figure 4). In other words, consequences of the animal's self-movements are constantly predicted and compared against the actual sensory signals, and the result of the comparison is regarded as "perception". Tolman [181] referred to this concept as "means-end-expectation" or "sign-gestalt-expectation".

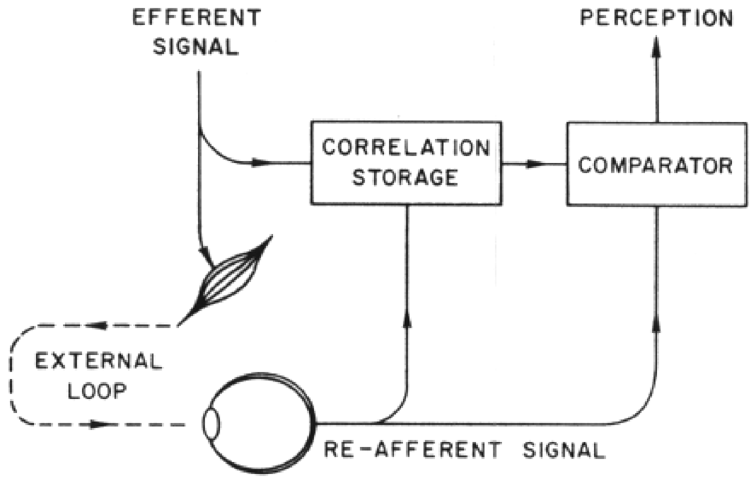


Figure 4: Perception of a self-induced movement proposed by Held [71]. (Diagram reproduced from [71].)

2.1.2 The Somatic Marker Hypothesis

As discussed in Chapter 1, in order to understand the nature of the relationship between extended experience of the robot and its ability to attain proactive behavior, one of the subsidiary research questions was set up to determine whether *somatic markers* can help a robot towards this goal. The somatic marker hypothesis, proposed by Damasio [45], describes the role of emotion on decision-making. When an animal (mammal) interacts with the world, emotional cues in the environment (e.g., loud noise, snake, fearful facial expression, etc.) are discerned by amygdalae [3, 95, 96, 199], elements in the medial temporal lobe that reside adjacent to hippocampi (Figure 5). Damasio [45] conjectured that some of the responses triggered by emotional cues arrive at the somatosensory cortex, an area in the parietal lobe where physical sensations such as touch, pain, and pleasure are registered. These emotional responses are converted into somatosensory signals conceivably through the bodily pathway [96] (i.e., via the hypothalamus (hormonal) and/or brainstem/spine (neural)) or direct wiring within the brain [46].

The significance of emotionally induced somatosensory signals is that they can be incorporated into episodic memories. More specifically, before a memory is composed in the hippocampus, sensory signals from the somatosensory cortex as well as other types of sensory signals from different cortical areas are assembled at a transitional cortex[2] and form a single integrated representation [96, 166] (Figure 6). Damasio [45] referred to this assortment of the sensory signals as *dispositional representation* and the emotionally induced somatosensory signals encapsulated in a dispositional representation as *somatic marker*. His chief premise is that, by being embedded in the memories, the somatic markers helps an animal selecting an action that is expected to yield the most preferable outcome based on similar situations encountered before.

[2] The perirhinal and parahippocampal areas as well as the entorhinal cortex [96, 166].

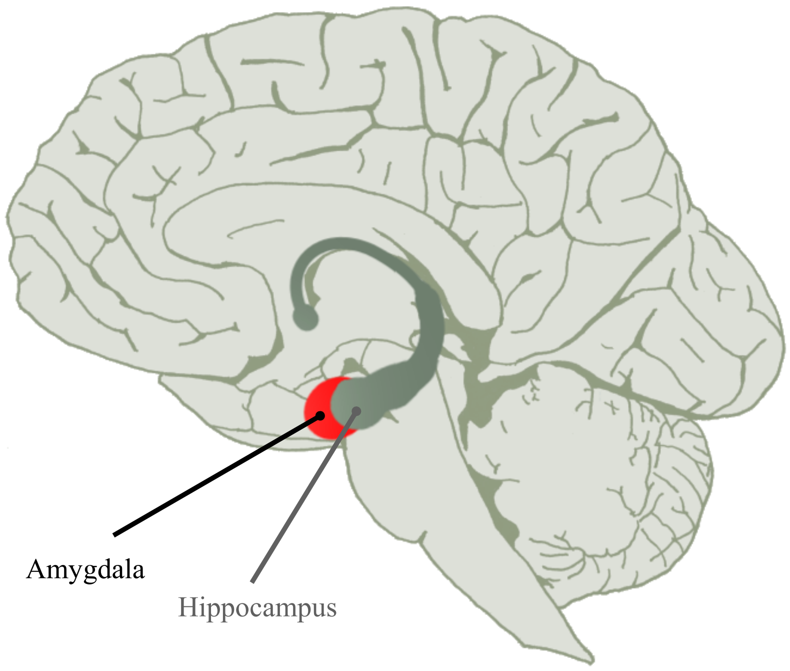


Figure 5: The amygdala in a human brain.

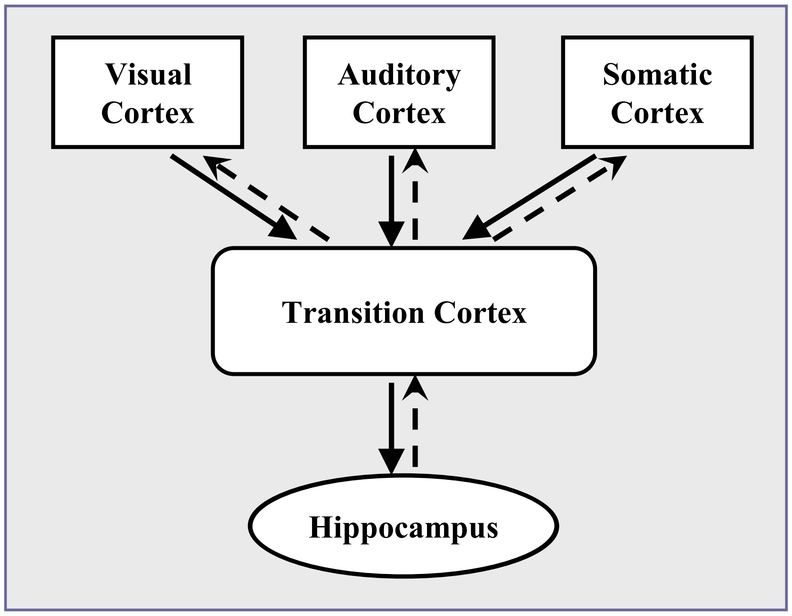


Figure 6: The pathways between the sensory cortical areas and the hippocampus. (Diagram reproduced from [96].)

Damasio [46] presumed that, in humans, such anticipatory decision-making guided by previous emotional episodes is orchestrated by the prefrontal cortex. For example, Bechara et al. [23] conducted a gambling experiment using patients whose ventromedial prefrontal cortices had been damaged. The subjects were asked to draw cards from two types of decks. Each card was denoted with an amount of money that could be either a gain (reward) or loss (punishment) to the subjects. The first type of the decks consisted of a mixture of mild reward cards and severe punishment cards. The second type of the decks was comprised of only minor reward and moderate punishment cards, but it was set up to yield a more profitable net-gain in the end than the first type. After a number of trials, the normal subjects had successfully learned to pick the cards from the second type of the decks in order to maximize their long-term profit. However, the patients with damaged ventromedial prefrontal cortices could not learn to do so. A subsequent skin conductance response (SCR) study [24] indicated that an anticipated monetary loss influences the human body physiologically the same way as an actual monetary loss does. Furthermore, in a similar experiment [19], damages to other components in "the somatic marker circuitry" such as the insular/somatosensory cortex or amygdala were also found to disrupt making lucrative/anticipatory judgments.

Note that the functionality of somatic markers is also approximated by our computational model developed in Chapter 3. As the embedded somatosensory signals were shown to help an animal determine the expected utility of its current action, in our model, reward signals encoded within an episodic memory are also designed to help compute the expected utility of its current action.

2.2 Anticipation

As discussed in Chapter 1, the primary research question of this dissertation is to understand how lifelong experience of a robot influences its ability to anticipate as well as the way it improvises its actions. In this section, we review issues related to anticipatory behavior. *Anticipation* is defined, here, as one's ability to assess the current state, predict the future consequence of the situation, and execute an action to have a desired outcome based on the determined assessment and prediction. Anticipation has been studied by a number of researchers in a variety of fields such as biology, psychology, physiology, engineering, artificial intelligence, robotics, economy, and music. For example, classical conditioning in physiology, concerning the relationship between environmental stimulus and anticipatory behavioral response, has been investigated for more than a century [125]. Recently, the European Commission funded a three-year research project called *Mind RACES: from Reactive to Anticipatory Cognitive Embodied Systems*, in which certain aspects of anticipatory systems such as behavior, perception, learning and emotion were investigated [56, 79]. The project has yielded several dozen papers, almost half of which were related to behavioral aspects of anticipation (e.g., [18, 41, 78, 86, 126, 127]). In this section, some noteworthy research on anticipation that is especially relevant to the main computational model of this dissertation is reviewed.

2.2.1 Anticipatory Systems

In terms of understanding anticipation in physically realized systems, Rosen [143] laid out a foundational framework for anticipatory systems. The diagram in Figure 7 depicts the concept of an anticipatory system proposed by Rosen. The labels S, M, and E in the figure are for *object system*, *model*, and *effector*, respectively. More specifically, S represents the main system (e.g., microorganism, animal, regional economy, etc.) that has to be controlled, so that it can arrive at some desirable state. M is a model of the target system and is capable of foretelling in what state it is going to be next given a current condition. E is an effector that interacts with either the target system itself or the surrounding environment in order to influence the outcome. According to Rosen [143], the functionality of an anticipatory system is supposed to: (a) do nothing if the model predicts that the target system is likely to stay in a "desirable" course; or (b) activate the effector to correct the "trajectory" of the target system if the model warns that an undesirable outcome is imminent. A prominent property of anticipation systems is that, unlike a reactive system, which executes actions simply as a response to currently observing stimuli, it reacts to a state that is expected to happen in the future.

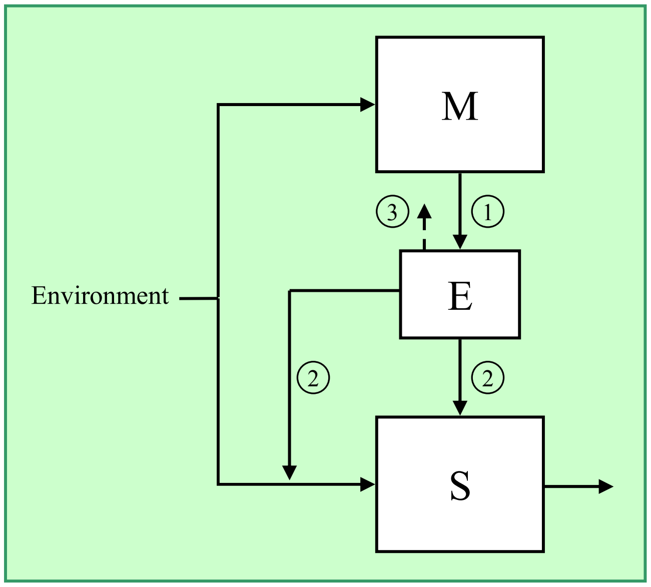


Figure 7: Rosen's Anticipatory system where M = Model, E = Effector, and S = Object System. (Diagram reproduced from [143].)

Note that, in our computational model of proactive intelligent behavior for robots (Chapter 3), the notions of *robot*, *episodic memory*, and *behavior* correspond to Rosen's S, M, and E, respectively. In other words, based on the world model stored in episodic memory (M), appropriate behaviors (E) that are expected to lead the robot (S) to the most rewarding situation are executed.

2.2.2 Anticipatory Behavioral Control

A conceptual framework for anticipatory behavioral control (ABC) was proposed by Hoffmann [72]. The ABC framework is based on a mixture of findings from the field of psychology such as James [77] and Tolman [181]. For example, it assumes that the relationship between action and effect is more relevant to how animals behave in the environment than a mere stimulus-response relationship, the notion attested to by the behaviorists. For example, when an animal moves closer to an object, the image of the object will appear to be looming. The ABC framework asserts that the animal's brain automatically predicts this looming image as soon as the animal starts moving towards the object. If the prediction is correct, the association between the action (moving closer) and the effect (looming image) is further strengthened (Figure 8). This action-effect association is remembered in terms of a situational context. It should be noted that this concept, which Tolman [181] referred to as "means-end-expectation", is comparable to Held's concept of perception via re-afferent signals [71] as discussed above (Section 2.1.1).

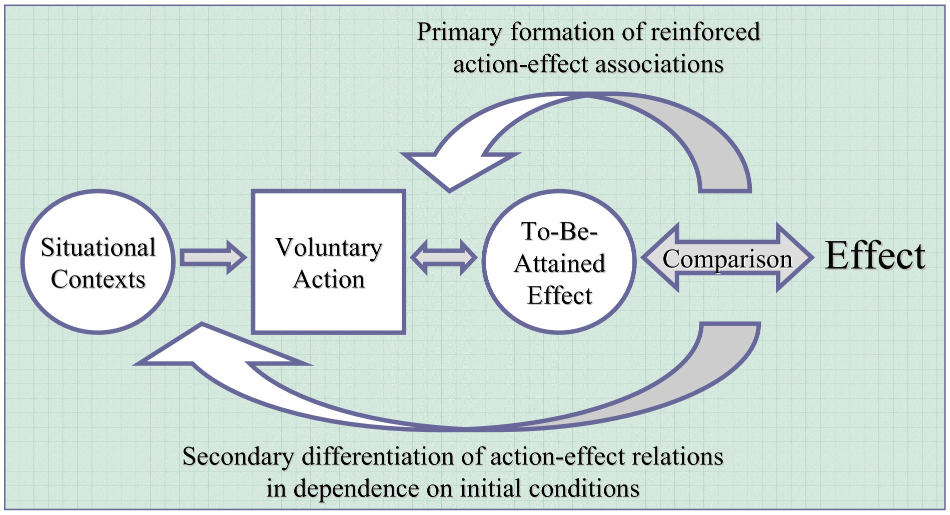

Primary formation of reinforced action-effect associations
Situational Contexts
Voluntary Action
To-Be-Attained Effect
Comparison
Effect
Secondary differentiation of action-effect relations in dependence on initial conditions

Figure 8: Hoffmann's anticipatory behavioral control (ABC) framework. (Diagram reproduced from [72].)

Based on the ABC framework, Stolzmann [167] developed a machine learning algorithm called *anticipatory classifier system* (ACS). An ACS builds a model of the environment by incrementally learning condition-action-effect relationships from experience, assuming that the state space is fully observable and the state transitions are deterministic. With an experiment using a simulated robot, Stolzmann [168] demonstrated that an ACS permits latent learning (i.e., relationships among environmental stimuli can be learned before a reinforcer is presented). By incorporating reinforcement learning formulae, ACS was further adapted by Butz and his colleagues [39, 40] to attain optimal policies in Markov decision process (MDP) problems.

Note that our computational model of proactive intelligent behavior for robots (Chapter 3) does not explicitly incorporate the ABC framework as it is. Rather, in our model, the ABC framework is implicitly integrated into the representation of an episodic memory. More specifically, an episodic memory stores a series of action-perception pairs in accordance with what the robot experienced. Hence, by executing an action specified in episodic memory, the consequent perception can be deduced by tracing the contents of the episodic memory. Indeed, our computational model exploits this property to attain anticipatory behavior.

2.2.3 Hippocampally Inspired Approaches

As the main computational model of this dissertation (Chapter 3) was inspired by how a mammalian hippocampus functions, numerous systems have been developed by researchers that were also influenced by the noble findings from the hippocampi studies. While many of those researchers were interested in developing high fidelity models of the hippocampal formation itself, some of them have attempted to frame their research in terms of its anticipatory behavioral implication.

Recall, for example, that one of the presumed hippocampal functions is novelty detection. Based on a recalled memory, current sensory signals are predicted and compared against actual ones. Schmajuk's hippocampal model [150-152, 193] was designed to reproduce such an anticipatory property and implemented within a navigational system using a neural network (Figure 9). In this system, a simulated animal either searches (random movement) or approaches a motivationally driven goal object. By referring to the "cognitive map" that encodes the topology of the environment, it attempts to predict what is going to be perceived before moving into the next distinct location. If the prediction fails, the cognitive map is updated, and the simulated animal moves towards a random direction. If the prediction turns out to be correct, it moves towards the goal (if visible) or a closest distinct location that is presumably in the shortest path to the goal. Schmajuk and Thieme [152] claimed that their navigational system can solve Tolman and Honzik's detour problem [182] (Figure 10). More specifically, even when an obstacle (blockage b in the figure) is suddenly introduced in the middle of an accustomed route in the maze, if sufficiently trained, the simulated rat can take an optimal route ($1 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 8$) instead of the non-optimal one ($1 \rightarrow 2 \rightarrow 3$).

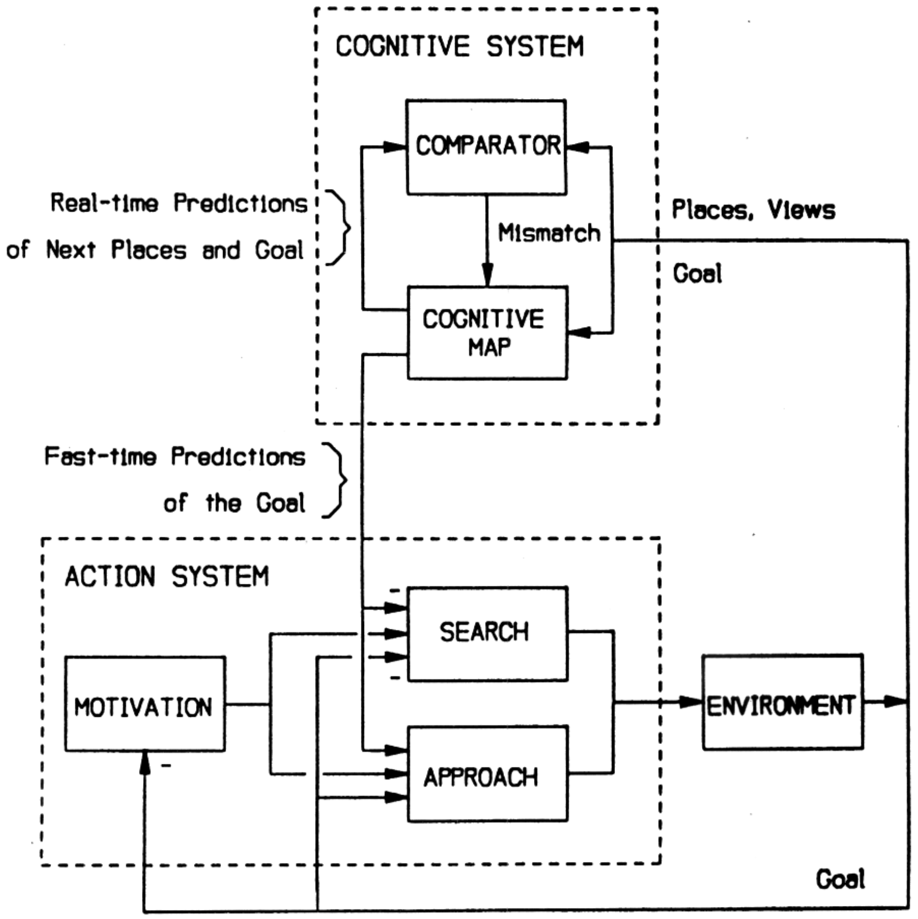


Figure 9: A navigational system proposed by Schmajuk and Thieme. (Diagram reproduced from [152].)

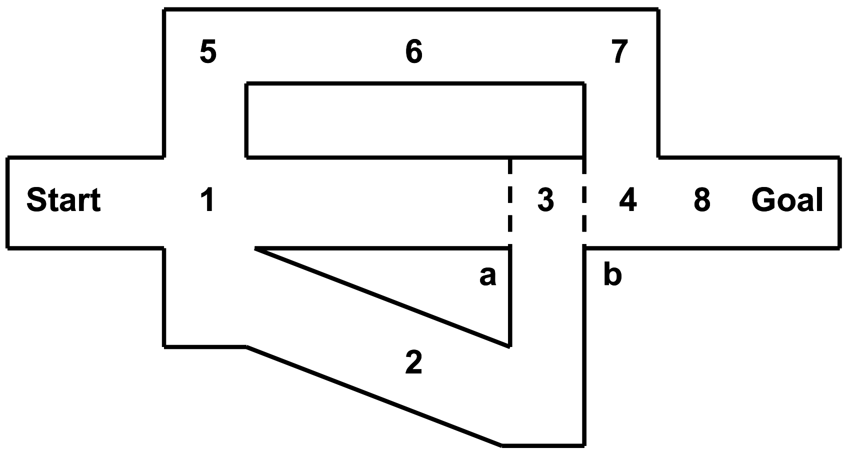


Figure 10: Tolman and Honzik's maze used by Schmajuk and Thieme's detour experiment. (Diagram reproduced from [152].)

From a high-level perspective, this approach is related to our computational model (Chapter 3) as the process of building a model of the world is driven by prediction. The difference is that, in Schmajuk's approach, the model (cognitive map) represents a topology of the environment while, in our approach, the model (episode) represents a temporal sequence of events that a robot has experienced. While both models can be utilized to solve navigational tasks, by encoding spatial information explicitly, Schmajuk's approach can perform spatial reasoning such as finding an optimal alternative route when a regular path is blocked. On the other hand, in our approach, while reflexive detouring can be invoked by the improvisation process (as demonstrated in one of the experiments in Chapter 5 (Section 5.3)), the optimal path is not guaranteed since the model is based on experience, not topology. However, the advantage of this approach is that the task is not limited to spatial navigation. It can also be applied for non-spatial tasks. It should be noted that Tolman and Honzik [182] did not conclude that real rats possess the reflexive detour ability to determine an optimal alternative route in response to a sudden obstruction. In fact, their experiment failed to show conclusively that the rats are able to find an optimal detour route when a regular path becomes unexpectedly impassable.

Mataric [107] proposed a topologically organized distributed map for a mobile robot, which was also inspired by how the hippocampus of a rat operates [108]. Each node in the topological network represents a landmark (Figure 11), and nodes can communicate among themselves through spreading activation. It is anticipatory in a sense that, when a goal object is given, each node can suggest a real-time procedure for the robot to navigate through the environment in order to reach the goal. Mataric [107, 108] implemented this map within a subsumption architecture [31] framework, and it was shown that the robot can successfully reach goals in the environment without colliding with obstacles. With reference to Mataric's work, Kuipers and Byun [93] notably proposed a similar notion of a topologically organized map, in which control strategies are embedded within its representation to advise how to traverse in the environment. Each node in the topological map represents a distinctive place in the environment, which is detected by a hill-climbing method; and the connectivity among those nodes is described in terms of control policies. Kuipers and Byun's topologically organized map was successfully implemented on a real robot by Lee [98].

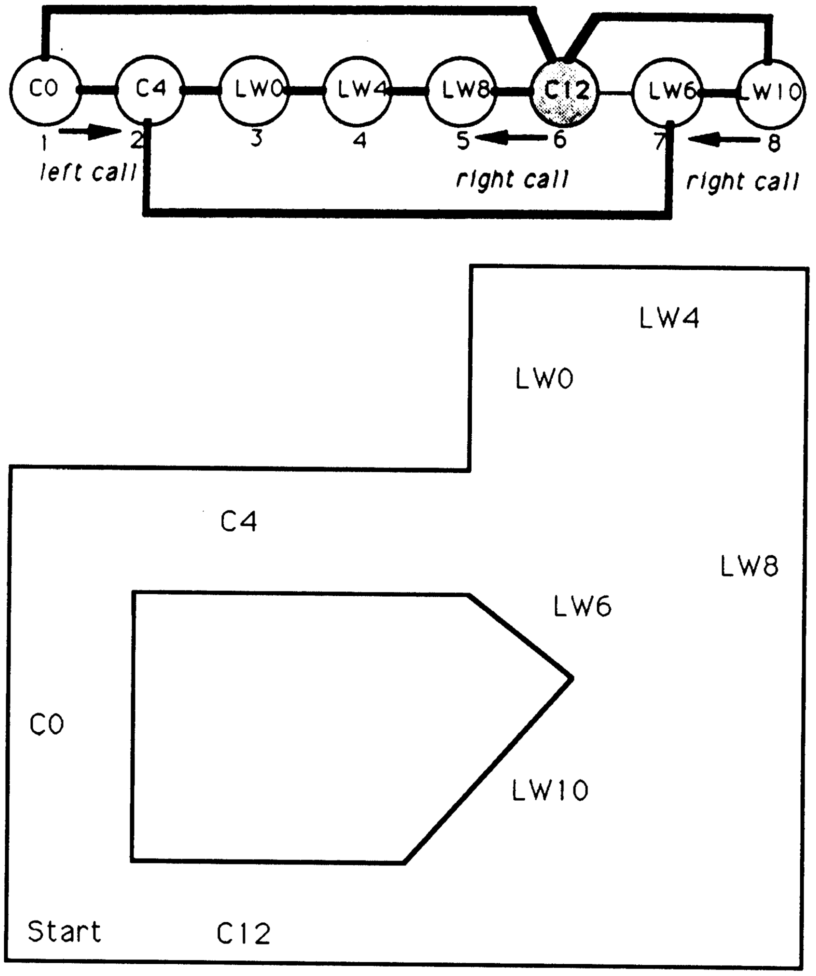


Figure 11: Mataric's topologically organized distributed map [107] (top) and the corresponding environment (bottom). (Diagram reproduced from [107].)

Note that the uniqueness of the topological maps discussed above is that, unlike conventional maps (whether metric or topological), these maps encode behavioral information within the representations. As suggested by the theory of event coding (TEC) [74] discussed above (Section 2.1.1), the joint representation of perceptual and behavioral information seems to help a robot behave in an anticipatory fashion. Indeed, our computational model of proactive intelligent behavior for robots (Chapter 3) also integrates the behavioral information within the representation of the world model (episodic memory), enabling the computation of anticipatory behavior.

2.3 Improvisation

Recall that the primary research question of this dissertation is to understand how lifelong experience of a robot influences its ability to anticipate as well as the way it improvises its actions (see Chapter 1). In this section, issues related to improvisational behavior are reviewed. *Improvisation* is defined here as one's ability to promptly detect an unanticipated circumstance of the situation, find a fallback solution to deal with the situation, and execute an action to have a desired outcome. One of the essential aspects of improvisation is promptness. For example, according Bailey [17], music composer Frederic Rzewski once asked jazz musician Steve Lacy if he could describe the difference between composition and improvisation in fifteen seconds. Lacy replied, "in fifteen seconds the difference between composition and improvisation is that in composition you have all the time you want to decide what to say in fifteen seconds, while in improvisation you have fifteen seconds," taking exactly fifteen seconds to answer. In other words, there is always a time constraint when performing improvisation. In case of extreme events such as earthquakes or terrorist attacks, such a time constraint becomes extremely important for an emergency response organization to produce workable solutions when dealing with dynamic problems [112]. As each improvised jazz performance is a product of the knowledge gained through lifelong preparation [26], without prior knowledge/skill gained before, sufficient improvisation cannot be attained. Improvisation is indeed relevant to *intelligence* as defined by Piaget [42, 128].

While improvisation can be observed in a variety of art forms such as music, dance, and theater, research on improvisation seems to be premature when compared to, for example, studies on anticipation. Even so, musical improvisation (jazz improvisation in particular) has been studied by a substantial number of researchers. In this section, musical improvisation and some computational models of jazz improvisation are discussed first. Some limited examples of improvisation in artificial intelligence are then reviewed.

2.3.1 Musical Improvisation

Regardless of genre, music performers often incorporate some degree of improvisation in their artistic expression. For example, when traditional Persian and Indian music are played, improvisation becomes an essential element of the performance; on the other hand, improvisation plays only a nominal role in traditional Japanese music as it follows original scripts very tightly [132]. Today, jazz is certainly one of the most popular music genres that profoundly integrate improvisation, and it has been a subject of significant academic research.

Kernfeld [83], for example, identified that there are at least three forms of jazz improvisations: namely, *paraphrase*, *formulaic*, and *motivic* improvisations. In a paraphrase improvisation, a performer produces a variation of an existing tune in a way that it carries a new or "jazzy" sound, but the original melody is still recognizable to the audience. On the other hand, in a formulaic improvisation, the original melody becomes hardly recognizable even though the original musical structure is unchanged. For each fragment of the original melody, a performer tries to find an alternative melodic fragment that is most appropriate for that juncture. Each jazz performer is supposed to maintain a repertoire of such melodic fragments, known as motives or "licks", accumulated through out his/her lifetime. A motivic improvisation is a rather systematic exploration of new sounds. During a session, a performer creates a new tune from less than a few selected motives by methodically transforming the arrangement of their notes (e.g., raising or lowering tones, stretching or compressing intervals).

Note that, out of these three forms, our computational model of proactive intelligent behavior for robots (Chapter 3) addresses the formulaic aspect of improvisation. More specifically, when a robot encounters an unanticipated situation when performing a familiar task, the recalled episode that was being used to perform the task is subdivided into fragments. The robot then attempts to find an alternative episodic fragment that is most appropriate for that juncture.

2.3.2 Computational Jazz Improvisation

Notice that, in any of these jazz improvisation forms, performers do not extemporaneously create completely new pieces from scratch. They utilize some form of basic outlines to derive their improvisations. Pressing [132] refers to such an outline as a *referent*. More specifically, a referent is "a set of cognitive, perceptual, or emotional structures (constraints) that guide and aid in the production of musical materials" [134]. Pressing [133] proposed a computational model of human improvisation that incorporates the notion of referents. By partitioning a single improvisational performance into a series of segments[3], the improviser constantly determines what behavior or "schema" has to be executed in the next

[3] Pressing [133] referred them as *event clusters*.

segment. The decision is made based on various factors such as referents, current goals, properties of current and previous segments, and long-term memory storing relevant past experiences. Grachten [65], for example, took this referent/constraint based approach to implement a computer program called JIG. Incorporating case-based reasoning (see Section 2.4.2 below) and probabilistic sampling methods, JIG achieves formulaic jazz improvisation by progressively retrieving appropriate motives and adapting their properties (e.g., pitches and durations of notes) to match with current constraints. The referent-based improvisation has been also applied to non-artistic academic fields such as organizational theory. For example, Mendonca [113] developed a cognitive model of emergency management, in which referents guide the deliberation of a workable course of actions to deal with extreme events. Furthermore, our computational model of proactive intelligent behavior for robots (Chapter 3) also utilizes the notion of referents. More specifically, when improvisational behavior is desired, the representation of the most relevant episodic memory is converted into a more abstract form, a *referent*, and it is used as a basic outline to derive the improvisational behavior.

According to Johnson-Laird [80], only three types of algorithms can truly facilitate creativeness: namely, *neo-Darwinian*, *neo-Lamarckian*, and a hybrid of the two. Inspired by the natural selection process in biology, a neo-Darwinian algorithm generates a new piece by randomly blending different pieces together. Based on some evaluation criteria, if this new piece sounds sufficient, it is kept for a future generative process. In a neo-Lamarckian algorithm, on the other hand, a new piece is derived from some relevant domain knowledge. In this case, unlike the neo-Darwinian approach, the evaluation process is not required. The third type, a hybridized version of the two, utilizes expert knowledge in both composition and evaluation processes. Johnson-Laird noted that, even though it is fated to produce a substantial amount of unwanted pieces due to the randomness in the production process, the neo-Darwinian approach might be the only way one can improvise if no expert knowledge is available. On the other hand, the neo-Lamarckian approach produces a new piece by effectively assuming that there is appropriate expert knowledge available to guide the production. However, such an assumption is not always guaranteed to hold. Hence, the aim of the hybrid approach is to overcome the limitations of both algorithms.

Taking the neo-Darwinian approach, Weinberg et al. [196], for example, constructed a two-arm robot [195] that plays a xylophone in a way that new melodies are extemporaneously adapted from different preexisting melodies that had been pre-generated using a genetic algorithm. For jazz improvisation, however, Johnson-Laird [80] conjectured that the neo-Lamarckian or hybridized version are more reasonable choices. Our computational model (Chapter 3) can be classified as *neo-Lamarckian* since its improvisation is derived from relevant domain knowledge (experience), and randomness plays an insignificant role in the process.

2.3.3 Artificial Intelligence

The role of improvisation is not yet fully explored in the robotics community. Even in the artificial intelligence community, there are only limited examples. Nevertheless, at least two noteworthy studies were conducted on improvisation in the context of AI: one by Agre [4] and the other by Anderson [7]. Conceptually, improvisation is similar to planning in classical AI as both suggest a plan of action to achieve a predefined goal [5]. Agre [4] noted, however, that improvisation is performed when the consequence of actions are not necessarily fully known while, in planning, the information regarding the state space is already predetermined. In other words, improvisation has to work with incomplete knowledge, or, in some cases, some part of the knowledge is deliberately ignored in order to avoid exhaustive computation. The latter cases in fact relate to anytime algorithms [28, 202] in which a solution to a problem is deliberated in an incremental fashion. More specifically, anytime algorithms are constructed in a way that the quality of a solution improves monotonically with respect to the amount of time spent for computation [201]. The computation can hence be interrupted anytime, yielding a solution that is attainable given the time constraint. Likewise, in order to answer the fifth subsidiary research question in Chapter 1, one of the experiments in Chapter 5 was setup to determine the trade-off between promptness and the quality of our proactive behavior computation (Section 5.3).

Anderson [7] conjectured that improvisation can be *weak* or *strong*. If the situation is totally novel, it requires strong improvisation, which means that "deeper reasoning" is necessary upon choosing the action. On the other hand, if the situation is within a familiar domain, some "routine response" (weak improvisation) can be applied quickly. Thus, in order to improvise, an agent has to be able to determine how much the current situation is different from a routine activity and appropriately adjust the routine based on relevant domain knowledge [7]. Note that, our computational model of proactive intelligent behavior for robots (Chapter 3) refers to Anderson's notions of weak and strong improvisations as *anticipation* and (just) *improvisation*, respectively. From weak to strong, our computational method is designed to handle the range of Anderson's terms of improvisation.

Both Agre and Anderson (independently) implemented improvisational agents in their software architecture called Pengi [4, 5] and Waffler [7, 8], respectively. Pengi interacts with a simulated environment in which animated characters interact with certain objects (Figure 12). In order to "kill the bee", for example, the agent does not construct a "plan" as in classical AI planning per se. It instead recalls the interaction it had with the objects. If it can recall (or "visualize" in Agre's term) appropriate past experience and consider that, for example, kicking the ice cube would lead to some desirable consequence, it then executes the action. It should be noted here, however, that the experience that the agent's decision is based on is some precompiled routine activities entered by programmers (i.e., it is not automatically acquired knowledge).

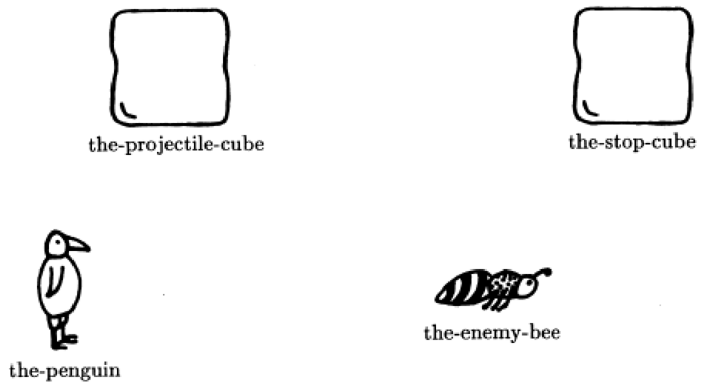
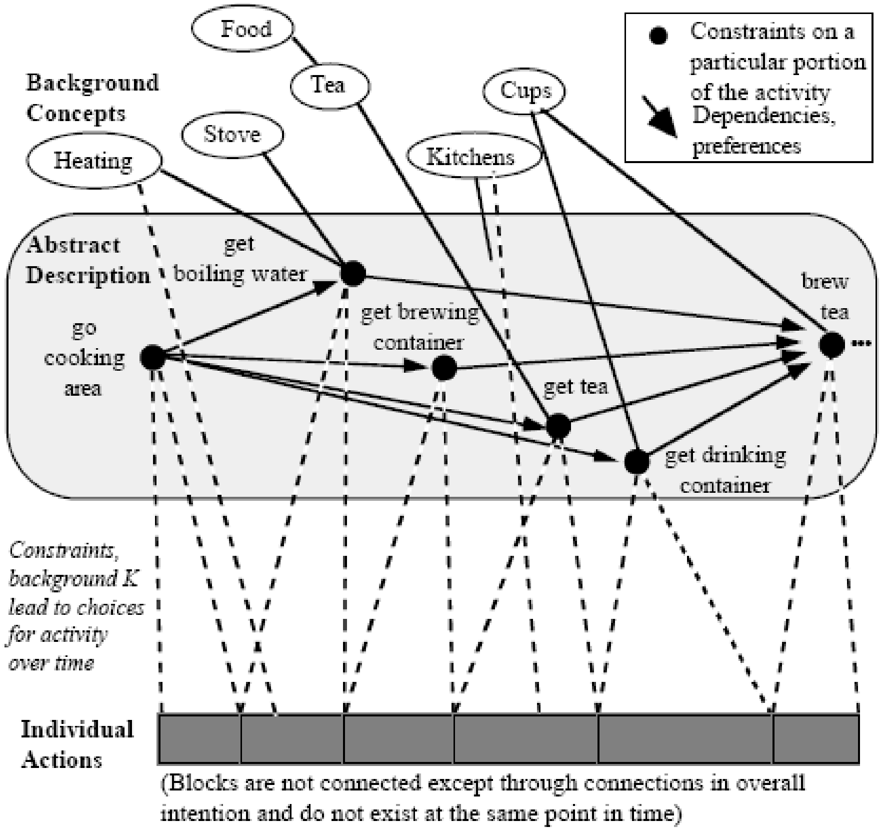


Figure 12: Simulated environment for Pengi. (Diagram reproduced from [5].)

While Agre's Pengi architecture was tested in a game-like environment, Anderson's Waffler works in a simulated kitchen. The diagram in Figure 13 shows how an activity such as "making tea" can be processed via improvisation. As in Pengi, Waffler is also equipped with a repertoire of precompiled routines (plans) that the agent can base its decision on. Furthermore, the agent in Waffler also has built-in domain knowledge that is used to guide the selection of appropriate actions. Anderson [7] calls such knowledge "constraints". Indeed, the emphasis of Waffler is utilization of such constraints during improvisation. Constraints can help the agent narrow down its options or limit the search space. Constraints may also provide additional alternative options that the agent otherwise would have ignored. In other words, constraints influence how an agent retrieves relevant actions from its memory. Furthermore, constraints also affect improvisation on when to execute the action. Anderson [7] noted that delaying deliberation of a plan can provide the agent with more time to search alternatives, but it also could compromise the end result if any underlying assumption being made is time dependent. Thus, in Waffle, the constraints ("utility threshold") also guide the decision on when is good time to deliberate. Note that our computational method (Chapter 3) also utilizes constraints to attain improvisation. Borrowing the concept from computational jazz improvisation (see Section 2.3.2 above), the representation of the most relevant episodic memory for a current goal is abstracted into a basic outline, *referent*, and it is used to constrain the derivation of improvisational behavior.



Food
Tea
Cups
Constraints on a particular portion of the activity
Dependencies, preferences
Background Concepts
Stove
Heating
Kitchens
Abstract Description
get boiling water
go cooking area
get brewing container
get tea
get drinking container
brew tea
Constraints, background K lead to choices for activity over time
Individual Actions
(Blocks are not connected except through connections in overall intention and do not exist at the same point in time)

Figure 13: The process of "making tea" with Waffler. (Diagram reproduced from [8].)

Hayes-Roth and Brownston [70] also proposed a software system, CAIT, implementing similar constraint-directed improvisation. The domain of CAIT is a virtual puppet theater. Compared to Waffler, the underlying mechanism for choosing appropriate actions seems much simpler. However, the main concern of CAIT is human-robot interaction (HRI) as well as multi-agent coordination. In CAIT, a human user specifies a goal of a play in order to make virtual puppets enact. The specified goal along with the internal state of the puppet itself and interpreted states of other puppets become part of the constraints in CAIT. A similar theatrical improvisational system was also proposed by Moraes and da Rocha Costa [116], focusing on the director's role in multi-agent coordination. Note that, even though constraints are indeed used to derive improvisational behavior, unlike CAIT, our computational model of proactive intelligent behavior for robots (Chapter 3) does not particularly address the issues on HRI or multi-agent coordination.

2.4 Relevant Machine Learning Techniques

In this section, various machine learning techniques that are relevant to the main computational model of this dissertation (Chapter 3) are described. In particular, temporal difference learning, case-based reasoning, instance-based learning, partially observable Markov decision processes, and predictive state representation are explained.

2.4.1 Temporal Difference Learning

Temporal difference (TD) learning is a predictive reinforcement learning method that was most notably theorized by Sutton [170]. It is incorporated into our computational model (Chapter 3) in a manner so that the sensor readings are constantly predicted by this learning method in order to detect novelty in the environment. The core idea of TD learning can be explained well by contrasting it with standard supervised learning. In standard supervised learning, an algorithm is employed to generate a hypothetical function that best explains the trends of training data. Typically, such a function is refined iteratively and then used to interpolate missing data points or classify query points into predefined categories. On the other hand, in TD learning, training data points are presented sequentially, and the algorithm generates a hypothetical function that interprets the temporal relationships among them in order to predict unseen future points. Since the function is recursively updated based on previously computed parameters (i.e., bootstrapping), TD learning is considered a form of dynamic programming [20]. The algorithm that is best known for implementing TD learning is TD(λ) [170]. TD-Gammon by Tesauro [174], for example, incorporates TD(λ) into a neural network framework in order to play backgammon, a two-person board game. It has been demonstrated that not only could TD-Gammon outperform another backgammon-playing computer program, it was also able to play comparably well against human world champions [173, 174].

TD learning is often considered relevant to how animals learn association of certain stimuli in the environment [171]. In particular, Sutton and Barto [172] have shown that classical conditioning [125] can be modeled using TD learning. By monitoring dopaminergic signals in primate midbrains, Schultz et al. [153, 154] further concluded that actual neurophysiological activities from expected rewards could be indeed described by a TD model. Parenthetically, the hippocampal model proposed by Foster et al. [60] employs TD learning to learn the spatial coordinates of the environment.

2.4.2 Case-Based Reasoning

Case-based reasoning (CBR) is a class of memory-based problem-solving techniques in which a solution to a current problem is sought in its memory (case library). A case library reserves past problem-solving experiences (cases), so that they can be utilized again in the future. The process of CBR typically involves: 1) retrieval of a case from the memory; 2) customization of the case to create a solution for current needs; 3) execution of the solution; and 4) storage of the solution as a new case. The diagram in Figure 14 shows, for example, a case-based reasoning cycle suggested by Kolodner and Leake [91].

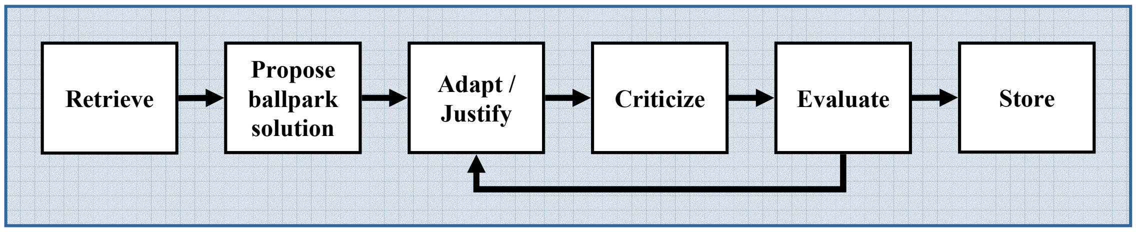


Figure 14: Steps of case-based reasoning, proposed by Kolodner and Leake [91].

One of the essential attributes in CBR is indexing [89]. More specifically, each case in the memory is labeled with an index that uniquely identifies the situation in which the case is used. The postulation is that, if indexing is adequately proceeded, the most suitable solution to the current problem can be promptly presented simply by seeking a case whose index best describes the current problem [89, 91]. For example, the diagram in Figure 15 shows how certain diplomatic meetings are stored in CYRUS, one of the earliest case-based reasoners developed by Kolodner [90]. To save the diplomatic meetings in a way that they can be effectively recalled in the future, Kolodner proposed a memory structure called E-MOP (episodic memory organization packet), which is an enhanced version of Schank's MOP [148, 149]. The E-MOP in the figure comprised with two indexes: namely, participants and topic. If the value of participants is Begin, or the value of topic is Camp David Accords, E-MOP points to the diplomatic meeting that took place at Camp David in 1978, involving the Israeli and Egyptian leaders. On the other hand, if the values of participants and topic are Gromyko and SALT, respectively, the diplomatic meeting to which the E-MOP is referring is the talk on the Strategic Arms Limitation Treaty (SALT) between the United States and the Soviet Union.

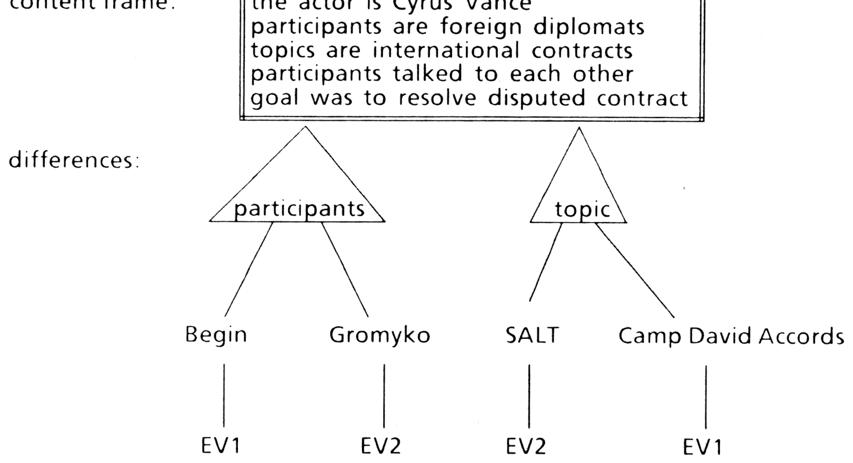

content frame:
the actor is Cyrus Vance
participants are foreign diplomats
topics are international contracts
participants talked to each other
goal was to resolve disputed contract
differences:
participants
topic
Begin
Gromyko
SALT
Camp David Accords
EV1
EV2
EV2
EV1

Figure 15: An example of E-MOPs (episodic memory organization packets) in CYRUS [90], This E-MOP in particular is shown to encapsulate particular diplomatic meetings that took place. (Diagram reproduced from [90].)

CBR is exceptionally relevant to our computational model (Chapter 3) because, as pointed out by Anderson [7], CBR shares a common connotation with improvisation; instead of considering every contingency, which is computationally burdensome, it is more efficient to propose an extemporized solution that is synthesized from previous experiences. For example, in the context of Chinese (Szechwan) cooking, Hammond's CHEF [67] composes a recipe for *stir-fried beef with broccoli* by recalling relevant recipes such as *stir-fried beef with green beans* (i.e., substituting green beans with broccoli).

CBR has demonstrated its efficacy in a variety of domains such as medical diagnosis, legal proceedings, and industrial optimization [89]. In robotics, there are at least three types of CBR applications. The first type is planning. Veloso and Carbonell's PRODIGY [191], for example, constructs a sequence of behavioral actions by referring to relevant past problem-solving episodes. There have been also attempts to apply CBR in path planning [63, 92].

The second type of CBR in robotics applications is regarding reactive navigation. For example, CBR can be employed to propose an appropriate action for the current situation by retrieving a case that best represents the current spatial configuration [142]. To identify the situational similarities more fluidly, some have also tried to incorporate temporal information. Ram and Santamaria [137] referred to this type of CBR as *continuous case-based reasoning*. More specifically, a robot is set up to monitor how the sensor readings change with respect to time. The case-based reasoner then computes the temporal similarity of the current sensory sequence with respect to the ones experienced in the previous episodes (cases). After identifying a matching case, it retrieves the behavioral parameters (e.g., ***goal gain***, ***noise gain***, or ***sensible distance***) from the case in order to adapt the control strategy to the current environment [100, 136, 137].

Finally, the third type of CBR applications in robotics pertains to human-robot interaction (HRI). For example, Endo et al. [55] implemented a high-level mission planning tool that assists users in specifying multi-robot missions. The planning tool utilizes CBR to retrieve the most suitable mission from the case library that best matches with the current specification of the mission. In this line of work, CBR was further applied to repair a faulty component of the executed mission in case of failing to accomplish its intended goal [120].

Our computational model (Chapter 3) is related to the second type of CBR that deals with reactive navigation. Ram and Santamaria's continuous CBR [137] is perhaps the most relevant to our method as the temporal aspect of the sensory information is utilized in both approaches. However, the main distinction is that, in Ram and Santamaria's work, CBR retrieves cases (episodes) as well as proposes appropriate actions while, in our method, CBR[4] only retrieves cases, and a partially observable Markov decision process (Section 2.4.4) is

[4] To be precise, our method employs instance-based learning (see Section 2.4.3).

separately employed to identify the most suitable actions. In fact, this type of hybridization can be considered a contribution to the field of case-based reasoning. Note that hybridization of CBR and reinforce learning has been successfully applied to controlling simple physical systems (e.g., pendulum) [145] as well as a real-time strategy game [159]. Likewise, our method may well be employed to such applications.

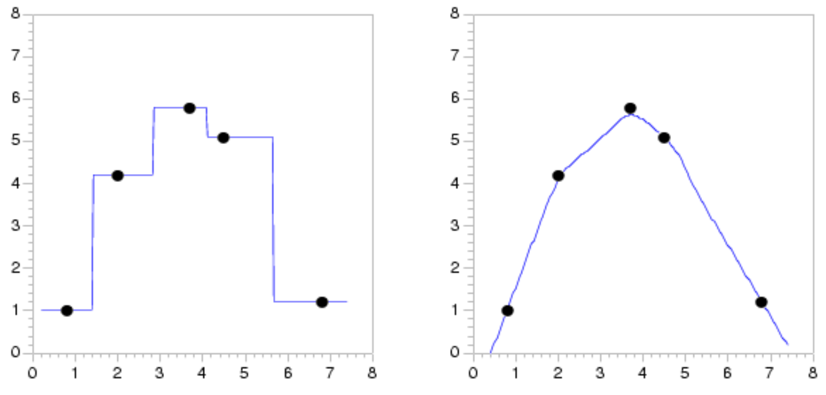
2.4.3 Instance-Based Learning

Our computational model of proactive intelligent behavior for robots (Chapter 3) employs instance-based learning to retrieve relevant episodes from its memory. Instance-based learning is a memory-based learning method, and it has a sensible resemblance with case-based reasoning. Both CBR and instance-based learning are often referred to as a "lazy learning method" as they retain training data in its original form and postpone generalization of the data until a solution to a new problem is asked to be deliberated [115]. The distinction between the two is somewhat vague. Mitchell [115] construed that the difference is in how the data (case/instance) is represented in the memory; while a case in CBR consists of symbolic notations that encapsulate highly abstract descriptions of the world, an instance in instance-based learning is comprised of numerical values that represent certain points in a *n*-dimensional Euclidean space. The major components of instance-based learning are similarity and classification functions [6]. The similarity function computes how similar between a query point (current problem) and a point in the training data (previous episodes) is. Based on the output of the similarity function, the classification function determines in what category the query point belongs.

Because of the mathematical properties gained by representing the query point and instances within a Euclidean space (assuming the parameters are easily quantifiable), the *k*-nearest neighbor algorithm is often employed to implement instance-based learning. For example, McCallum [109, 110] applied the *k*-nearest neighbor algorithm to help uncover hidden states in a hidden Markov model (HMM) problem. More specifically, based on the recent action, perception, and reward histories, the algorithm picks k data points from the memory that are presumably the closest representations of the current state. The state parameter (Q-value), which is used to calculate an optimal policy, is determined by averaging Q-values from those k instances. This approach of instance-based learning has been further adapted by Littman et al. [102] to find repair policies in cases of computer network failures.

Note that our computational model of proactive intelligent behavior for robots (Chapter 3) can be viewed as comparable to McCallum's method. In both methods, k instances that are closest representations of a current state are selected from the memory and used to determine a current policy. The difference is that, in our method, a goal is used to retrieve relevant instances while McCallum's method retrieves instances based on the recent action-perception-reward sequence. Furthermore, upon determination of the current policy, a model-based approach (Bellman's equation [25]) is utilized in our method in contrast to the model-free approach (Q-value) employed in McCallum's method.

Another known example of algorithms that have been employed to implement instance-based learning is locally weighted regression. As in the *k*-nearest neighbor algorithm, locally weighted regression also takes advantage of the Euclidean distances of the points in training data with respect to a query point. Instead of considering a fixed number (k) of points to interpolate the query point, however, in locally weighted regression, a predefined distance is used as a threshold to opt for relevant points in the training data. To attain a solution, the contribution from each point is (inversely) weighted based on the Euclidean distance from the query points. The diagram in Figure 16 shows the difference between the nearest neighbor and locally weighted regression approaches in terms of interpolating a function using five data points.



Nearest Neighbor **Locally Weighted Regression**

Figure 16: Interpolations of a function using five data points via the nearest neighbor (left) and locally weighted regression (right) algorithms. (Diagram reproduced from [147].)

Locally weighted regression has been applied to analyze data in a range of domains such as biology, chemistry, economy, meteorology, image processing, and speech recognition [15]. Most notably, Schaal and Atkeson [16, 147] implemented a robot that plays devil sticks (juggling). In order to keep the target object (baton) in space, locally weighted regression was used to identify control policies of two control sticks in real-time.

While biasing the computation of the current policy based on the Euclidean distance between the relevant instance (episode) and a query point (goal) can certainly be explored in the future, the current implementation of our computational model (Chapter 3) does not incorporate locally weighted regression.

2.4.4 Partially Observable Markov Decision Process

The objective of a Markov decision process (MDP) problem is to find the best policy, that is, to map an action for a current state that can maximize expected rewards. The assumption here is that the probability of transitioning from any one state to another is known within the state space, and such transition probabilities strictly follow the Markov property. In other words, when transitioning from state A to state B, for example, complete information that is necessary to compute its transition probability can be found within just state A. Hence, information that was available before arriving to state A does not influence how to arrive at state B. While solving a standard (stochastic) MDP problem itself suffers from a computational complexity as the state space broadens, solving a partially observable MDP (POMDP) problem is known for its severe computational burden because the current state cannot be assessed directly and therefore has to be estimated first. Unfortunately, when dealing with real robots, the assumption of complete observability cannot be guaranteed because various types of uncertainties influence the robot's state [88]. Hence, a challenge for the robotics researchers has been to find a computationally tractable solution while working in a partially observable environment. As described in Chapter 3, our computational method is designed to handle POMDP problems efficiently.

While McCallum [109, 110] has applied an instance-based learning method to estimate the current state as discussed above, standard approaches to deal with POMDP problems are to use Bayes' rule. Most notably, Cassandra et al. [44] laid out one of the first Bayesian-based frameworks for the artificial intelligence community. More specifically, as shown in Figure 17, a belief state (i.e., a state that best represents the current situation) is at first estimated from the current observation, previous belief state, and previously executed action. Note that state estimation is indeed the step that is computed probabilistically, incorporating Bayes' rule. The process is recursive in a sense that the result from the previous estimation is used to compute the current value. Once the belief state is identified, the second step is to find the most advantageous policy for that state (i.e., to identify the best action that maximizes expected rewards). In robotics, Koenig and Simmons [88] developed Xavier, a computational architecture for robot navigation that incorporates the POMDP model. Representing the environment with a topological map, in their method, the optimal policies are refined offline through the Baum-Welch algorithm [21, 135].

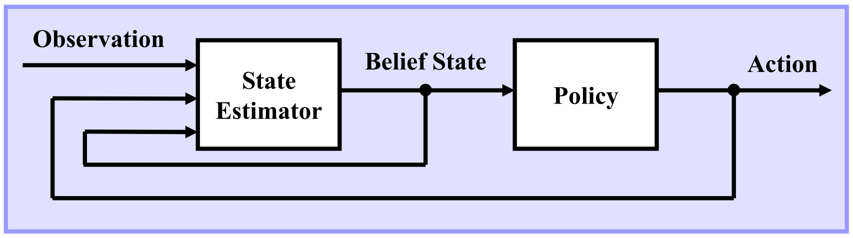


Figure 17: Computational steps proposed by Cassandra et al. [44] to deal with partially observable Markov decision process (POMDP) problems. The first step (state estimation) is computed probabilistically. (Diagram reproduced from [44].)

Various attempts have been made to reduce the POMDP computational load. One way to accomplish such reduction is to represent the state space hierarchically. For example, in Theocharous and Mahadevan's approach [175], the state space was abstracted based on spatial granularities. Through their experiment using a real robot, the hierarchical dissection of the state space was proven effective especially when covering a large area. Likewise, Pineau et al. [130] tackled a POMDP problem by decomposing the action space hierarchically. The application of their method on a real robot in nursing homes has successfully provided necessary assistances to the elderly residents.

Another approach that has been taken to reduce the POMDP computational load is to use sampling. Thrun [176], for example, has demonstrated that Monte Carlo sampling over belief space can attain solutions that are near optimal. Conversely, Pineau et al. [129] proposed a sampling method that takes advantage of how a trajectory of the value function is shaped. More specifically, at each computational cycle, a finite set of sampling points that are enough to recover the shape of value function through a piecewise linear function is selected. After updating state parameters on those points, by stochastically forecasting the path of its future trajectory, the points that are projected to be no longer relevant are eliminated, reducing the size of the state space that affects computation. Note that this form of sampling is known as *trajectory sampling* [171].

Note that our computational model of proactive intelligent behavior for robots (Chapter 3) also attempts to reduce the POMDP computational burden in several ways. State space abstraction, action space abstraction, and trajectory sampling are among those methods used toward that end. The details of our method to reduce the load of the POMDP computation are explained in Chapter 3.

2.4.5 Predictive State Representation

In order to describe a state of a certain dynamic system (e.g., robot), Littman et al. [101] introduced a representation called *predictive state representation* (PSR). In terms of assessing a current state within a predefined state space and finding an optimal policy for that state, PSR can be employed alternatively to instance-based learning or partially observable Markov decision process. Recall that, in McCallum's implementation of instance-based learning [109, 110], the policy was determined by averaging the state parameters (Q-values) from k (neighboring) states that most effectively encapsulate the recent action-perception-reward sequence. On the other hand, in standard POMDP approaches, the current state is determined by recursively applying Bayes' rule [44, 88]. Littman et al. [101] referred to the former approach (McCallum's) as *history-based approach* and the latter one (POMDP) as *generative-model approach*; and suggested that PSR is a combination of both. In other words, while PSR utilizes history to estimate a current state as in a history-based approach, the state parameters are recursively updated as in a generative-model approach. Unlike McCallum's approach, however, in PSR, each state explicitly maintains a predicted sequence of what may be observed in the future given a predefined sequence of past actions. In McCallum's approach, such information concerning possible future outcomes is implicitly encoded as the Q-values. On the other hand, in standard POMDP approaches, while the state representation does not usually address a predicted future explicitly, such information can be computed by, for example, employing Bellman's equation [25] since the probability distribution over the state space is known.

The underlying assumption of PSR is that representing an expected future consequence of past actions within a state should be more advantageous than the conventional state representations. It should noted, however, that, while the theoretical foundation of PSR has been soundly established [162], the practical implication of this approach has not yet been fully demonstrated. The preliminary empirical results have shown that PSR is at least comparable to POMDP [76, 163].

Note that, from a high-level perspective, our computational method (Chapter 3) can be considered comparable to PSR as both methods combine history-based and generative-model approaches. More specifically, our method is generative in a sense that it computes POMDP solutions recursively. However, unlike PSR, our method does not explicitly encode a history within the representation of a state (event). Instead, each state can infer such information from the state space (episode) formed in a unidirectional temporal linear chain fashion.

2.5 Summary

As the main computational model of this dissertation (Chapter 3) was inspired by how a mammalian brain works, the research related to episodic memories and hippocampal functions as well as the somatic marker hypothesis was first examined in this chapter.

Episodic memory is a form of declarative memory, storing a particular episode of experience that can be later retrieved and recounted in accordance with how it initially proceeded [184]. Our computational model is founded upon a neurophysiological hypothesis in which the hippocampus is considered to construct a "memory space", a collection of discrete episodic memories; it is the same school of thought that defines *episodic memory* as "a sequence of event representations, with each event characterized by a particular combination of spatial and nonspatial stimuli and behavioral actions" [51]. By integrating both sensory and behavioral information into a common representation, this notion of *event* is also in accordance with the theory of event coding (TEC) [74] found in cognitive psychology.

Somatic markers are emotionally induced somatosensory signals embedded within episodic memories, and they have been shown to help an animal determine the expected utility of its current action [45]. Our computational model approximates the functionality of somatic markers by integrating reward signals into its foundational data structure (episodic memory) that stores the experience of a robot.

As the primary research question of this dissertation is to understand how lifelong experience of a robot influences its ability to anticipate as well as the way it improvises its actions (Chapter 1), we also reviewed issues regarding anticipatory and improvisational behaviors. For example, the anticipatory behavioral control (ABC) framework [72] considers how the association between voluntary action and predicted consequent perception plays an essential role for attaining anticipatory behavior. Likewise, to process anticipation, our computational model implicitly incorporates the ABC framework in a way that a temporal sequence of action-perception pairs is encoded within the representation of episode in accordance with what a robot experienced. While some researchers have utilized a joint representation of action and perception within topologically organized spatial maps to gain anticipatory behavior (e.g., [107, 152]), our computational model is unique in a sense that it encodes both behavioral and perceptual information within a temporally organized memory structure (episodic memory). By focusing on the temporal relationship instead of the spatial one, our computational model can be applied even for non-spatial tasks.

In terms of improvisation, our computational model can be categorized as *neo-Lamarckian* [80] since the computation of improvisational behavior is driven by relevant expert knowledge (experience) as opposed to *neo-Darwinian* in which a new idea is derived from a mixture of randomized old ideas. To guide the derivation of improvised behavior, our computational method abstracts a basic outline known as a *referent* [133] from episodic memory. Taking the *formulaic improvisation* approach from jazz music [83], the referent is subdivided into fragments, and the algorithm seeks the most suitable episodic fragment for the moment from its memory. Note that, upon derivation, constraints (referents) are used to help narrow down possible improvisations and hence avoid excessive computation. Some recognized work on constrained-based improvisation in artificial intelligence focused on this aspect (e.g., [7, 70]).

Several machine learning methods relevant to our computational model were also reviewed in this chapter. For example, detection of novelties in the sensor signals is implemented by temporal difference (TD) learning [170] in which the algorithm attempts to generalize the temporal relationships among sequentially presented data points in order to predict unseen future points. Our computational model is exceptionally relevant to case-based reasoning as it shares a common connotation with improvisation [7]. In particular, continuous CBR [137] is most comparable to our method as both exploit the temporal aspect of sensory information. Another "lazy learning method" that is relevant to our computational approach is instance-based learning. McCallum's work [109] in particular is most relevant. In both McCallum's and our method, the current action is determined by the k closest representations of a current state retrieved from the memory. The difference is that, in McCallum's method, the representation (instances) are retrieved based on the recent action-perception-reward sequence while, in our method, instances are retrieved based on a goal. Our computational model also deals with the partially observable Markov decision process (POMDP) problem of a robot. In other words, it computes the most advantageous action for the current state after assessing what exactly the current state itself is. As is common practice (e.g., [44, 88]), our method employs a Bayesian-based approach to estimate the current state. Furthermore, our method attempts to reduce the infamous computational burden associated with the POMDP calculation in several ways by including state space abstraction, action space abstraction, and trajectory sampling. Finally, by combining history-based and generative-model approaches, our method can also be considered comparable to predictive state representation (PSR) [101] although, in our method, the history is not directly encoded within the representation of a state (event) itself.

CHAPTER 3

A COMPUTATIONAL MODEL OF PROACTIVE INTELLIGENT BEHAVIOR FOR ROBOTS

In this chapter, a computational model of proactive intelligent behavior for robots is developed. *Proactive intelligent behavior* here means that a robot is acting in an either *anticipatory* or *improvisational* manner. More specifically, anticipation is the robot's ability to assess the current state, predict the future consequences of the situation, and execute an action to have a desired outcome based on the determined assessment and prediction. Improvisation, on the other hand, is the ability to promptly detect an unanticipated circumstance of the situation, find a fallback solution to deal with the situation, and execute an action to have a desired outcome. To be self-sustained in any environment, the robot has to be able to compute behavior based solely on what it knows about the world throughout its experience. In other words, the robot has to be able to convert its ongoing experience autonomously into a particular representation that can be effectively utilized during the computation of proactive intelligent behavior.

The computational model was founded upon certain principles found in biology. More specifically, the foundational data structure utilized in this model was inspired by how, according to Eichenbaum and his colleagues [51], an animal's experience is represented in the hippocampus (see Section 2.1.1). In particular, from low-level sensor readings to the abstract notion of episodic memories, specific mathematical expressions were formulated to represent the robot's experience hierarchically within the data structure. Furthermore, this computational model approximates the functionality of Damasio's somatic markers [45] (see Section 2.1.2) by integrating reward signals into this data structure as well.

The details of this biologically inspired foundational data structures are explained in the first section of this chapter. In the second section, the algorithmic processes involved in the computation of proactive intelligent behavior using the foundational data structure are described. The auxiliary functions that do not belong to the core processes but play essential supportive roles in computing proactive intelligent behavior are discussed in the third section. The relevance of our computational model with respect to biology and machine learning is then discussed in the final (fourth) section.

3.1 Foundations

In this section, the foundational data structures that are utilized in the computation of proactive intelligent behavior are explained. In particular, the notions of *events*, *episodes*, and *referents* are explained.

3.1.1 Temporal Abstraction of Ongoing Experience

As discussed in Chapter 1, the primary objective of this dissertation is to determine how extended experience of a robot influences its ability to compute proactive behavior. In particular, the question of *what* information should be extracted from current experience in order to be utilized for future proactive behavior (Subsidiary Question 2) is discussed in this subsection.

Experience

By definition, *experience* is "direct observation of or participation in events as a basis of knowledge" [1]. Correspondingly, in our model, observation (or perception) is considered a fundamental attribute to define experience of robots. Here, a robot's observation is expressed in terms of sensors with which the robot is integrated. Presenting formally, observation (o) is an m-length vector of sensor readings (z) where m is the number of integrated sensors:

$$o = \{z_1, z_2, \ldots, z_m\} \tag{3.1}$$

Another type of firsthand knowledge that encompasses experience is behavior. For example, recall the neurophysiological studies by Eichenbaum and his colleagues [51, 58, 198] indicating that both perceptual and behavioral aspects of an animal's experience are stored using the hippocampus (see Section 2.1.1). Recall also that, according to the cognitive psychological theory of event coding (TEC) [74], jointly storing perceptual and behavioral information of experience in the memory helps an animal attain anticipatory behavior (discussed also in Section 2.1.1). In our model, a robot's behavior is expressed in terms of Arkin's *motor schemata* [10, 11], a distributed motor control method based on Arbib's schema theory [9]. More specifically, each motor schema comprises a tight-loop control program to compute a primitive action given sensor readings. Multiple motor-schemata can be processed simultaneously, and consequently a complex motor behavior can emerge by coordinating their outputs (actions). Instead of being expressed in low-level motor commands (velocity, turning angle, etc.), a motor schema offers a rather abstract notion of actions (***Move-To-Goal***, ***Avoid-Static-Obstacle***, etc.), which helps reduce the action space. In behavior-based robotics, motor-schemata have been successfully implemented in architectural frameworks such as AuRA [14] and RS [104]. Presenting formally, here, overall behavior (b) consists of a set of motor schemata (σ) being instantiated:

$$b = \{\sigma_1, \sigma_2, \ldots, \sigma_\beta\} \tag{3.2}$$

Note that execution of behavior b entails simultaneous (i.e., not sequential) instantiation of specified motor schemata, yielding some overt emergent behavior as a result.

As discussed in Chapter 1, we are also interested in determining whether somatic markers can help a robot achieve better proactive behavior (Subsidiary Question 4). As reviewed in Chapter 2, somatic markers are emotionally induced somatosensory signals saved in episodic memories (along with other types of perceptual signals) and have been shown to help a person determine the expected utility of current action [45]. In our model, a robot is assumed to maintain an independent function that determines whether the current situation is advantageous for the robot itself or not (see Section 3.3.2 below). We refer to the output of such a function as *reward*. Reward approximates an emotionally induced somatosensory signal, serving as a *somatic marker* when saved in memory. Presenting formally, reward (r) is a scalar value indicating to what extent the current state is desirable for the robot. The value can be positive or negative. Having a positive value implies that the robot is currently at some desirable state while at an undesirable state, the reward value becomes negative. When the reward value is zero, the desirability is neutral.

Event

As shown above, in our model, *observation*, *behavior*, and *reward* are considered the attributes to define a robot's experience. More specifically, a stream of the perceptual signals (observation) is constantly monitored by a robot; whenever the characteristic of the signal is found to be distinct from the one received before, a snapshot of all three signals (observation, behavior, and reward) are encapsulated into a joint representation called an *event* (denoted with e) and remembered for future usage:

$$e = \{o, b, r\} \tag{3.3}$$

Each event captures a distinctive configuration of the robot and world, the equivalent of a *state* in a finite state machine.

Perceptual Segmentation

The process of encapsulating a robot's continuous experience into a series of discrete events is here referred to as *event sampling*. As noted above, event sampling is done by segmenting the perceptual signal with respect to its temporal changes (Figure 18). More specifically, the segmentation is performed based on how predictable the perceptual signal is. In other words, given a history, the current value of the robot's observation is constantly predicted; whenever the robot fails to predict the current observation correctly, it is assumed to be entering a new perceptual state, and thus a snapshot of the robot's experience at this particular instance is saved as a new event. Hence, if the environment is simple, since it is highly predictable, the amount of information required to store the state space is minimal. On the other hand, if the environment is complex and highly unpredictable, the amount of information to cover the state space has to be adequately large. In other words, state space abstraction in our method is done based on predictability of the environment.

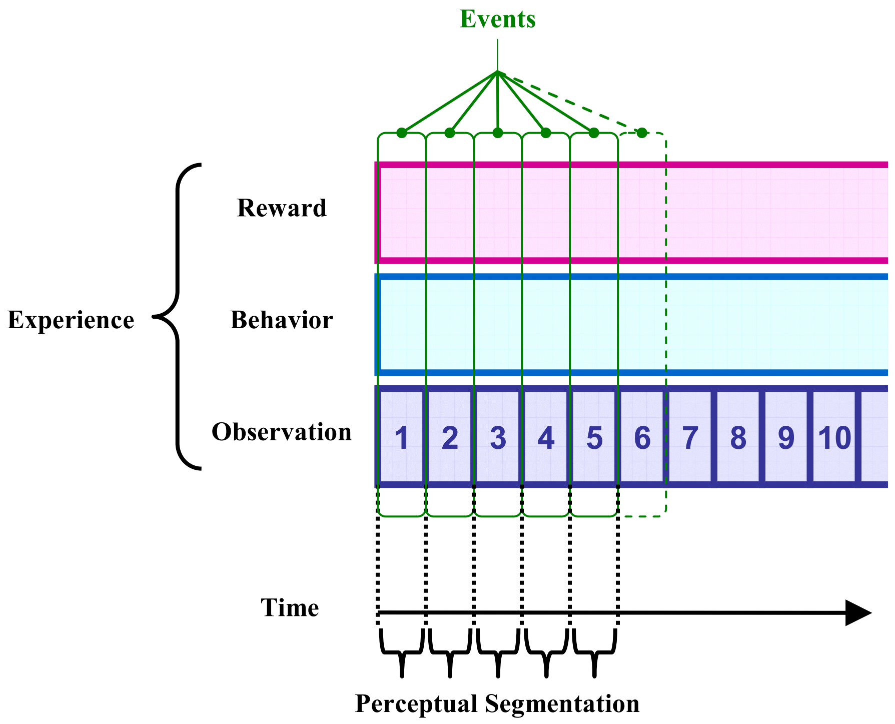


Figure 18: Event sampling via perceptual segmentation of the experience. The robot's continuous experience (observation, behavior, and reward signals) is encapsulated into a series of discrete events based on temporal changes in the reward signal.

To predict the current observation, the current reading of every integrated sensor (*z*) is estimated by a designated predictive function; the predictive function is modeled by a straightforward linear equation as shown in Equation 3.4[5]:

$$z'_t = w_t \, z_{t-1} \tag{3.4}$$

where z'_t is the predicted sensor reading at instance t (current), z_{t-1} is the actual sensor reading at the previous instance, w is a current weight being updated by some simple reinforcement learner. Here, the reinforcement learner can be implemented with, for example, TD(λ) [170] (Section 2.4.1), in which the update rule shown in Equation 3.5 is applied to adjust the weight:

$$w_{t+1} = w_t + \alpha(z_t - z'_t)\sum_{k=1}^{t} \lambda^{t-k} \nabla z'_k \tag{3.5}$$

where α is a learning rate, λ^k is an exponential weighting factor[6], and the gradient $\nabla z'_k$ is a partial derivative of z'_k with respect to the weight[7]. At each time cycle, a root-mean-squared (RMS) difference of predicted and actual sensor readings is calculated. When the RMS difference is the greatest (local maximum) among neighboring instances, a new event is sampled:

$$f_{\text{sample}}(t) = \begin{cases} \text{true} & \text{if } \dfrac{df_{\text{rms}}(o'_t - o_t)}{dt} = 0 \text{ and } \dfrac{d^2 f_{\text{rms}}(o'_t - o_t)}{dt^2} < 0 \\ \text{false} & \text{otherwise} \end{cases} \tag{3.6}$$

where f_{rms} is a function that returns a RMS of a vector.

---

5 The subscript (*t*) denotes a temporal index, different from the subscript used to denote the integrated sensor types in Equation 3.1.

6 Sutton and Barto [171] refers to this factor as an *eligibility trace*.

7 Because of Equation 3.4, $\nabla z'_k$ is simply z_{k-1}.

A simple example of the event sampling process is demonstrated in Box 1. Furthermore, Figure 20 shows the prediction errors of observations when a simulated robot (integrated with 10 sonar sensors) proceeds along a corridor in an office building. Each spike represents the occurrence of an event, and it shows how events are clustered around salient features of the environment such as open doors and a corridor junction.

Box 1: A simple example of event sampling.

Figure 19 below illustrates a simple environment with a concave wall. A mobile robot, equipped with a one-dimensional range sensor facing the wall, moves from Position 0 to Position 8, measuring the distance to the wall nine times. Table 1 shows the measurements (z) at the nine instances, the predicted measurements (z') attained via Equation 3.4, the root-mean-squared differences (error) between z and z', and the outputs of the sampling function (Equation 3.6).

In this case, the events are sampled at instances 3 and 6 as the characteristic of the sensor signal significantly changes at those instances.

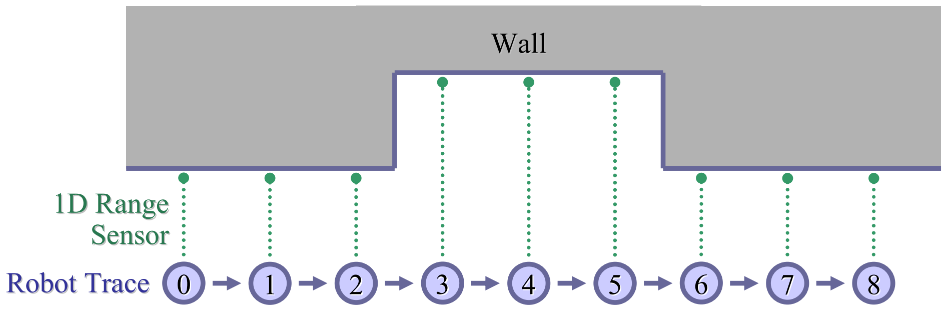


Figure 19: A trace of a robot measuring the distance to a concave wall.

Table 1: The measurements (z) at the nine instances, the predicted measurements (z'), the error of z', and the outputs of the sampling function.

t	0	1	2	3	4	5	6	7	8
z	1.0	1.0	1.0	2.0	2.0	2.0	1.0	1.0	1.0
z'	1.0	1.0	1.0	1.0	2.0	2.0	2.0	1.0	1.0
error	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
$f_{sample}(t)$	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE

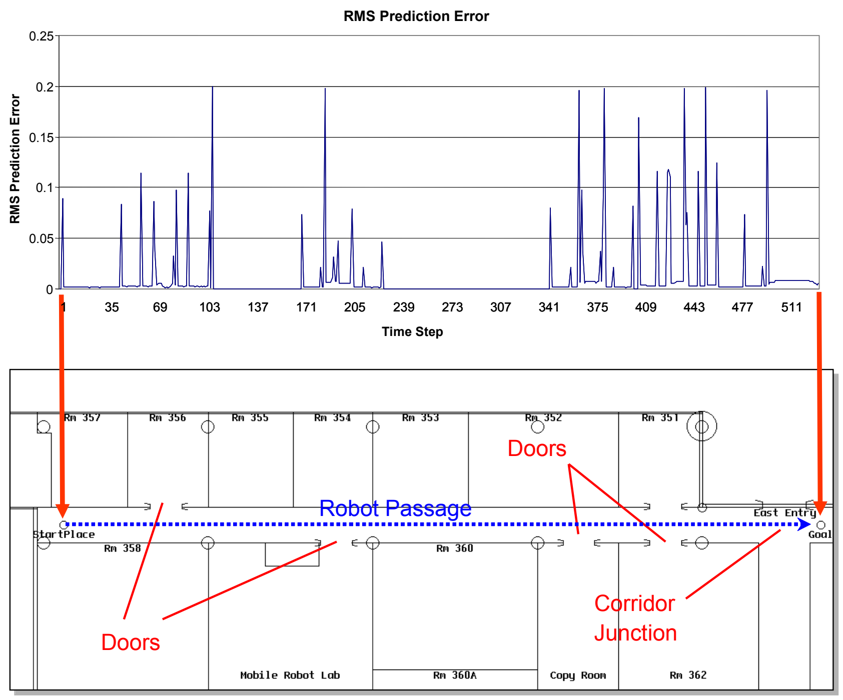


Figure 20: A graph of prediction errors and a corresponding robot path in a simulated indoor environment. The robot has 10 sonar sensors onboard. Each spike in the graph indicates the occurrence of a new event.

3.1.2 Formation of Episodic Memories

Like a single frame in a motion picture film, an event itself is a mere snapshot of the world at some particular instance. However, when presented collectively in a sequence, the information becomes vivid and provides a robot with a foundation for computing proactive intelligent behavior. Here, contextually similar events in a sequence are grouped together, forming another computational element called an *episode* (or *episodic memory* when collectively saved in the robot's physical memory). Presenting formally, an episode (E) holds an n-length ordered vector of events along with its contextual information (χ):

$$E = \{(e_1, e_2, \ldots, e_n), \chi\} \qquad (3.7)$$

The notion of *context* (χ) is explained in detail below.

Purposive Contextualization

The method of forming episodes based on goals is here referred to as *purposive contextualization*. A goal is a desired perceptual state that the robot is attempting to reach. For example, suppose that the robot is equipped with a gripper and tactile sensors. If the robot intends to grab a ball, a set of expected tactile sensor readings, indicating a ball being grabbed, becomes a goal. Recall that an *observation* (o) is defined in terms of readings of integrated sensors (Equation 3.1). Likewise, *goal* (g) is defined with an m-length vector of sensor readings where m is the number of the integrated sensors:

$$g = \{z_1, z_2 \ldots, z_m\} \tag{3.8}$$

Different goals can be activated (or deactivated) depending on the robot's motivational state (Section 3.3.1). When activated, the events that were sampled during the pursuit of a goal are grouped together, forming a new episode (Figure 21). In this case, the episode in Equation 3.7 is alternatively represented as:

$$E_P = \{(e_1, e_2, ..., e_n), g\} \tag{3.9}$$

Note that deactivation of the goal results in termination of this particular episode. However, deactivation of a goal does not necessary imply that the goal is actually met by the robot. As mentioned above, activation and deactivation of a goal are determined by a motivation function whose outcome reflects the robot's motivational state (see Section 3.3.1). Hence, even if the goal is not met, the episode can be terminated.

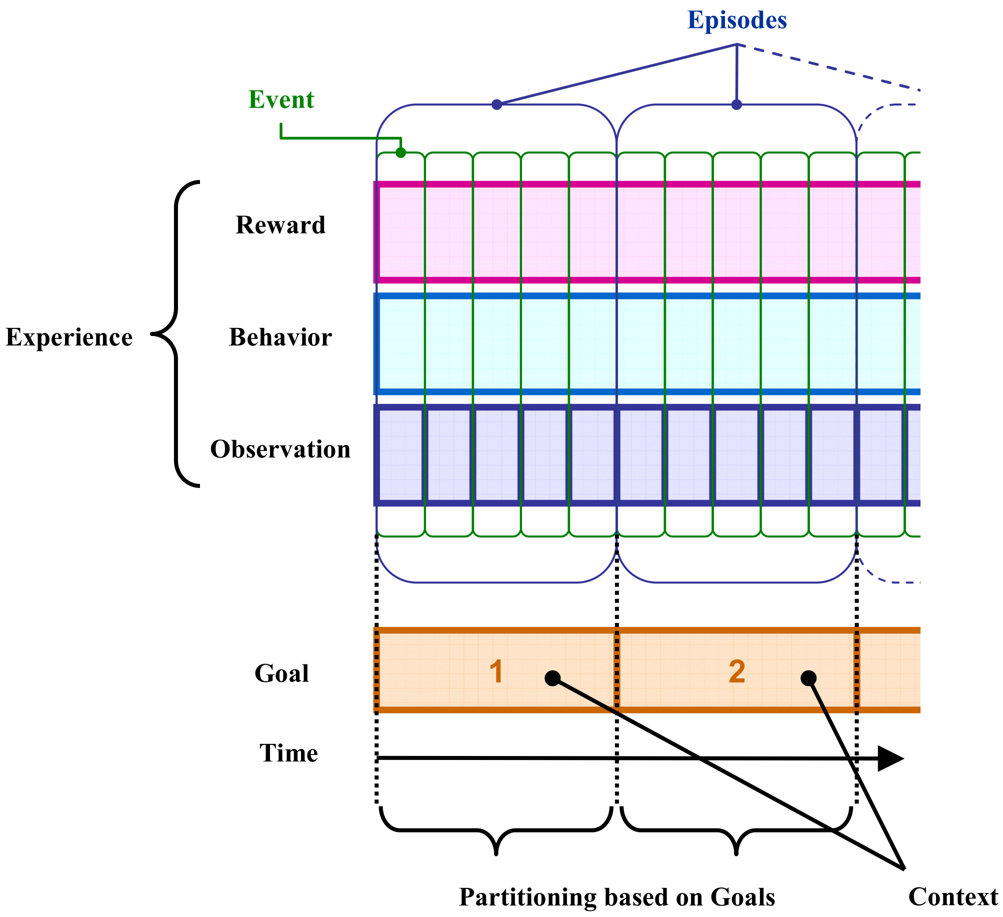


Figure 21: Production of episodes via purposive contextualization. Events are divided into different episodes based on changes of the goals that the robot pursues.

<u>Utilitarian Contextualization</u>

The robot, however, is not required to have a specific perceptual goal all the time. For example, it may wander around to explore the environment without a particular desired perceptual state in mind. In this case episodes are not formed based on goals, but instead by changes in the reward signal that guides the partitioning of the events (*utilitarian contextualization*) (Figure 22).

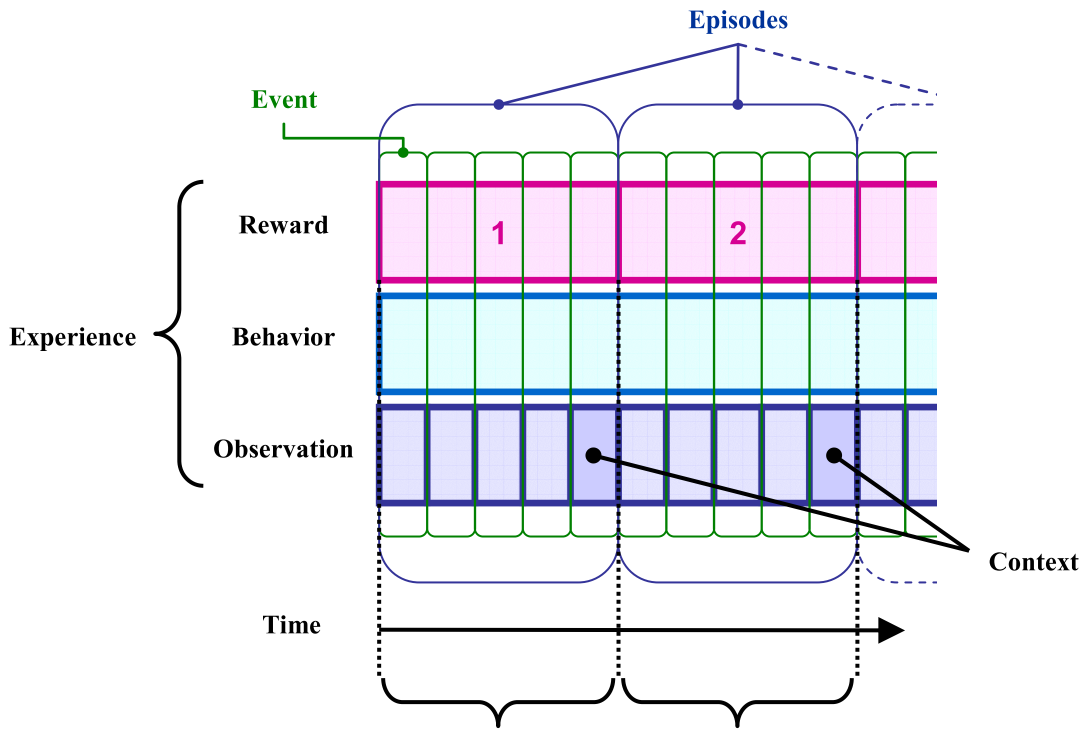


Figure 22: Production of episodes via utilitarian contextualization. Events are divided into different episodes based on how significantly the characteristics of the reward signal changes.

More specifically, when a goal is not pursued, the value of the reward signal is closely monitored; when the signal is having some momentous value, the current environment and the experience leading up to that point is considered important for the robot, and thus the events occurred during that period are grouped together as a new episode. Here, the reward signal is considered having a momentous value when the characteristic of the reward signal is changed significantly. For example, a first-order differential equation can be used to identify mathematically critical points (i.e., local maximum and minimum) in the signal:

$$f_{\text{partition}}(t) = \begin{cases} \text{true} & \text{if } \dfrac{dr_t}{dt} = 0 \\ \text{false} & \text{otherwise} \end{cases} \tag{3.10}$$

In other words, if the derivative of the reward signal becomes zero, $f_{\text{partition}}$ returns true, the events until then are partitioned into a new episode. Note that $f_{\text{partition}}$ can be implemented using other appropriate functions as long as the characteristic of the reward signal can be identified.

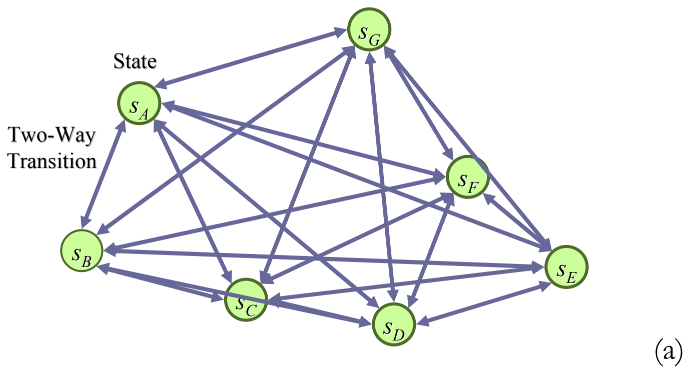
In the utilitarian contextualization, the observation stored in the end-event when the reward signal had a momentous value is set as the context of the episode. Hence, the episode in Equation 3.7 can be expressed as:

$$E_U = \{(e_1, e_2, ..., e_n), o_{[e_n]}\} \tag{3.11}$$

where $o_{[e_n]}$ is the observation of the end-event.

Unidirectional Temporal Linear Chain

It should be noted that, in terms of machine learning, an episode can be considered the equivalent of a *state space* (as an event being the equivalents of a state). However, in an episode, events are organized in a unidirectional temporal linear-chain fashion where the state (event) transitions are guided by the specific episode trajectory (i.e., the exact order of the event sequence recorded in the episode) (Figure 23). This property in fact benefits the computation of proactive behavior in two ways. First, it allows us to assume that the transition probability between any two events can be approximated by a discrete probability distribution (Poisson) based on the event distance between them. As explained in detail in Section 3.2.2 and verified in Section 5.1, because of this property, the computational time required to estimate the current state can be reduced from $O(n^2)$ to $O(n)$. The second advantage is that, as explained in Section 3.2.3 below, when assessing the utility of each state, a state transition between two states that do not belong to a common episode trajectory can be ignored, reducing the computational steps required for value iteration. In other words, *trajectory sampling* [171] can be applied to assess the utility values.



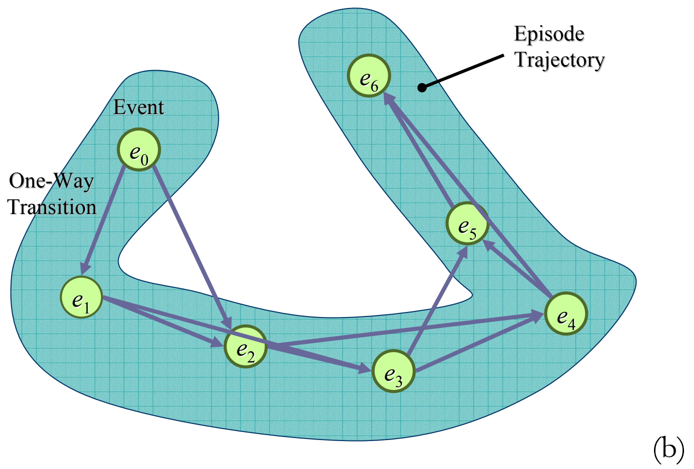


Figure 23: (a) A general state machine with fully connected transitions. (b) Formation of an episode with a unidirectional linear chain of events (from e_0 to e_6).

3.1.3 Construction of Referents

As described above, *events* and *episodes* are computational elements that carry both precise and abstract levels of information, respectively. Here, *referents* offer a middle ground between the two to provide flexibility of solutions in computing proactive intelligent behavior. In particular, referents are utilized in computing improvisational behavior (Section 3.2.5). The concept of referents was adopted from Jazz improvisation [133] (Section 2.3.2).

A referent outlines the behavioral progression in an episode. Each referent is constructed from a single episode. A referent consists of a sequence of nodes, and each node has a unique association with a particular behavior type instantiated in that episode. Along with the behavior type, a referent node also retains the notions of what the robot was observing before and after the behavior was executed. Here, the observation perceived before executing the behavior is referred to as a *nodal precondition*, and the perceptual state perceived after the execution is referred to as a *nodal effect*. Expressed formally, a referent node (ω) consists of a behavior (b), a nodal precondition (o_{init}), and a nodal effect (o_{end}):

$$\omega = \{b, o_{\text{init}}, o_{\text{end}}\} \tag{3.12}$$

The representation of b is exactly same as the one defined in Equation 3.2 (i.e., a set of active motor schemata), and both o_{init} and o_{end} share the exactly same representation with observation (o) defined in Equation 3.1 (i.e., a set of sensor readings).

A sequence of nodes collected from a single episode composes a referent (Ω), which can be formally expressed as:

$$\Omega = (\omega_1, \omega_2, ..., \omega_R) \tag{3.13}$$

where R is the number of referent nodes. Note that the number of referent nodes is same as the number of times the behavior instantiation has been altered in one episode (Figure 24). An example of how a referent can be extracted from an episode is shown in Box 2.

Note that the framework of referents is set up in a way that some classical planning algorithm (e.g., STRIPS planning [57]) may be employed to plan a sequence of actions based on the referent nodes. In this case, the behavior in a referent node is an action (operator). After associating the numerically described goal (Equation 3.8) and nodal preconditions/effects with some abstract symbols, an agent should be able to generate a sequence of actions that leads a robot to reach a goal state from a current state. However, such symbolic planning is beyond the scope of this dissertation. Our use of referents in the context of improvisation is described in Section 3.2.5.

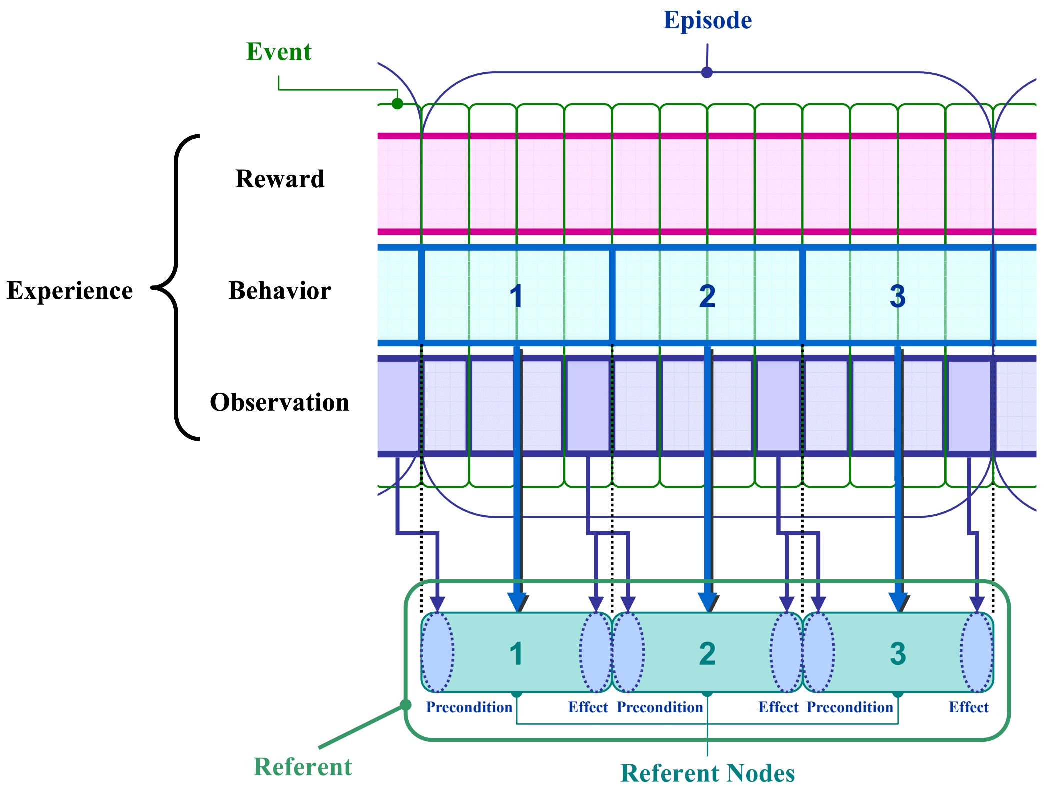


Figure 24: Construction of a referent from an episode based on behavioral instantiations. In this case, three referent nodes are extracted since there were three behavioral instantiations during this episode. The observation perceived before Behavior 1 is stored as the nodal precondition of Node 1. The observation at the end of Behavior 1 is the nodal effect of Node 1 as well as the nodal effect of Node 2. Similarly, the observation at the end of Behavior 2 is the nodal effect of Node 2 as well as the nodal effect of Node 3. Finally, the observation at the end of Behavior 3 is the nodal effect of Node 3. (See Box 2 for an example with concrete numbers.)

Box 2: An example of constructing a referent.

Suppose that an episode (Episode 1) consists of 10 events as shown in the table (Table 2) below. Since its behavior type (b) was altered twice (at events e_3 and e_7), three referent nodes (ω_1, ω_2, and ω_3) are extracted (similar to the diagram in Figure 24 above). The *nodal precondition* of ω_1 is the observation perceived before b_{MF} is executed (i.e., $z = 1.7$), and its *nodal effect* is the observation of e_2 (i.e., $z = 4.7$). The observation of e_2 is also the nodal precondition of ω_2. Similarly, the observation of e_6 (i.e., $z = 0.9$) is the nodal effect of ω_2 as well as the nodal precondition of ω_3. The nodal of effect of ω_3 is the observation of e_9 ($z = 5.8$).

Table 2: The perceptual information (*z*), behavioral information (*b*), and referent nodes (*ω*) of the events in Episode 1.

Episode 1	—	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
z	1.7	9.1	5.8	4.7	5.7	5.0	7.1	0.9	4.0	1.9	5.8
b	—	b_{MF}	b_{MF}	b_{MF}	b_{SO}	b_{SO}	b_{SO}	b_{SO}	b_{MF}	b_{MF}	b_{MF}
ω	—	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_3	ω_3	ω_3

3.2 Computation of Proactive Intelligent Behavior

In this section, the algorithmic processes involved in computation of proactive intelligent behavior are explained. The overall progression of the computation is illustrated in Figure 25. Recollection, event matching, behavior selection, validation, and recovery are performed during the computation. The following subsections describe these processes in detail.

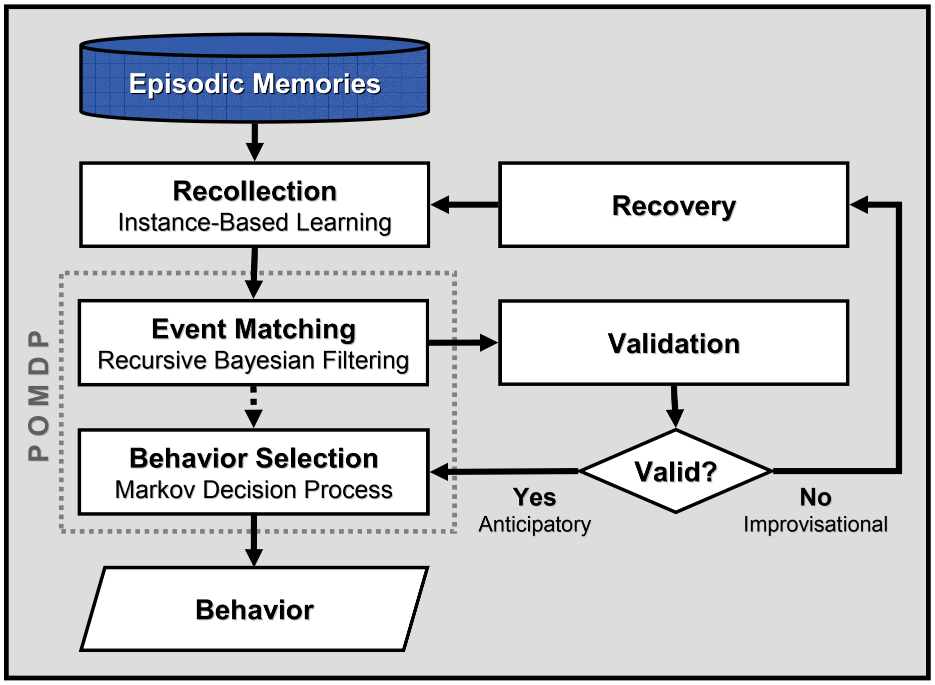


Figure 25: A flow chart of the proactive behavior computation.

3.2.1 Recollection

Recollection is a process to recall relevant episodes from memory in order to deal with the environment in which the robot currently is located. The recalled episodes offer a basic framework on which the computation of proactive behavior can be based. Here, the relevance of an episode is calculated with respect to a goal that the robot pursues. In terms of machine learning, recollection can be considered as the equivalent of defining an appropriate state space for a given goal. However, it should be noted that while a standard machine learning problem defines only a single state space to work with, in this episodic-memory-based computation, multiple state spaces (episodes) can be specified for a sole problem (goal). On the other hand, if the robot pursues no goal, the recollection process yields no relevant episode (i.e., the state space is empty); hence, proactive behavior cannot be computed in this case.

The core algorithms involved in the recollection process are essentially the same as those found in instance-based learning (Section 2.4.3), consisting of similarity and classification functions [6]. The details of the similarity and classification functions used in this recollection process are explained below.

Similarity Function

Recall that an episode consists of a sequence of events as well as the context (χ) (Equation 3.7). The relevance of the episode is measured in terms of the similarity between the context and goal, which are both expressed in the form of a set of sensor readings (see Equations 3.9, 3.11, and 3.8). As explained in Section 3.1.2 above, depending on how the episode was formed, the context can be either a past goal that the robot was pursuing (purposive contextualization) or a set of particular perceptual readings at the climax of the episode (utilitarian contextualization). The similarity value (ρ_E) in both cases is computed probabilistically as shown in Equation 3.14:

$$\rho_E = f_{\mathrm{L}}(g_{\mathrm{cur}}, \chi_{[E]}) \tag{3.14}$$

where the function returns the likelihood of the goal (g_{cur}), sampled from a certain probability distribution[8] where the context of the episode ($\chi_{[E]}$) is the mean. An example of computing similarity values for a simple case is shown in Box 3. Note that, as explained in Section 3.2.5 below, an intermediate goal attained via the *recovery* process can also be evaluated with this function when the robot is computing improvisational behavior.

[8] Here, we assume the normal distribution, a common distribution used to model a sensor with noise (e.g., Kalman filter [82]).

Box 3: An example of computing similarity values.

Suppose that the current goal (g_{cur}) is to touch a red object, which is expressed as:

$$g_{\text{cur}} = \{z_{\text{Red}}, z_{\text{Green}}, z_{\text{Blue}}, z_{\text{Dist}}\} = \{1.0, 0.0, 0.0, 0.0\}$$

The first three elements (normalized) are the red, green, and blue values of the object color, respectively. The fourth element is the distance to the object. Suppose also that there are two types of episodic context, χ_1 is χ_2, representing the robot's perceptual states in which the robot touches a pink object and a green object, respectively:

$$\chi_1 = \{1.0, 0.7, 0.8, 0.0\} \text{ and } \chi_2 = \{0.0, 1.0, 0.0, 0.0\}$$

Having χ as its mean, the Gaussian probability density function is expressed as:

$$f_{\text{Gauss}}(X)_\chi = \frac{1}{(2\pi)^{N/2}\sqrt{|\Sigma|}} \exp\left(\frac{(X-\chi)^{\text{T}}\Sigma^{-1}(X-\chi)}{-2}\right)$$

where N is the dimension of the vector (which is 4), and Σ is a covariance matrix. For the sake of this example, we assume that Σ is the identity matrix (i.e., the sensors are uncorrelated, and the variance is 1.0); the determinant of Σ is thus 1.0.

The similarity values are computed by inserting g_{cur} into the Gaussian function as:

$$\rho_E = f_{\text{L}}(g_{\text{cur}}, \chi) = f_{\text{Gauss}}(g_{\text{cur}})_\chi = \frac{1}{(2\pi)^2} \exp\left(\frac{(g_{\text{cur}}-\chi)^{\text{T}}(g_{\text{cur}}-\chi)}{-2}\right)$$

Therefore, the similarity values of χ_1 and χ_2 with respect to g_{cur} are computed as:

$$\rho_{E1} = \frac{1}{(2\pi)^2} \exp\left(\frac{(1.0-1.0)^2 + (0.0-0.7)^2 + (0.0-0.8)^2 + (0.0-0.0)^2}{-2}\right) = 0.0144$$

$$\rho_{E2} = \frac{1}{(2\pi)^2} \exp\left(\frac{(1.0-0.0)^2 + (0.0-1.0)^2 + (0.0-0.0)^2 + (0.0-0.0)^2}{-2}\right) = 0.0093$$

Since it has a larger similarity value, in this case, χ_1 is more similar to g_{cur} than χ_2.

Classification Function

Once the similarity value is determined, the next step in recollection is to classify the episode as to whether it is relevant or irrelevant to the current goal. The episode is classified as *relevant* if the similarity value is found above a predefined threshold. Having a very high threshold means that the only episodes whose contexts are identical to the current goal are considered relevant. On the other hand, if the threshold is set to low, the episodic contexts that are not substantially similar to the current goal can be considered relevant. Note that while a tentative value was set for the threshold in our implementation (Chapter 4), this value can be eventually learned by some reinforcement method (e.g., simulated annealing [85]) although it is beyond the scope of this dissertation.

Presenting formally, for any episode that is in the robot's memory (C), if the similarity value is above the threshold (θ_ρ), the episode is classified as *relevant* and added to the working memory (M_{rel}):

$$M_{\text{rel}} = \{E_{1:K} \mid \{E_{1:K}\} \subseteq C \wedge \rho_{E_{1:K}} \geq \theta_\rho\} \tag{3.15}$$

where K is an upper limit value posed to restrict the size of M_{rel}. Note that a high K value is certainly always desirable since a statistically significant number of episodes can be collected to make an informed decision for a specific goal. However, as discussed in Section 5.3, having a higher K value requires greater computational power. A similar notion of restricting the size of the working memory has been suggested by Kira and Arkin [84] in the context of case-based reasoning (see Section 2.4.2). In order to reduce the size of a case library, four types of strategies were proposed to eliminate cases from the library: namely, 1) random elimination, 2) performance-based elimination (i.e., delete poorly performed cases), 3) recency-based elimination (i.e., delete old cases), and 3) frequency-based elimination (i.e., delete infrequently used cases). According to their experiment in simulation, the performance-based elimination strategy and a combination of the three non-random strategies were found most useful for robot navigation. However, the difference from our method is that their *case* consists of behavior parameters while ours consists of an episode that is a sequence of perceptions/behaviors/rewards. As we will discuss in Section 3.2.3, an episode with poor performance should not be eliminated because such an episode can contribute to discount the utility of executing a certain undesirable behavior. Since we assume that the environment does not stay static, in our case, we apply the recency-based elimination strategy to restrict the size of the working memory. In other words, the newer a recalled episode the higher its priority is; episodes that are not in the first K newest episodes are thus eliminated from the selection. We refer to this process as *history-length trimming*, and its effectiveness is evaluated in the third experiment (Section 5.3). Box 4 shows an example of how to compute the relevance of episodes.

Box 4: An example of computing relevance of episodes.

Suppose that there are nine episodes in the robot's memory whose similarity values (ρ_E) are specified as shown in Table 3 below. The episodes in the table are ordered chronologically; E_1 is the earliest episode, and E_9 is the latest one. Here, we suppose that the threshold (θ_ρ) is set to 0.01. If the upper bound is not set (i.e., $K = \infty$), six episodes (E_1, E_3, E_4 E_5, E_6, and E_8) will be classified as *relevant* as they exceed the threshold. If however K is set to 3, the only three latest episodes from the six (E_5, E_6, and E_8) will be classified as *relevant*.

Table 3: The similarity values (ρ_E) and relevance of sample episodes.

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
ρ_E	0.014	0.009	0.012	0.011	0.010	0.013	0.008	0.015	0.006
$\rho_E \geq \theta_\rho, K = \infty$	✓ Relevant		✓ Relevant	✓ Relevant	✓ Relevant	✓ Relevant		✓ Relevant	
$\rho_E \geq \theta_\rho, K = 3$					✓ Relevant	✓ Relevant		✓ Relevant	

3.2.2 Event Matching

If recollection is the process to define a state space for solving the current goal, event matching is the equivalent of state estimation. More specifically, the function of event matching is to identify the event that best represents (matches) the current situation from every relevant episode collected by the recollection process. Here, event matching is accomplished by a recursive Bayesian filter, the probabilistic method commonly used for solving the simultaneous localization and mapping (SLAM) problem [178]. At first, for each relevant episode, the posterior probabilities (belief) of being at some event (e_q) in the episode given the history of the observations (o^{τ}) and executed behaviors (b^{τ}) are solved by the following recursive equation[9]:

$$p(e_q|o^{\tau},b^{\tau}) = \eta\, p(o_{\tau}|e_q) \sum_{e_{\tau-1}\in E} p(e_q|b_{\tau},e_{\tau-1})\, p(e_{\tau-1}|o^{\tau-1},b^{\tau-1}) \tag{3.16}$$

where η is a normalization factor, $p(o_{\tau} \mid e_q)$ is the sensor model, $p(e_q|b_{\tau},e_{\tau-1})$ is the transition model, and $p(e_{\tau-1}|o^{\tau-1},b^{\tau-1})$ is the belief of the previous computational cycle. An example of the posterior probability computation is shown in Box 5.

To implement the sensor model, which is the conditional probability of observing o_{τ} given the query event (e_q), the same similarity function used in the recollection process (Equation 3.14) can be employed:

$$p(o_{\tau} \mid e_q) = f_{\mathrm{L}}(o_{\tau}, o_{[e_q]}) \tag{3.17}$$

where o_{τ} is the current observation, and $o_{[e_q]}$ is the observation saved in the querying event.

[9] See Appendix A for derivation.

Box 5: An example of the posterior probability computation.

Suppose that a robot experienced an episode, Episode 1 (Table 4), in which it sampled five events while moving forward and measuring the distance to a concave wall as shown in the figure. Table 5 shows the current sequence of the robot's perceptual and behavioral information ($\tau - 3$ is the earliest, and τ is the latest). As shown in Table 6, the posterior probabilities (Equation 3.16) at instances $\tau - 3$, $\tau - 2$, $\tau - 1$, and τ have their highest values at e_0, e_1, e_2, and e_3, respectively. Note that $p(e_3|o^{\tau}, b^{\tau})$ at $\tau - 2$ also has a relatively high value (0.14) since e_1 and e_3 are similar in terms of their sensor readings (2.00).

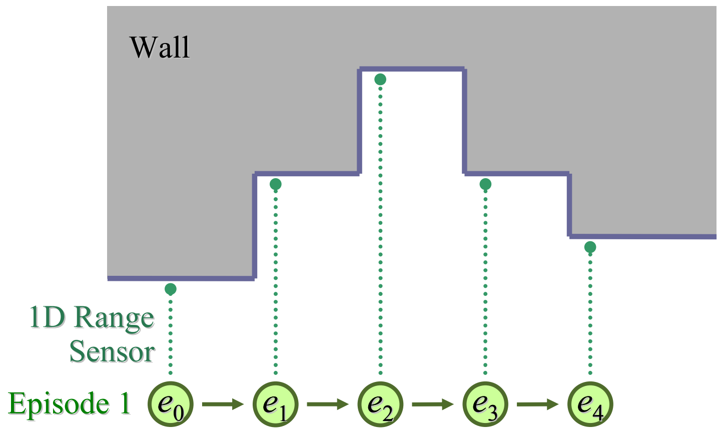


Table 4: The perceptual (*z*) and behavioral (*b*) information of Episode 1.

Episode 1	e_0	e_1	e_2	e_3	e_4
z	1.00	2.00	3.00	2.00	1.50
b	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*

Table 5: The perceptual (*z*) and behavioral (*b*) information of the current sequence.

Current Sequence	τ-3	τ-2	τ-1	τ
z	0.99	2.01	3.05	2.04
b	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*

Table 6: The posterior probabilities of the current sequence.

Posterior Probability	$p(e_0 \mid o^{\tau}, b^{\tau})$	$p(e_1 \mid o^{\tau}, b^{\tau})$	$p(e_2 \mid o^{\tau}, b^{\tau})$	$p(e_3 \mid o^{\tau}, b^{\tau})$	$p(e_4 \mid o^{\tau}, b^{\tau})$
τ - 3	0.99	0.00	0.00	0.00	0.01
τ - 2	0.00	0.86	0.00	0.14	0.00
τ - 1	0.00	0.00	1.00	0.00	0.00
τ	0.00	0.00	0.00	1.00	0.00

On the other hand, the transition model is the transition probability of the robot arriving at the target event (e_q) when the previous event is $e_{\tau-1}$ and behavior b_τ is currently being executed. In the certainty equivalence approach [94, 189], the transition probabilities may be estimated by taking the statistic of the transitions while exploring the environment [81, 171]. For example, a robot may randomly explore the environment and keep track how many transitions from one state to another have occurred. On the other hand, in this episodic-memory-based approach, since events are formed in a unidirectional temporal linear chain (Equation 3.7), the transition model can be computed in terms of how many events the robot has to advance from $e_{\tau-1}$ in order to reach e_q. Let e_j be $e_{\tau-1}$, the transition model can be formally represented as:

$$p(e_q \mid b_\tau, e_j) = \begin{cases} f_{\mathrm{P}}(e_j, e_q) + \varepsilon_m & \text{if } q > j \text{ and } b_q = b_\tau \\ \kappa_m\, f_{\mathrm{P}}(e_j, e_q) + \varepsilon_m & \text{else if } q > j \text{ and } b_q \neq b_\tau \\ \varepsilon_m & \text{otherwise} \end{cases} \tag{3.18}$$

where ε_m is some extremely small number to ensure that the probability does not become absolutely zero, κ_m is a discount factor, and f_{P} is a function that returns the probability of the robot reaching e_q from e_j. Here, f_{P} assumes a discrete probability distribution, namely, the Poisson distribution. More specifically, let us define $d_{j:q}$ to denote the distance between e_j and e_q in terms of event numbers, and $\bar{d}$ to denote the average number of events that the robot advances within one computational cycle. The probability of the robot reaching e_q from e_j is then computed by the following equation:

$$f_{\mathrm{P}}(e_j, e_q) = \mathrm{Poisson}(d_{j:q}, \bar{d}) = \frac{\exp(-\bar{d})\, \bar{d}^{(d_{j:q})}}{d_{j:q}!} \tag{3.19}$$

In other words, the output of the transition model (Equation 3.18) is the probability computed by f_{P} if the index of e_q is greater than the index of $e_{\tau-1}$, and also if b_τ is the same behavior that is stored in e_q; if the behaviors mismatch, the probability is discounted. Ideally, the amount of the discount should be analytically assigned based on the characteristics of the behaviors. For example, if b_q and b_τ makes a robot move in opposite directions, the discount should be greater than when moving in the same direction. Currently, however, the discount factor is assumed a constant. The analytical assignment of the value should be addressed in the future. An example of this transition model computation is shown in Box 6.

Box 6: An example of the transition model computation.

Suppose a previously stored episode (Episode 1) consists of five events whose behavioral information is shown in Table 7 below. Suppose also that the robot is currently instantiating the *Move-Forward* motor-schema (i.e., b_τ = *Move-Forward*). Table 8 shows four cases of the transition model computation in which the transition probability of arriving at an event (e_q) in Episode 1 from another event (e_j) in Episode 1 is calculated by Equation 3.18. In Case 1, since q comes after j, and b_q matches with b_τ, the transition probability is simply the output of the Poisson function (Equation 3.19) plus some ε_m. In Case 2, since b_q and b_τ mismatch, the output is multiplied by a predefined discount factor. In Cases 3 and 4, since q comes earlier than j, the transition probability is just ε_m.

Table 7: The behavioral information of Episode 1.

Episode 1	e_0	e_1	e_2	e_3
b	*Move-Forward*	*Move-Forward*	*Swirl-Obstacle*	*Swirl-Obstacle*

Table 8: Four cases of transition model computation.

	Case 1	Case 2	Case 3	Case 4
	$e_q = e_1$ $e_j = e_0$	$e_q = e_2$ $e_j = e_1$	$e_q = e_1$ $e_j = e_2$	$e_q = e_2$ $e_j = e_3$
$q > j$	TRUE	TRUE	FALSE	FALSE
$b_q = b_\tau$	TRUE	FALSE	TRUE	FALSE
$f_P(e_j, e_q)$	0.368	0.368	0.000	0.000
$p(e_q \mid b_\tau, e_j)$	$0.368 + \varepsilon_m$	$0.368\kappa_m + \varepsilon_m$	ε_m	ε_m

Since the posterior probabilities are computed whenever the event sampling captures a new event, the value of $\overline{d}$ is assumed to be 1.0. The graph of the Poisson probability mass function when $\overline{d}$ is 1.0 is shown in Figure 26. Since $\overline{d}$ is always assumed to be 1.0, the transition probability is always chosen from this distribution. Consequently, as indicated in the figure, the probability becomes near-zero (0.003) when the distance from e_j to e_q becomes five. This is a property of this function that can be in fact exploited to reduce the computational burden of event matching for each episode from $O(n^2)$ to $O(n)$ by computing the transition model in Equation 3.16 only for a 5-event distance (instead of all n events).

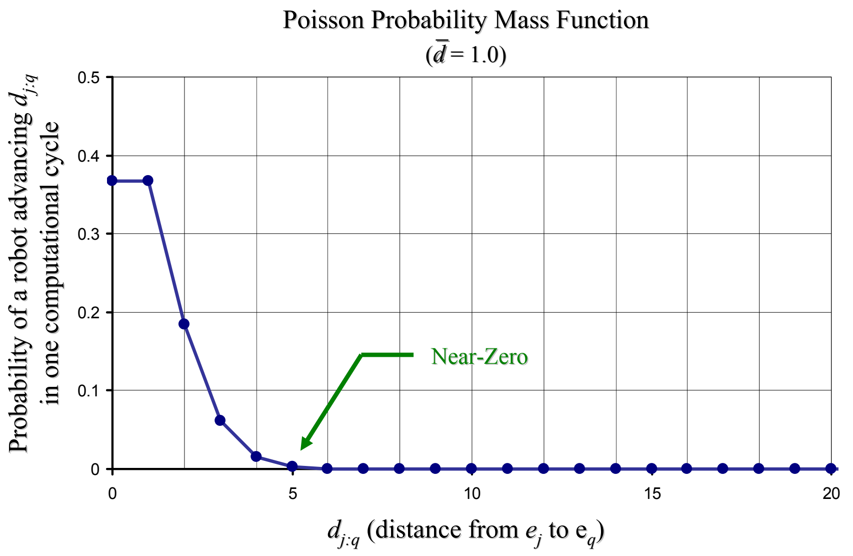


Figure 26: The probability mass function for the Poisson distribution.

After the posterior probabilities for all of the events in the episode are computed, the one with the highest probability is considered as the event that best represents the current state. However, it is possible that the state space was not appropriately chosen by the recollection process. In other words, none of the events could correspond to the current state. In that case, similar to the approach suggested by Tomatis et al. [183], the entropy of the posterior probability distribution is checked. Here, the entropy (H) of the posterior probability distribution for an episode (E) is computed by Shannon's information entropy equation

[157]:

$$H(E) = -\sum_{e_i \in E} p(e_i \mid o^{\tau}, b^{\tau}) \log_2 p(e_i \mid o^{\tau}, b^{\tau}) \tag{3.20}$$

Having a high entropy value infers that the probability distribution is close to uniform. Thus, only if the entropy is below the predefined threshold (θ_H), the event with the highest posterior probability in the episode is considered *matched* ($\hat{e}_{[E]}$) to the current event (state):

$$\hat{e}_{[E]} = \{e \mid e = \operatorname*{argmax}_{e \in E} p(e|o^{\tau}, b^{\tau}) \wedge H(E) \le \theta_H\} \tag{3.21}$$

Note that the value of θ_H is currently determined empirically although a more sophisticated method to determine the value should be addressed in future work. For example, in our indoor experiment using a real robot (Section 5.2), θ_H was set to 2.5 (Section C.1). An example of this event-matching process is shown in Box 7.

It should be also noted that, if the recollection yields k episodes as *relevant* (Equation 3.15), there will be at most k events that could be legitimately matched. A set of all relevant episodes whose events are legitimately matched is denoted with $\hat{M}_{\text{rel}}$:

$$\hat{M}_{\text{rel}} = \{\forall E \mid E \in M_{\text{rel}} \wedge H(E) \le \theta_H\} \tag{3.22}$$

The example shown in Box 8 demonstrates this equation in use.

Box 7: An example of event matching.

Suppose that a robot experienced two episodes, Episode A (Table 9) and Episode B (Table 10). The current sequence of the robot's perceptual and behavioral information is shown in Table 11. As shown in Table 12, at the current instant (τ), the highest posterior probabilities (Equation 3.16) for Episodes A and B can be found at e_{A3} and e_{B3}, respectively. Suppose that the threshold (θ_H) is set to 2.0. As shown in Table 12, in this case, e_{A3} is the only event that is considered *matched* since the entropy (H) of Episode A does not exceed the threshold.

Table 9: The perceptual (*z*) and behavioral (*b*) information in Episode A.

Episode A	e_{A0}	e_{A1}	e_{A2}	e_{A3}	e_{A4}
z	1.00	2.00	3.00	2.00	1.50
b	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*

Table 10: The perceptual (*z*) and behavioral (*b*) information in Episode B.

Episode B	e_{B0}	e_{B1}	e_{B2}	e_{B3}	e_{B4}
z	1.95	2.00	2.05	2.10	2.30
b	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*

Table 11: The perceptual (*z*) and behavioral (*b*) information of the current sequence.

Current Sequence	τ-3	τ-2	τ-1	τ
z	0.99	2.01	3.05	2.04
b	*Move-Forward*	*Move-Forward*	*Move-Forward*	*Move-Forward*

Table 12: Event matching of the two episodes and their entropy values.

	$p(e_0 \mid o^\tau, b^\tau)$	$p(e_1 \mid o^\tau, b^\tau)$	$p(e_2 \mid o^\tau, b^\tau)$	$p(e_3 \mid o^\tau, b^\tau)$	$p(e_4 \mid o^\tau, b^\tau)$	H	$H \le \theta_H$
Episode A	0.00	0.00	0.00	1.00	0.00	0.02	TRUE
Episode B	0.17	0.21	0.22	0.23	0.17	2.31	FALSE

Box 8: An example of restricting relevant episodes based on entropy.

Here, the same set of nine episodes examined in Box 4 is presented. Along with the similarity value (ρ_E), in this example, the entropy value (H) for each episode is also shown. Suppose that the similarity threshold (θ_ρ), the upper limit (K), and the entropy threshold (θ_H) are set to 0.01, 3, and 1.00, respectively. In this case, according to Equation 3.22, E_5 is the only episode whose event is legitimately matched (i.e., $E_5 \in \hat{M}_{\text{rel}}$) because it is one of the K latest episodes whose similarity values is above the threshold, and its entropy is kept below the threshold.

Table 13: The similarity values (ρ_E), relevance, and entropy values (H) of sample episodes.

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
ρ_E	0.014	0.009	0.012	0.011	0.010	0.013	0.008	0.015	0.006
$\rho_E \geq \theta_\rho$, $K = 3$					✓ Relevant	✓ Relevant		✓ Relevant	
H	0.02	1.33	0.04	1.35	0.06	1.37	0.08	1.39	1.10
$H \leq \theta_H$	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE

3.2.3 Behavior Selection

Based on the defined state space and estimated current state, the most suitable action for proactive intelligent behavior can be computed. Here, the action is selected in terms of behavior as a set of motor schemata (Equation 3.2). In other words, a set of motor schemata that is expected to lead the robot to the most rewarding situation is determined. As in a standard Markov decision process (MDP) problem, the notion of *utility* is incorporated into this computation. By applying a Bellman equation [25], the utility (U) of each event in an episode is computed as shown in Equation 3.23:

$$U(e_i) = r_i + \kappa_U \sum_{e' \in E} p(e' \mid b_{i+1}, e_i)\, U(e') \tag{3.23}$$

where r_i is the reward value stored in e_i, and κ_U is a factor that determines the influence of other events. It should be noted that $p(e' \mid b_{i+1}, e_i)$ is the same transition probability employed in the transition model (Equation 3.18). The example in Box 9 shows the utility values of events in a sample episode. Generally, in an MDP problem, the Bellman equation has to be iterated for a number of times to obtain converged utility values (value iteration) [171]. On the other hand, here, because of the events forming a unidirectional temporal linear chain[10], from the end event to the start event, the utility value can be computed by a recursive (dynamic programming [25]) fashion without any iteration. Furthermore, the utility computation does not in fact have to be carried out each time when an action is determined. It has to be performed only once when the episode is added to the memory.

[10] ε_m in the transition probability (Equation 3.18) is zero in this case.

Box 9: An example of the utility computation.

Suppose that a previously stored episode (Episode 1) consists of four events whose reward values (*r*) are recorded as shown in the table below. Note that the events in the table are ordered from the newest one (e_4) to the oldest one (e_0) as their utility values are recursively computed in this order (i.e., from $U(e_4)$ to $U(e_0)$) via Equation 3.23. For example, $U(e_3)$ is calculated based on $U(e_4)$, and $U(e_2)$ is calculated based on $U(e_3)$ and $U(e_4)$.

Table 14: The utility values of the events in Episode 1.

Episode 1	e_4	e_3	e_2	e_1	e_0
r	1.00	0.50	0.00	0.00	0.00
$U(e_i)$	1.00	0.87	0.50	0.41	0.31
$p(e_4 \mid b_{i+1}, e_i)\,U(e_4)$	—	0.37	0.18	0.06	0.02
$p(e_3 \mid b_{i+1}, e_i)\,U(e_3)$	—	—	0.32	0.16	0.05
$p(e_2 \mid b_{i+1}, e_i)\,U(e_2)$	—	—	—	0.19	0.09
$p(e_1 \mid b_{i+1}, e_i)\,U(e_1)$	—	—	—	—	0.15
$p(e_0 \mid b_{i+1}, e_i)\,U(e_0)$	—	—	—	—	—

Based on the utility value associated with each event, the optimal behavior (b^*) that attains the highest utility value is determined by the following maximization equation:

$$b^* = \arg\max_b \frac{1}{|\Gamma^+(b)|} \sum_{E \in \Gamma^+(b)} \sum_{e' \in E} p(e' \mid b, \hat{e}_{[E]})\, U(e') \tag{3.24}$$

where $p(e'|b_{i+1}, e_i)$ is the same transition probability employed in Equations 3.18 and 3.23; and, Γ^+ is a function that returns a subset of episodes from $\hat{M}_{\text{rel}}$. More specifically, given a behavior (*b*), Γ^+ returns a special case of episodes in $\hat{M}_{\text{rel}}$ where the matched events ($\hat{e}_{[E]}$) in these episodes are followed by events storing *b*:

$$\Gamma^+(b) = \{\forall E \,\big|\, E \in \hat{M}_{\text{rel}} \wedge \{e_i, e_{i+1}\} \subseteq E \wedge e_i = \hat{e}_{[E]} \wedge b \in e_{i+1}\} \tag{3.25}$$

An example of selecting the optimal behavior is shown in Box 10.

Box 10: An example of selecting the optimal behavior.

Suppose that a robot experienced three episodes (E_A, E_B, and E_C) shown in the tables below. Let us define a function, f_{UB}, that computes the expected utility of a behavior:

$$f_{\text{UB}}(b, E) = \sum_{e' \in E} p(e' \mid b, \hat{e}_{[E]})\, U(e')$$

The outputs of f_{UB} are also listed in the tables. Equation 3.24 can be then expressed as:

$$b^* = \arg\max_b \frac{1}{|\Gamma^+(b)|} \sum_{E \in \Gamma^+(b)} f_{\text{UB}}(b, E)$$

Thus, having two behavioral types (b_{MF} and b_{SO}), b^* is chosen by comparing these two:

$$(1)\quad \frac{1}{|\Gamma^+(b_{\text{MF}})|} \sum_{E \in \Gamma^+(b_{\text{MF}})} f_{\text{UB}}(b_{\text{MF}}, E) = \frac{1}{\left|\{E_A, E_B\}\right|}\left(f_{\text{UB}}(b_{\text{MF}}, E_A) + f_{\text{UB}}(b_{\text{MF}}, E_B)\right)$$

$$(2)\quad \frac{1}{|\Gamma^+(b_{\text{SO}})|} \sum_{E \in \Gamma^+(b_{\text{SO}})} f_{\text{UB}}(b_{\text{SO}}, E) = \frac{1}{\left|\{E_C\}\right|} f_{\text{UB}}(b_{\text{SO}}, E_C)$$

Note that these are the averaged expected utilities of executing b_{MF} and b_{SO}, respectively. By substituting the f_{UB} values into these equations, the output of (1) becomes 0.31, higher than the output of (2) that is 0.28. Hence, b_{MF} is selected as the optimal behavior (b^*).

Table 15: The expected utility of executing b_{MF} given episode E_A.

E_A	e_{A0}	e_{A1}	e_{A2}	Localized e_{A3}	e_{A4}	$f_{\text{UB}}(b_{\text{MF}}, E_A)$
b	b_{MF}	b_{MF}	b_{MF}	b_{MF}	b_{MF}	—
$U(e)$	0.31	0.41	0.50	0.87	1.00	—
$p(e_i \mid b_{\text{MF}}, e_{A3})U(e_i)$	0.00	0.00	0.00	0.00	0.37	0.37

Table 16: The expected utility of executing b_{MF} given episode E_B.

E_B	e_{B0}	Localized e_{B1}	e_{B2}	e_{B3}	e_{B4}	$f_{\text{UB}}(b_{\text{MF}}, E_B)$
b	b_{MF}	b_{MF}	b_{MF}	b_{MF}	b_{MF}	—
$U(e)$	0.19	0.25	0.32	0.37	1.00	—
$p(e_i \mid b_{\text{MF}}, e_{B1})U(e_i)$	0.00	0.00	0.12	0.07	0.06	0.25

Table 17: The expected utility of executing b_{SO} given episode E_C.

E_C	e_{C0}	Localized e_{C1}	e_{C2}	e_{C3}	e_{C4}	$f_{\text{UB}}(b_{\text{SO}}, E_C)$
b	b_{SO}	b_{SO}	b_{SO}	b_{SO}	b_{SO}	—
$U(e)$	0.22	0.28	0.34	0.68	0.50	—
$p(e_i \mid b_{\text{SO}}, e_{C1})U(e_i)$	0.00	0.00	0.13	0.13	0.03	0.28

Note that Equation 3.24 is the equivalent of how an optimal policy is computed in a standard MDP problem. However, while the standard MDP assumes only one state space and one estimated current state, there can be more than just one estimated current state (event) space since there can be more than just one state space (episode). The number of state spaces is the number of episodes returned by function Γ^+. Hence, the expected utility of executing b is averaged over the number of episodes returned by function Γ^+. For instance, in the previous example (Box 10), since Γ^+ for the first behavior (b_{MF}) returned two episodes (E_A and E_B), the expected utility of executing b_{MF} was averaged over the two episodes. If two or more different behaviors have the exactly same expected utility value, the behavior is chosen randomly among them.

3.2.4 Validation

The behavior attained through the processes of recollection, event matching, and behavior selection above assumes that the current world is the same world that the robot interacted with when those utilized episodic memories were formed. In other words, by executing a known sequence of actions in a familiar environment, the consequence of the actions is assumed predictable. Furthermore, the maximization equation (Equation 3.24) infers that it is the most profitable choice. Hence, the type of proactive intelligent behavior computed by these processes is here referred to as *anticipatory behavior*. However, in reality, the static world assumption does not always hold. The current environment may not be quite the same as the one the robot interacted with before.

Thus, the *validation* process provides the robot with a chance to examine whether the recalled episode indeed represents the current state space accurately. This examination is done by monitoring how events progress when the robot interacts with the current world and comparing them against the ones stored in the recalled episode. The following function, Δ_E, quantifies the delay of current event progress with respect to the schedule specified in the recalled episode:

$$\Delta_E(t) = \frac{t - T(e_i)}{T(\hat{e}_{[E]i+1}) - T(\hat{e}_{[E]i})} - 1 \tag{3.26}$$

where t is the current time; $T(e_i)$ is the timestamp of the event that was sampled most recently (at instance *i*); $T(\hat{e}_{[E]i})$ is the timestamp of the event from E that was matched at instance *i*; and $T(\hat{e}_{[E]i+1})$ is the timestamp of the event from E that was sampled right after $\hat{e}_{[E]i}$. In other words, Δ_E determines how much the occurrence of the current event is delayed with respect to the expected interval specified in the recalled episode. If the occurrences of events are found to be on schedule, an episode is classified as *valid* (Equation 3.27). On the other hand, if the delay of the event progress exceeds a predefined threshold (θ_Δ), the episode is classified as *invalid* and suspended from performing event matching.

$$f_{\text{valid}}(t, E) = \begin{cases} \text{true} & \text{if } \Delta_{\text{E}}(t) \le \theta_\Delta \\ \text{false} & \text{otherwise} \end{cases} \tag{3.27}$$

Similar to the entropy threshold (Equation 3.21), θ_Δ is also determined empirically. Having a too large threshold could result in delaying or even failing the detection of invalid episodes. On the other hand, if the threshold is smaller than the range of ordinary noise, it could end up eliminating valuable episodes. For example, in our experiments (Chapter 5), the threshold value was set to 5.0 as the number was found to work reasonably well detecting invalid episodes. Nevertheless, a more sophisticated method to determine a right threshold value should be addressed in the future. An example of this validation process is shown in Box 11. If all of the recalled episodes are found invalid, and there is no relevant episode left to select the output behavior, anticipation is considered to have failed. Therefore, in this case, the recovery process (improvisation) has to be invoked next.

Box 11: An example of validation.

Suppose a previously stored episode (Episode 1) consists of four events whose time stamps are recorded as shown in the table below. Suppose also that the current time is 5003, and the robot is being matched to event e_3 since time 5000. Using Equation 3.26, the delay of current event progress is calculated as:

$$\Delta_E(5003) = \frac{5003 - T(e_i)}{T(e_4) - T(e_3)} - 1 = \frac{5003 - 5000}{1008 - 1006} - 1 = 0.5$$

Hence, if the threshold (θ_Δ) is set to 0.5 or above, Episode 1 will be classified by Equation 3.27 as *valid*; on the other hand, the episode will be classified as *invalid* if the threshold is below 0.5.

Table 18: The timestamps of the events in Episode 1.

Episode 1	e_0	e_1	e_2	Localized e_3	e_4
Time Stamp	1000	1002	1004	1006	1008

3.2.5 Recovery

Even if the anticipatory aspect of proactive intelligent behavior could not be attained, the robot may still be able to compute the improvisational aspect of proactive intelligent behavior via a recovery process. The recovery process attempts to revive the proactive intelligent behavior computation by proposing an intermediate goal. The intermediate goal is then injected back into the processes of recollection, event matching, and behavior selection to re-compute the appropriate behavior. Recall that at each time when an episode is formed, the behavioral progression of the event is outlined by a referent (Section 3.1.3). The recovery process selects an intermediate goal based on a primary referent, where *primary referent* (Ω^*) refers to the referent of the previously recalled episode whose last known matched event has the highest utility value comparing to other relevant episodes:

$$\Omega^* = \arg\max_{\Omega} U(\Omega) \tag{3.28}$$

where $U(\Omega)$ is a function that returns the utility value (Equation 3.23) of the last known matched event that belongs to the episode from which Ω was constructed. The assumption here is that the recalled episode is not an exact representation of the current world to compute anticipatory behavior (since the world appears to have changed), but its basic outline (referent) still holds sufficient information to perform improvisation. An example of selecting a primary referent is shown in Box 12.

Box 12: An example of selecting a primary referent.

Three sample episodes (Episodes 1, 2, and 3) and the utility values of their events are shown in the table (Table 19) below. Suppose that the last known matched events for Episodes 1, 2, and 3 are e_2, e_3, and e_1, respectively. The function, $U(\Omega)$, in Equation 3.28 returns the utility values of those matched events (also shown in the table). In this case, the referent abstracted from Episode 1 is chosen as the primary one since its last known matched event (e_2) has the highest utility value (0.50) comparing to other two.

Table 19: The utility values of sample episodes.

Utility	$U(e_0)$	$U(e_1)$	$U(e_2)$	$U(e_3)$	$U(e_4)$	$U(\Omega)$
Episode 1	0.31	0.41	0.50	0.87	1.00	0.50
Episode 2	0.19	0.25	0.32	0.37	1.00	0.37
Episode 3	0.22	0.28	0.34	0.68	0.50	0.28

In order to select the intermediate goal, the last known matched event is first identified (Figure 27). It is up to this event that the episode was able to represent the current world adequately. One of the nodes in the primary referent is considered an *active node* (ω^*) if the occurrence of the last known event coincided with it:

$$\omega^* = \{\omega \mid \omega \in \Omega^* \wedge T(o_{\mathrm{end}[\omega]}) \geq T(\hat{e}) \wedge T(o_{\mathrm{init}[\omega]}) < T(\hat{e})\} \tag{3.29}$$

where $T(o_{\mathrm{init}[\omega]})$ and $T(o_{\mathrm{end}[\omega]})$ are the timestamps of the nodal precondition and effect of a referent node (ω), respectively; and $T(\hat{e})$ is the timestamp of the last known matched event. Finally, the nodal effect ($o_{\mathrm{end}[\omega^*]}$) of the active node is selected as the intermediate goal (g_{int}):

$$g_{\mathrm{int}} = o_{\mathrm{end}[\omega^*]} \tag{3.30}$$

An example of selecting an intermediate goal is shown in Box 13.

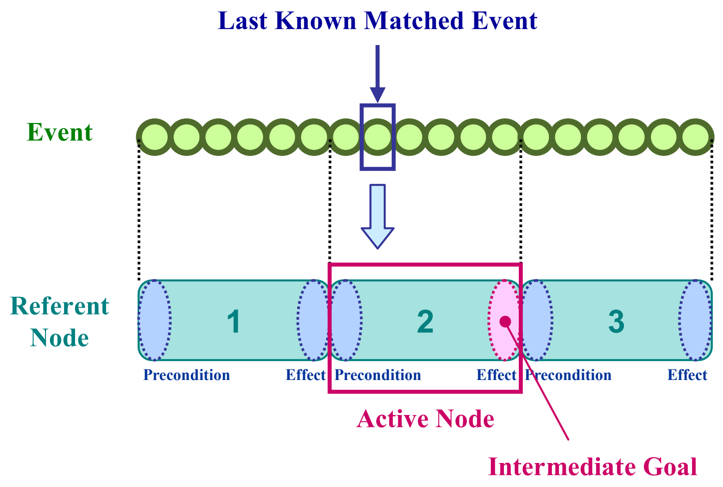


Figure 27: Selection of an intermediate goal. Referent Node 2 is here identified as an active node since the last known matched event resides within this nodal period. Hence, the nodal effect of this active node (i.e., the perceptual state stored inside the last event of this nodal period) is selected as the intermediate goal.

Box 13: An example of selecting an intermediate goal.

Suppose that a previously stored episode (Episode 1) consists of 10 events as shown in the table (Table 20) below. Since its behavior type (b) was altered twice (at events e_3 and e_7), three referent nodes (ω_1, ω_2, and ω_3) are extracted (similar to the diagram in Figure 27). Suppose also that e_4 is the last known matched event. In this case, ω_2 is considered an *active node* according to Equation 3.29. More specifically, the observation at e_2 (i.e., $z = 4.7$) becomes the *nodal precondition* of ω_2 ($o_{\text{init}[\omega_2]}$) while the observation of e_6 (i.e., $z = 0.9$) becomes the *nodal effect* of ω_2 ($o_{\text{end}[\omega_2]}$). Since e_4 occurred after $o_{\text{init}[\omega_2]}$ and before $o_{\text{end}[\omega_2]}$, ω_2 is qualified to be an active node. Thus, $o_{\text{end}[\omega_2]}$ that is the observation of e_6 ($z = 0.9$) becomes a new intermediate goal (Equation 3.30).

Table 20: The perceptual information (z), behavioral information (b), referent nodes (ω), and timestamps (T) of the events in Episode 1.

Episode 1	e_0	e_1	e_2	e_3	Localized e_4	e_5	e_6	e_7	e_8	e_9
z	9.1	5.8	4.7	5.7	5.0	7.1	0.9	4.0	1.9	5.8
b	b_{MF}	b_{MF}	b_{MF}	b_{SO}	b_{SO}	b_{SO}	b_{SO}	b_{MF}	b_{MF}	b_{MF}
ω	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_3	ω_3	ω_3
T	5000	5002	5004	5006	5008	5010	5012	5014	5016	5018

Overriding the current goal, the selected intermediate goal is injected back into the path of recollection, event matching, and behavior selection. In other words, the intermediate goal becomes the desired perceptual state that the robot is now set to pursue. Accordingly, relevant episodes are recalled based on this goal, a current state is matched with respect to the recalled episodes, and the behavior that is expected to benefit the robot most is attained at last. The behavior computed through this modus operandi is considered *improvisational* as it is indeed a resolution to an unanticipated circumstance of the situation, and the action that is expected to maximize the desired outcome is nevertheless selected from its own previous experiences.

It should be noted, however, that even if the intermediate goal is injected back into the system, it is possible for the recollection process to find no relevant episode in the memory (i.e., there is no episode in the memory whose context is similar to the perceptual state specified in the intermediate goal). In this case, the recovery process is invoked again to find more intermediate goals. More specifically, the nodal effects of the two nodes residing before and after the active node are selected as the new intermediate goals. For instance, in the case of the example shown in Box 13, if the first intermediate goal (the nodal effect of the active node (ω_2)) fails to yield a new set of relevant episodes, the recovery process will then select the nodal effects of ω_1 and ω_3 (i.e., the observations in e_2 and e_9, respectively) as the new intermediate goals. The two intermediate goals are simultaneously injected back into the recollection process. Until the recollection process finds at least one new relevant episode in the memory or none of the nodes in the primary referent is found to yield any relevant episode (i.e., improvisation failure), the recovery process will be invoked indefinitely.

3.3 Auxiliary Functions

The method described above attempts to compute proactive intelligent behavior for any given environment. The relevance of the behavior is determined with respect to a goal that is given to the robot, and its utility is assessed based on the reward signal. Specification of a goal and modulation of the reward signal are computed by two auxiliary functions: namely, the motivation and reward functions, respectively. Here, the motivation and reward functions are considered auxiliary since they serve essential but supportive roles, residing outside the core processes of the proactive intelligent behavior computation.

3.3.1 Motivation Function

Recall that *goal* is defined as a desired perceptual state represented in the form of sensor readings (Equation 3.8). Motivation is defined here as a set (π) consisting of a goal (g) and its magnitude (ψ):

$$\pi = \{g, \psi\} \tag{3.31}$$

Having a high ψ value implies that the robot is highly motivated to activate the goal. A goal becomes active if the motivation to which it belongs has the highest magnitude among possible candidates:

$$g_{\text{cur}} = \{g \mid g \in \pi \wedge (\pi = \arg\max_{\pi \in \Pi} \Psi(\pi))\} \tag{3.32}$$

where Π is a set of all possible motivations, and $\Psi(\pi)$ is a function that returns the magnitude of the motivation.

Note that, at each time cycle, the magnitude of every motivation in Π is adjusted based on the current goal (g_{cur}) and observation (o_{cur}):

$$\psi = f_{\text{motiv}}(g_{\text{cur}}, o_{\text{cur}}) \tag{3.33}$$

More specifically, f_{motiv} is the motivation function that returns the degree of motivation for pursuing a particular goal given the current observation. An example of how to select a current goal is shown in Box 14.

The use of motivation has been exploited by many robotics researchers, especially in behavior-based robotics [12, 29, 124, 146, 169]. In those cases, motivation influences behaviors directly by adjusting behavior parameters such as the activation level. Here, motivation instead influences behaviors by setting a goal, and the goal influences behaviors by recalling relevant episodic memories.

Box 14: An example of selecting a current goal.

Suppose that the current observation (o_{cur}) is expressed as:

$$o_{\text{cur}} = \{z_{\text{Bumper}}, z_{\text{Battery}}\} = \{1.0, 0.9\}$$

where z_{Bumper} and z_{Battery} are the readings of a bumper-sensor and a battery-meter, respectively (both normalized). The numerical values indicate that the robot is currently colliding with an object, and the battery is 90% full. Suppose also that there are two types of motivation: namely, $\pi_{\text{Bump-Free}}$ and $\pi_{\text{Fully-Charged}}$. Like o_{cur}, the goals of these motivational types are expressed in terms of z_{Bumper} and z_{Battery} with the following numerical values:

$$g_{\text{Bump-Free}} = \{0.0, 0.5\} \text{ and } g_{\text{Fully-Charged}} = \{0.5, 1.0\}$$

In other words, $g_{\text{Bump-Free}}$ is a perceptual state at which no object is colliding with the robot while the battery is not necessary full. At $g_{\text{Fully-Charged}}$, the battery is full, but an object may or may not be colliding with the robot. Here, we arbitrarily define a function that determines the magnitude of motivation (Equation 3.33) as:

$$\psi = f_{\text{motiv}}(g, o_{\text{cur}}) = \begin{cases} -z_{\text{Bumper}[g]} + z_{\text{Bumper}[o_{\text{cur}}]} & \text{if } g = g_{\text{Bump-Free}} \\ z_{\text{Battery}[g]} - z_{\text{Battery}[o_{\text{cur}}]} & \text{else if } g = g_{\text{Fully-Charged}} \\ 0 & \text{else} \end{cases}$$

where $z_{\text{Bumper}[g]}$ and $z_{\text{Bumper}[o_{\text{cur}}]}$ are the normalized bumper-sensor readings in g and o_{cur}, respectively; and $z_{\text{Battery}[g]}$ and $z_{\text{Battery}[o_{\text{cur}}]}$ are the normalized battery-meter readings in g and o_{cur}, respectively. Substituting the numerical values into the equation, if $\pi_{\text{Bump-Free}}$ is evaluated (i.e., if g is $g_{\text{Bump-Free}}$), ψ becomes 1.0. On the other hand, if $\pi_{\text{Fully-Charged}}$ is evaluated (i.e., if g is $g_{\text{Fully-Charged}}$), ψ becomes 0.1. Having a larger motivational magnitude, Equation 3.32 will hence select $g_{\text{Bump-Free}}$ as the current goal.

3.3.2 Reward Function

The reward signal indicates how much the current state is desirable for the robot. The reward value, which is a scalar, can be a positive or negative number. When it is positive, it implies that the robot is at some desirable state; on the other hand, the robot is presumed at an undesirable state when the number is negative. Embedded within each event, the reward signal influences the choice of behaviors by providing their utilities (Equation 3.23). Independently from the core processes of the proactive intelligent behavior computation, the reward signal is continuously regulated by a reward function.

The reward function determines the current reward value based on three factors: 1) the similarity between the current goal and observation; 2) the similarity between the predicted and actual observations; and 3) the similarity between the associative rewarding states (explained below) and current observation. When the current observation matches with the current goal, since it is the indication that the goal is met, the reward value is increased. Note that the predicted observation is not the same observation predicted by TD(λ) (Equation 3.5). In this case, the observation is predicted based on the matched events obtained by Equation 3.21. More specifically, given an episode (E), the predicted observation ($o'_{[E]}$) refers to the observation of the event that is stored subsequently to the previously matched event ($\hat{e}_{[E]}$):

$$o'_{[E]\tau} = \{o_i \mid \{e_i, e_{i-1}\} \subseteq E \wedge o_i \in e_i \wedge e_{i-1} = \hat{e}_{[E]\tau-1}\} \qquad (3.34)$$

An example of computing the predicted observation is shown in Box 15.

Box 15: An example of computing the predicted observation.

Suppose that a previously stored episode (Episode 1) consists of seven events as shown in the table below. Suppose also that the robot is previously matched to event e_4. In this case, according to Equation 3.34, the predicted observation (Equation 3.34) is the observation of e_5 (i.e., $z = 7.1$).

Table 21: The perceptual information of the events in Episode 1.

Episode 1	e_0	e_1	e_2	e_3	Localized e_4	e_5	e_6
z	9.1	5.8	4.7	5.7	5.0	7.1	0.9

When the predicted observation matches the current observation, it implies that the episode used to represent the current world is indeed apposite. Hence, it is rewarded accordingly. The associative rewarding states are particular perceptual states that are intrinsically important for the robot. For example, a voltage reading that indicates the battery is fully charged can be one of the associative rewarding states. On the other hand, a perceptual state indicating that the robot is violently hit by an object may be considered an unrewarding (punishing) state. These states can be preprogrammed into the robot before it starts interacting with the real world or eventually learned through experiences. (The exact mechanism of how the associative rewarding states are learned is beyond the scope of this dissertation.) If any of the associative rewarding states matches with the current observation, such a situation is correspondingly rewarded (or punished). This concept is in fact related to how certain stimuli in the environment are associative with an animal's certain emotions (e.g., snakes being associated with fear). As mentioned in Section 2.1.2, according to the somatic marker hypothesis [45], the emotional responses induced by perceiving such stimuli are internally converted into somatosensory signals and incorporated into the animal's memory along with other sensory signals. Indeed, the associative rewarding states are the equivalents of those stimuli, and the reward signal generated by the associative rewarding states and saved in an episodic memory is considered the equivalent of a somatic marker.

As shown in Equation 3.35, each of the factors above is weighted by a predefined constant in order to attain the current reward value (r_{cur}):

$$r_{\text{cur}} = \kappa_g f_{\text{L}}(o_{\text{cur}}, g_{\text{cur}}) + \kappa_o \max_{E \in \Gamma^{+}(b^{*})} f_{\text{L}}(o_{\text{cur}}, o'_{[E]}) + \sum_{\iota \in \text{I}} \kappa_\iota f_{\text{L}}(o_{\text{cur}}, \iota) \tag{3.35}$$

where κ_g, κ_o, and κ_ι are the weights for the current goal (g_{cur}), predicted observation ($o'_{[E]}$), and associative rewarding state (ι), respectively. The similarities are computed by the same likelihood function (f_{L}) employed in Equations 3.14 and 3.17. Here, the rations among the predefined constants are more important than their values themselves. For example, as experimented in Section 5.2, if the influence of the somatic markers is desired to be substantial, κ_ι should be a significantly larger number than others (or zero if the influence should be eliminated). While a method to learn these weights should be addressed in the future, here, the values are assigned manually. An example of computing the current reward value is shown in Box 16.

Box 16: An example of computing the current reward value.

Similar to the example examined in Box 14, suppose that the current observation (o_{cur}) is set as:

$$o_{\text{cur}} = \{z_{\text{Bumper}}, z_{\text{Battery}}\} = \{1.0, 0.9\}$$

indicating that the robot is currently colliding with an object, and the battery is 90% full. Suppose also that the current goal (g_{cur}), the predicted observation (o'), and an associative rewarding state (ι) are respectively set as:

$$g_{\text{cur}} = \{0.0, 0.5\},\ o' = \{1.0, 0.8\},\text{ and }\iota = \{0.0, 1.0\}$$

Recall the Gaussian function employed to calculate similarity values in Box 3. The first term of Equation 3.35 can be then calculated by:

$$\kappa_g f_{\text{L}}(o_{\text{cur}}, g_{\text{cur}}) = \kappa_g f_{\text{Gauss}}(o_{\text{cur}})_{g_{\text{cur}}} = \frac{\kappa_g}{(2\pi)^2}\exp\left(\frac{(o_{\text{cur}} - g_{\text{cur}})^{\text{T}}(o_{\text{cur}} - g_{\text{cur}})}{-2}\right)$$

Assuming the weight (κ_g) is 1.0, inserting the numerical numbers to the equation yields:

$$\kappa_g f_L(o_{\text{cur}}, g_{\text{cur}}) = \frac{1}{(2\pi)^2}\exp\left(\frac{(1.0-0.0)^2 + (0.9-0.5)^2}{-2}\right) = 0.0891$$

Similarly, the second and the third terms are calculated as:

$$\kappa_o f_L(o_{\text{cur}}, o') = \frac{1}{(2\pi)^2}\exp\left(\frac{(1.0-1.0)^2 + (0.9-0.8)^2}{-2}\right) = 0.1584$$

$$\kappa_\iota f_L(o_{\text{cur}}, \iota) = \frac{1}{(2\pi)^2}\exp\left(\frac{(1.0-0.0)^2 + (0.9-1.0)^2}{-2}\right) = 0.0961$$

Combining all these values, the current reward value (r_{cur}) hence becomes:

$$r_{\text{cur}} = 0.0891 + 0.1584 + 0.0961 = 0.3436$$

3.4 Discussion

In this section, the computational model described above is examined from two different perspectives. First, the relevancy of the computational model to existing biological studies is discussed. Then, the machine learning aspect of the model is elucidated.

3.4.1 Biological Relevance

As noted above, the computational model described here is inspired by how a mammalian hippocampus works. While it is not a high fidelity model to explicate every detail of hippocampal physiology, major hippocampal functions are approximated by this model. In particular, the proposed data structure to implement episodic memory is based on the notion of "memory space" proposed by Eichenbaum and his colleagues [51]. As discussed in Chapter 2, this notion conflicts with the conventional belief that the hippocampal constructs a two-dimensional Euclidean cognitive map, transforming the egocentric view that an animal perceives into the geocentric framework via path-integration. Even though this 2D map hypothesis has been explained through various experiments using rats and two-dimensional mazes in laboratories, it does not elucidate why many wild mammals such as monkeys, squirrels, and chipmunks can effectively navigate on trees, a three-dimensional space. On the other hand, the memory-space notion of episodic memories does not suffer from such a spatial dimensionality limitation as it constructs a world model temporally. Furthermore, having both sensory and behavioral information integrated within a single event, it is also in accordance with the theory of event coding (TEC) [74], grounded in a series of psychological studies.

The computational model also implements novelty detection, one of the presumed hippocampal functions. In fact, there are two levels of novelty detection implemented within the model, and they are processed in different time scales. The first level is performed during the event sampling process. By employing TD(λ) [170], predicted sensor readings are constantly computed and compared against the actual readings. When the prediction fails, the robot is considered to be entering a new state. Although classical conditioning has been modeled using TD(λ) [153, 154, 172], modeling sensory prediction using this algorithm has not been particularly asserted by these scientists. Nevertheless, various hippocampal researchers have propositioned that such functionality itself indeed exists within areas CA1 and CA3 [68, 69, 97, 119, 192].

The second level of novelty detection is conducted during the validation process. By comparing a current event progression against the one stored in a recalled episode, if they start diverging, the robot is considered to be entering a new state space. This is perhaps the same level with which Hoffmann's anticipatory behavioral control (ABC) framework [72] operates. As discussed in Chapter 2, the ABC framework is also grounded on various studies in psychology, and its main premise is that when some voluntary action is executed, the predicted consequence is compared against the actual effect. The causal relationship is reinforced if the prediction turns out to be a valid one.

Finally, Damasio's somatic marker hypothesis [45] is also incorporated into our computational model. More specifically, a reward signal embedded within an event (Equation 3.3) is considered the equivalent of a somatic marker. Recall that emotionally induced somatosensory signals quantify how painful or pleasurable the animal feels about certain stimuli. Along with other sensory information, the somatosensory signals are integrated into episodic memory. As demonstrated by gambling experiments [23, 24], when such memories are recalled, the embedded somatosensory signals seem to help the animal determine the expected utility of current action. Similarly, in our computational model, the value of the reward signal is modulated by the reward function based on how desirable the current state is. After being integrated into episodic memories, the embedded reward signals also help the robot compute the expected utility of the current action. While the computational model employs a Bellman equation to compute the expected utilities (Equation 3.23), it is not clear whether the utility computation in the actual animal's somatic marker circuitry can be modeled by the Bellman equation. Nevertheless, the functionality of somatic markers is approximated by this computational model.

3.4.2 Machine Learning Aspects

The computational method described here can be categorized as a "lazy learning method" as it retains training data in its original form and postpones generalization of the data until the elaboration of a solution is requested. Indeed, the computational processes in this method (i.e., recollection, event matching, behavior selection, validation, and recovery) (Figure 25) bear a moderate resemblance to the case-based reasoning cycle suggested by Kolodner and Leake [91] (Figure 14). Ram and Santamaria's continuous case-based reasoning [137] may be the most relevant CBR approach to this method as they both utilize the temporal aspect of the sensory information. However, the difference is that, while temporal information is used to retrieve a case in continuous CBR, in our method temporal information is used to identify the most relevant event within a case (episode). The case retrieval in our method is done in terms of goals. Furthermore, in continuous CBR, the case-based reasoner proposes actions while, in our method, actions are separately computed by the combination of event-matching and behavior selection processes.

Strictly speaking, however, according to Mitchell's classification [115], our computational method belongs to instance-based learning rather than CBR since the data is described in terms of numerical values, representing certain points in a *n*-dimensional

Euclidean space. A comparable example may be McCallum's implementation of instance-based learning [109, 110]. As reviewed in Chapter 2, in McCallum's method, k instances that are the closest representations of a current state are selected from the memory based on the recent action-perception-reward sequence. A current policy is then determined after averaging Q-values from those k instances. Once again, in our method, a current goal determines the retrieval of the k instances from the memory, not the sequence. The similarity of the two methods is that the k retrieved instances collectively determine a current policy in both cases: with a model-free approach (Q-value) in McCallum's method and with a model-based approach (Bellman equation) in our method.

It should be noted that the computational method described here deals with a partially observable Markov decision process (POMDP) problem (see Section 2.4.4). More specifically, the event-matching process attempts to identify a current state in a POMDP by employing a recursive Bayesian filter. Once the current state is identified, treating the rest as an MDP problem, an optimal policy is determined using a Bellman equation.

Because the state space is organized in a unidirectional temporal linear chain fashion, this computational method can be also considered comparable to predictive state representation [101]. As reviewed in Chapter 2, Littman et al. [101] explained that PSR is a combination of history-based and generative-model approaches; each state retains information regarding a history as well as an expected future consequence, and the state representation is updated generatively. Our computational method is also a combination of history-based and generative-model approaches but in a different way from how it is done in PSR. Our computational method is closer to the generative-model approach than the history-based one as it computes POMDP solutions recursively. Even though the state (event) itself does not maintain information regarding a history or an expected future consequence explicitly as in PSR, it can be easily inferred from the state by referring to the state space (episode) formed in a unidirectional temporal linear chain fashion.

CHAPTER 4

IMPLEMENTATION

This chapter describes the implementation of the computational model detailed in Chapter 3. In particular, the architectural framework that realizes the computational model is explained in the first section. The details of how this architectural framework was further integrated into a complete robotic system are explained in the second section.

4.1 AIR: Anticipatory-Improvisational Robot Architecture

The computational model of proactive intelligent behavior was realized in a Java-based computer program whose architectural framework is illustrated in Figure 28. The architecture, referred to as *AIR* (Anticipatory-Improvisational Robot), is a hybrid deliberative/reactive system, an architectural type widely used in robotics [10]. AIR consists of two layers: namely, the behavioral subsystem (reactive component) and the episodic subsystem (deliberative component). The input and output of the whole AIR system are first explained, followed by the descriptions of the episodic and behavioral subsystems in detail.

4.1.1 I/O

The input to the AIR system is a set of readings from embedded sensors, and the output is a set of motor commands to be executed by the actuators. The data structures used to store these types of information are explained below in detail.

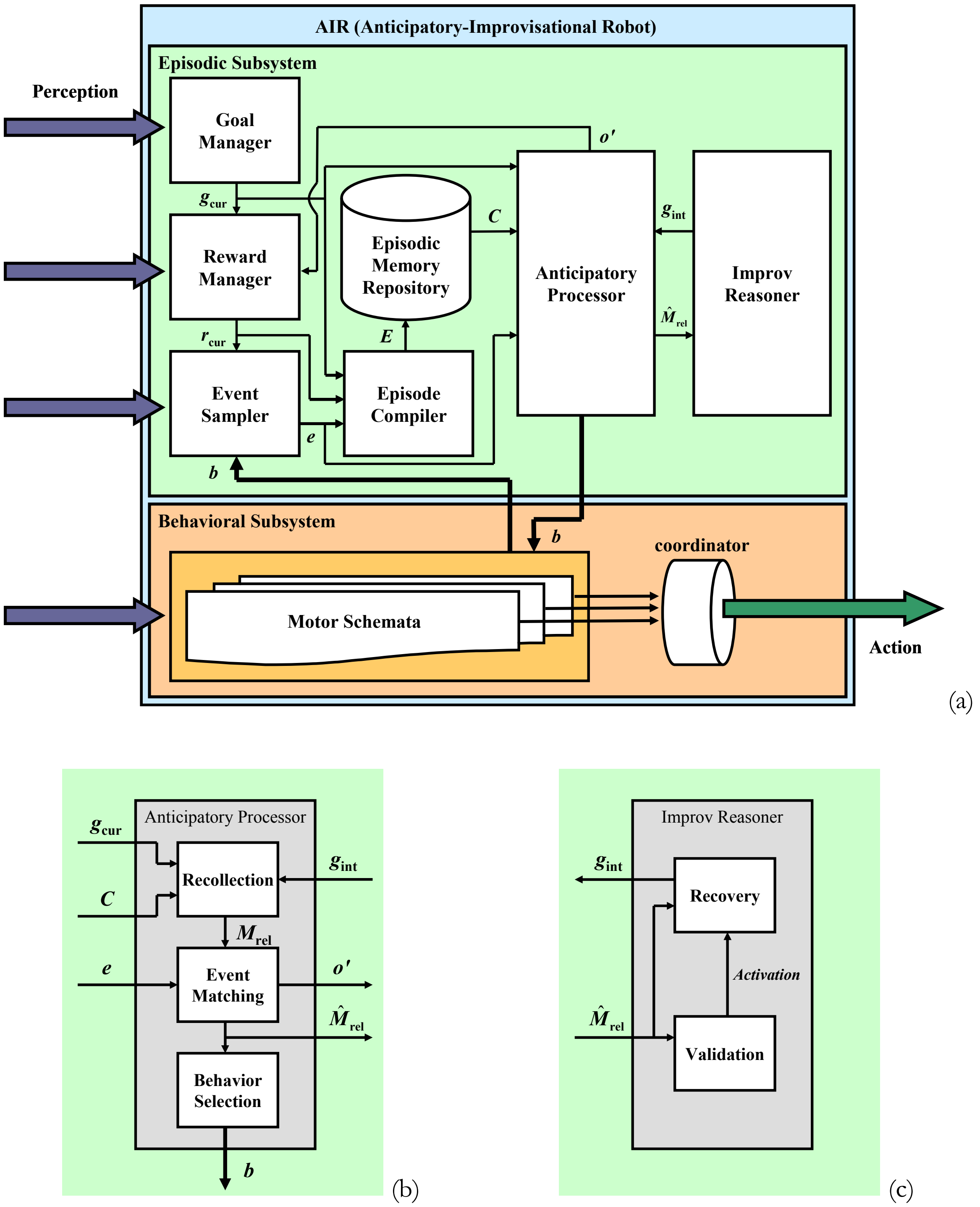


Figure 28: The AIR architecture: (a) the entire system, (b) the anticipatory processor module, and (c) the improvisational reasoner module. (Note: g_{cur} = goal (current), r_{cur} = reward (current), b = behavior, e = event, E = episode, C = past episodes, o' = predicted observation, g_{int} = goal (intermediate), $\hat{M}_{\text{rel}}$ = relevant episodes that contain successfully matched events, and M_{rel} = relevant episodes.)

Input Data Structure

A data structure called *Perception* is used to encapsulate the input information. As shown in Table 22, the sole member of *Perception* is called *readings*, and it stores the list of all sensor readings in an array form. An individual sensor reading is saved in a sub-data structure called *Reading*. As shown in Table 23, *Reading* comprises four members: namely, *id*, *values*, *phi_angles*, and *theta_angles*. The first member, *id*, contains a unique identification number for the sensor type. The second member, *values*, contains the actual data values of the reading. Note that some sensor may return multiple data points per single reading. For example, a laser scanner (SICK LMS200) returns 361 data points as it measures distances to objects in the environment 361 times per scan (with a 0.5-degree increment). Hence, in this case, the size of the array is 361. Note also that the contents of *values* do not have to be distances. If the sensor is a color blob detector, for example, the values can be the sizes of detected blobs. The third and forth members, *phi_angles* and *theta_angles*, specify the horizontal and vertical directions of each data point with respect to the center of the robot, respectively (Figure 29). More specifically, projecting the data points in the (egocentric) spherical coordinates system, each entry in the *phi_angles* array represents the azimuth angle of the corresponding data point measured from the positive X-axis (the robot's current heading direction). On the other hand, each entry in the *theta_angles* array represents the zenith angle of the corresponding data point measured from the positive Z-axis (vertically upwards).

Table 22: The member of *Perception*.

Name	Data Type	Description
readings	*Reading* []	An array containing the latest readings from the sensors.

Table 23: The members of *Reading*.

Name	Data Type	Description
id	int	The unique ID of this sensor.
values	double[]	The data points of the reading.
phi_angles	double[]	The azimuth angles (from the positive X-axis) of the data points.
theta_angles	double[]	The zenith angles (from the positive Z-axis) of the data points.

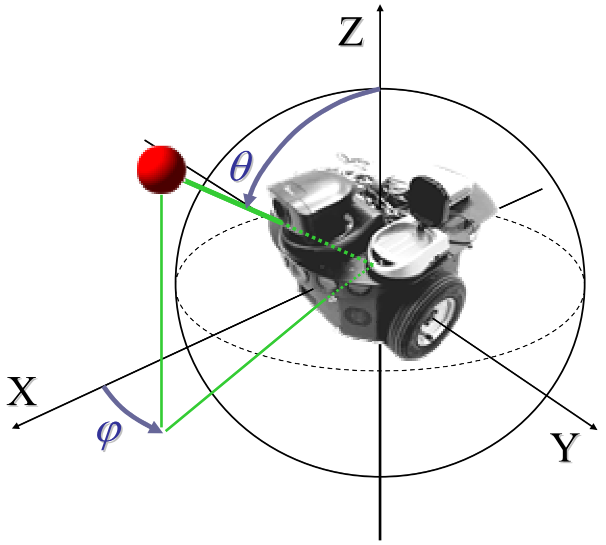


Figure 29: An input data point specified in the spherical coordinate system. In this case, the phi (*φ*) angle of the red ball is approximately 15°, and the theta (*θ*) angle is approximately 45°.

Output Data Structure

The output information (motor commands) is saved in a data structure called *Action*, which consists of six members (Table 24). The first three members (*speed_x*, *speed_y*, and *speed_z*) specify desired speeds (meters per second) along the positive X, Y, and Z axes of the (egocentric) Cartesian coordinate system, respectively (Figure 30). The positive X-axis is the direction of the robot's current heading, the positive Y-axis points horizontally to the left, and the positive Z-axis points vertically upwards. The other three members of *Action* (*speed_yaw*, *speed_pitch*, and *speed_roll*) specify desired rotational speeds (degrees per second) of the robot along its Z, Y, and X axes, respectively. If the robot is a typical wheeled robot, only *speed_x* and *speed_y* (or *speed_yaw*) are utilized as the platform has only two degrees of freedom. On the other hand, if the robot is an autonomous helicopter, all six members may be utilized.

Table 24: The members of *Action*.

Name	Data Type	Description
speed_x	double	The translational speed in the poistive X-axis.
speed_y	double	The translational speed in the positive Y-axis.
speed_z	double	The translational speed in the positive Z-axis.
speed_yaw	double	The rotational speed along the positive Z-axis.
speed_pitch	double	The rotational speed along the positive Y-axis.
speed_roll	double	The rotational speed along the positive X-axis.

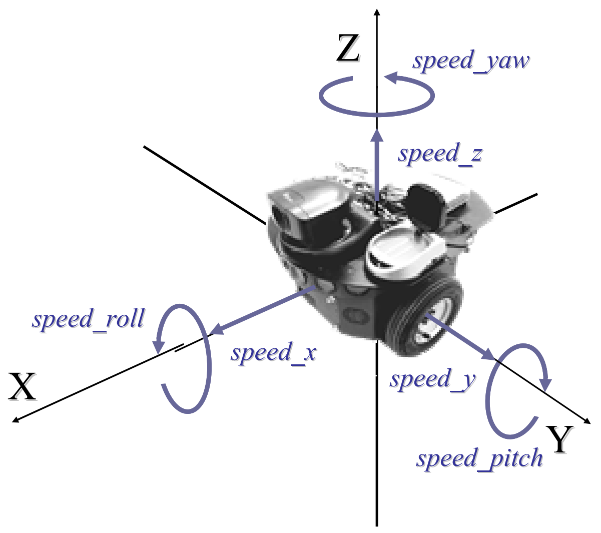


Figure 30: The egocentric Cartesian coordinate system and the motor commands.

4.1.2 Episodic Subsystem

The episodic subsystem implements the computational processes described in Chapter 3. More specifically, the processes of event sampling, episodic memory formation, recollection, referent construction, event matching, behavior selection, validation, and recovery as well as motivation and reward functions are all implemented in the episodic subsystem. The role of the episodic subsystem is to decide what type of motor schemata should be instantiated in the lower reactive layer (behavioral subsystem). The episodic subsystem consists of seven computational modules: goal manager, reward manager, event sampler, episode compiler, episodic memory repository, anticipatory processor, and improvisational reasoner. The functionalities of these modules are explained below.

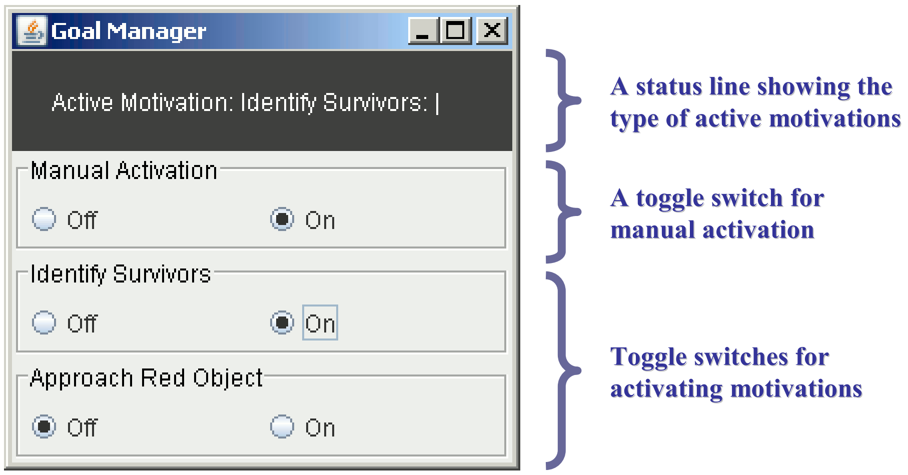
Goal Manager

This module determines the robot's current goal based on the motivation function described in Section 3.3.1. More specifically, there are a finite set of possible goals, and each goal is stored in a data structure called *Motivation* (Table 25). *Motivation* consists of three members: *id*, *goal* and *magnitude*. If one motivation has the highest *magnitude* value among all possible motivations, the goal belongs to this motivation is selected as the current goal. (See the pseudocode in Section B.1.1.) Note that intermediate goals used in improvisation are handled by the improvisational reasoner (described below).

Table 25: The members of *Motivation*.

Name	Data Type	Description
id	int	The unique ID of this motivational type.
goal	*Perception*	The goal state.
magnitude	double	The motivational magnitude.

As described in Section 3.3.1, in a fully autonomous mode, each motivational magnitude is computed by a designated motivation function based on the current observation (Equation 3.33). However, in this implementation, user inputs are also used to modulate the motivational magnitudes, so that the core part of the proactive behavior computation can be examined effectively. In this case, if a user selects one of the available motivation types through the graphical user interface (Figure 31), the magnitude of the selected motivation is set to a predefined positive value (1.0); if the motivation is turned off by the user, the magnitude is set to zero[11]. The current goal (g_{cur}) computed by this module is utilized by the reward manager, the episode compiler, and the anticipatory processor explained below.


Goal Manager

Active Motivation: Identify Survivors: |

Manual Activation: Off [✓ On]

Identify Survivors: Off [✓ On]

Approach Red Object: [✓ Off] On

A status line showing the type of active motivations
A toggle switch for manual activation
Toggle switches for activating motivations

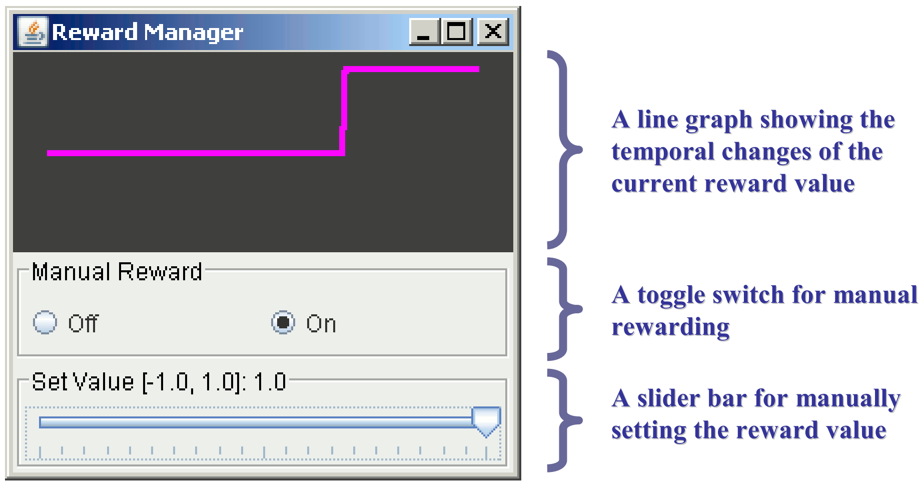
Figure 31: The graphical user interface for the goal manager.

Reward Manager

This module computes the current reward value based on the reward function explained in Section 3.3.2. The reward value is a scalar represented with a real number. More specifically, at each time cycle, the value is updated based on how the current observation is similar to 1) the current goal, 2) the predicted observation, and 3) the associative rewarding sates (Equation 3.35). (See the pseudocode in Section B.2.1.) Note that in the normal (anticipatory) mode, the current goal is the one computed by the goal manager (explained above). However, if the robot is in the improvisational mode, the intermediate goal computed by the improvisational reasoner (explained below) is used instead.

[11] While the current method of magnitude specification is binary (toggle), a slider bar, for example, can be used to specify the value in a continuous range when the motivation function needs to be evaluated.

As shown in Figure 32, the current reward value can be visualized by a graphical user interface. If desired, the user can also override the automatically calculated reward value and set the number manually through this interface (e.g., to examine the effectiveness of the reward signal or test the reward function itself). The current reward value (r_{cur}) computed by this module is utilized by the event sampler and the episode compiler explained below.



Reward Manager
Manual Reward
Off
On
Set Value [-1.0, 1.0]: 1.0
A line graph showing the temporal changes of the current reward value
A toggle switch for manual rewarding
A slider bar for manually setting the reward value

Figure 32: The graphical user interface for the reward manager.

Event Sampler

By implementing the event sampling process described in Section 3.1.1, temporal abstraction of ongoing experience is performed in this module. More specifically, the values of perceptual signals are constantly predicted (Equation 3.4) and compared against the actual values. When the prediction fails (Equation 3.6), the experience (observation, behavior, and reward signals) is sampled as an event (e). (See the pseudocode in Section B.3.1.) Each sampled event is stored in a data structure called *Event*s (Table 26), which comprises five members: namely, *index*, *observation*, *behavior*, *reward*, and *timestamp*. The first member, *index*, stores the index of the event within an episode. Note that this index is used to calculate a distance between two events when computing a transition probability (Equation 3.18). The next three data fields (*observation*, *behavior*, and *reward*) store the values of the observation, behavior, and reward signals, respectively. Note that the observation signal is the system input (Section 4.1.1), the behavior (b) is determined by the behavioral subsystem (Section 4.1.3), and the current reward value (r_{cur}) is the one computed by the reward manager described above. Note that *behavior* is saved in a data structure called *Behavior* (Table 27) whose sole member, *schemata*, is an array of integers storing the unique identifications of the motor schemata being activated when this event was sampled. The last member of *Event* is called *timestamp*, recording the time when the event was sampled. In particular, *timestamp* is utilized during the validation process (Section 3.2.4). If the validation fails (i.e., if the current event progress is substantially slower than the one recorded in the recalled episode), improvisation is invoked. Events (e) sampled in this module are utilized in the episode compiler, anticipatory processor, and improvisational reasoner (described below).

Table 26: The members of *Event*.

Name	Data Type	Description
index	int	The index of the event within an episode.
observation	*Perception*	The perceptual state of the world.
behavior	*Behavior*	The executed behavior.
reward	double	The reward value.
timestamp	long	The time when this event was sampled.

Table 27: The member of *Behavior*.

Name	Data Type	Description
schemata	int []	The IDs of active motor schemata.

Episode Compiler

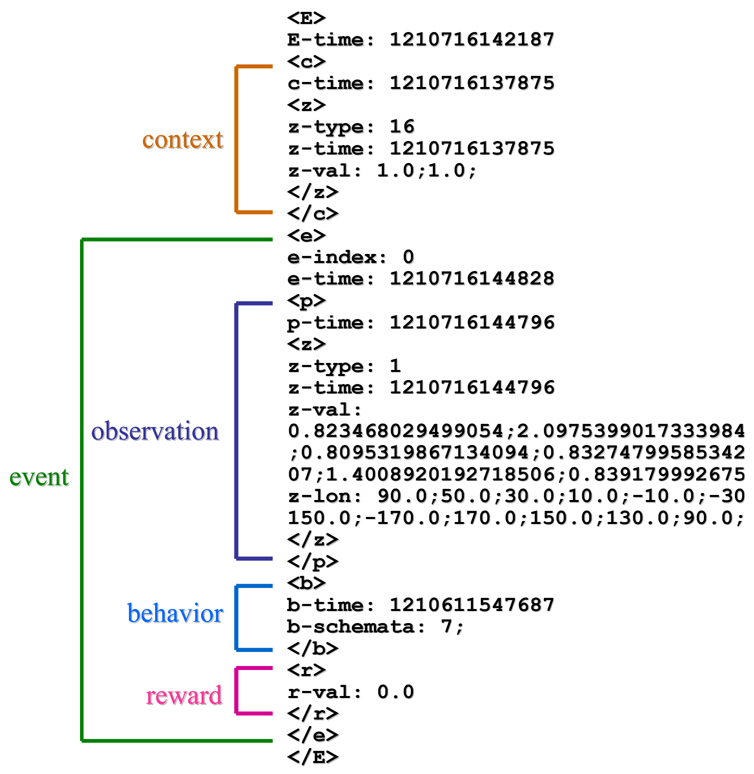
This module compiles episodes by partitioning a series of events arrived from the event sampler into separate subgroups based on their contexts. As explained in Section 3.1.2, depending on the mode, the episodic context can be a current goal (purposive contextualization) (Equation 3.9) or a perceptual state of the instance when the characteristics of the reward value are significantly changed (utilitarian contextualization) (Equation 3.11). In this version of AIR, the purposive contextualization is only supported. (See the pseudocode in Section B.4.1). The data structure used to store an episode is called *Episode* and shown in Table 28. The first member, *events*, stores all the events during this episode in an array of *Event* (recall Table 26). The second member, *context*, stores the context of this episode. Once compiled, episodes are delivered to the episodic memory repository, so that they can be preserved for a future use.

Table 28: The members of *Episode*.

Name	Data Type	Description
events	*Event* []	The series of events.
context	*Perception*	The context of this episode.

Episodic Memory Repository

After being created by the episode compiler, a new episode is added to the list of the episodes maintained by this module. When the program terminates, the list is saved in a text file, so that they can be reloaded to the system memory when the program starts up again. The text format of an example episode is shown in Figure 33. The anticipatory processor (explained below) utilizes the collection of these episodes (C) to compute anticipatory behavior.



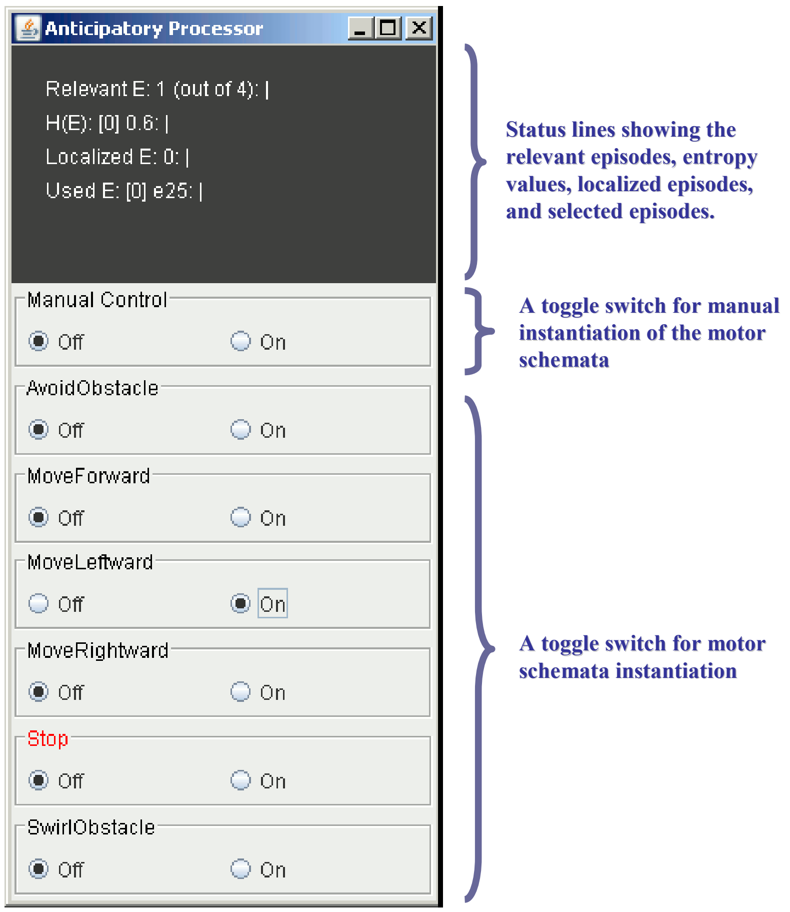
```
<E>
E-time: 1210716142187
<c>
c-time: 1210716137875
<z>
z-type: 16
z-time: 1210716137875
z-val: 1.0;1.0;
</z>
</c>
<e>
e-index: 0
e-time: 1210716144828
<p>
p-time: 1210716144796
<z>
z-type: 1
z-time: 1210716144796
z-val:
0.823468029499054;2.0975399017333984
;0.8095319867134094;0.83274799585342
07;1.4008920192718506;0.839179992675
z-lon: 90.0;50.0;30.0;10.0;-10.0;-30
150.0;-170.0;170.0;150.0;130.0;90.0;
</z>
</p>
<b>
b-time: 1210611547687
b-schemata: 7;
</b>
<r>
r-val: 0.0
</r>
</e>
</E>
```

Figure 33: An example of the text format used to store an episode. The timestamp of the episode is first specified. In the second section, the information regarding the episodic context is stored. The third section contains the information regarding the event sequence (only one event is saved in this case), including the index, time, observation, behavior, and reward of each event.

Anticipatory Processor

This module computes anticipatory behavior by performing the processes of recollection, event matching, and behavior selection discussed in Sections 3.2.1, 3.2.2, and 3.2.3, respectively. More specifically, based on the current goal (g_{cur}) specified by the goal manager, a set of relevant episodes are first selected from the past episodes (C) stored in the episodic memory repository (Equation 3.15). (See the pseudocode in Section B.5.1.) Given the current sequence of events (e) that are arrived from the event sampler, the posterior probability of each event in every relevant episode is then calculated (Equation 3.16). If the entropy of the posterior probability distribution is low enough, the event with the highest posterior probability value is chosen as the matched event (Equation 3.21). (See the pseudocode in Section B.5.2.) Finally, by analyzing the episode, if the behavior executed just after this past matched event has the highest expected utility value, it is chosen as the output of this module (Equation 3.24). (See the pseudocode in Sections B.5.3 and B.5.4.)

The window shown in Figure 34 is a graphical user interface for this module. The top portion of the window displays the status of the computation. More specifically, the current relevant episodes, their entropy values, episodes with matched events, and utilized episodes for the behavioral selection are displayed. This interface also allows the user to select types of desired motor schemata manually when generating training episodes during the experiments (Section 5).



Relevant E: 1 (out of 4): |
H(E): [0] 0.6: |
Localized E: 0: |
Used E: [0] e25: |

Manual Control: [✓ Off] On
AvoidObstacle: [✓ Off] On
MoveForward: [✓ Off] On
MoveLeftward: Off [✓ On]
MoveRightward: [✓ Off] On
Stop: [✓ Off] On
SwirlObstacle: [✓ Off] On

Status lines showing the relevant episodes, entropy values, localized episodes, and selected episodes.

A toggle switch for manual instantiation of the motor schemata

A toggle switch for motor schemata instantiation

Figure 34: The graphical user interface for the anticipatory processor.

As the output of this module, a set of desired motor schemata is stored in *Behavior* (Table 27) and utilized in the behavioral subsystem discussed below (Section 4.1.3). In addition, a set of all relevant episodes that had successful event matching ($\hat{M}_{\text{rel}}$) (Equation 3.22) is sent to the improvisational reasoner, so that an intermediate goal can be determined; and the predicted observation (o') (Equation 3.34) is sent to the reward manager for determining the current reward value.

Improvisational Reasoner

This module implements the improvisational aspect of the proactive intelligent behavior computation. A set of all relevant episodes that were successfully matched ($\hat{M}_{\text{rel}}$) is delivered to this module from the anticipatory processor. This module constantly monitors the event progress of those episodes based on the method described in Section 3.2.4. (See the pseudocode in Section B.6.1.) If the progress of the events is found to be significantly delayed, that episode is eliminated from the list of relevant episodes (Equation 3.27). If all of the episodes are eliminated from the list, the recovering process is then invoked.

To recover the proactive behavior computation, as discussed in Section 3.2.5, a primary referent has to be first selected from the previous list of the relevant episodes that contain matched events (Equation 3.28). We utilize the previous list because the current list is empty after eliminating the delayed episodes. The procedures for abstracting a referent from episode and selecting a primary referent from possible referents are described by the pseudocode in Sections B.6.2 and B.6.3, respectively. The data structure used to store a referent (*Referent*) is shown in Table 29. Its sole member, *nodes*, is an array storing the sequence of referent nodes. The sub-data structure used to store a referent node (*ReferentNode*) is shown in Table 30. The first member of *ReferentNode* is called *behavior*, storing the type of behavior executed during this nodal period. The second and third members (*precondition* and *effect*) store the perceptual states captured just before and after the behavioral execution, respectively. The fourth member, *timestamps*, copies the timestamps of the events sampled before and after the behavioral execution.

Table 29: The member of *Referent*.

Name	Data Type	Description
nodes	*ReferentNode* []	The nodes of the referent.

Table 30: The members of *ReferentNode*.

Name	Data Type	Description
behavior	*Behavior*	The executed behavior.
precondition	*Perception*	The initial perceptual state before the behavior was executed.
effect	*Perception*	The final perceptual state after the behavioral execution was completed.
timestamps	int[2]	The timesamps of the events just before and after the behavior was executed.

Finally, based on the primary referent, an intermediate goal is selected via Equation 3.29. More specifically, analyzing each node in the primary referent, if the timestamp of the last known matched event coincides with any of these nodes, that node is chosen as the active node. The nodal effect stored in the active node is then chosen as the final intermediate goal. (See the pseudocode in Section B.6.4). As the output of this module, the intermediate goal is delivered to the anticipatory processor, so that the processes of the recollection, event matching, and behavior selection can be reactivated.

The graphical user interface for the improvisational reasoner is shown in Figure 35. The interface visualizes the progression of the referent nodes and identifies what the current active node is. If desired (e.g., for debugging), the user can force the module to activate the recovery process from this window, bypassing the validation process.

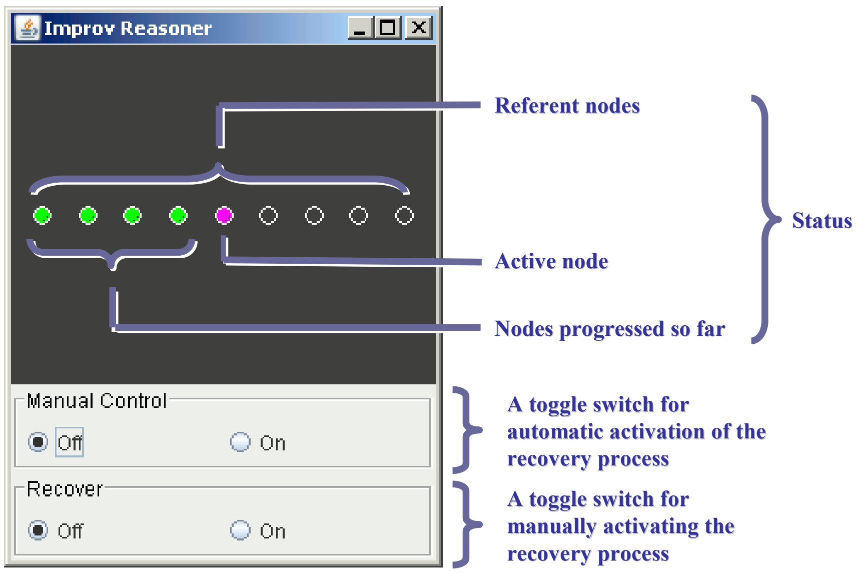

Improv Reasoner
Referent nodes
Status
Active node
Nodes progressed so far
Manual Control
Off
On
A toggle switch for automatic activation of the recovery process
Recover
Off
On
A toggle switch for manually activating the recovery process

Figure 35: The graphical user interface for the improvisational reasoner.

4.1.3 Behavioral Subsystem

A set of preprogrammed motor schemata (see Section 3.1.1) resides within the behavioral subsystem, and a subset of them is periodically instantiated (or de-instantiated) by the upper deliberative layer (episodic subsystem). The function of the behavioral subsystem is to compute the outputs of the active motor schemata and coordinate them to produce the resultant low-level motor commands (Section 4.1.1). The implemented motor schemata and coordinators are summarized below. The details of the individual algorithms are also described as pseudocode in Appendix B.

Motor Schemata

- **Avoid-Static-Obstacle**: A standard obstacle avoidance method [10, 11]. The robot moves away from detected obstacles by generating repulsive vectors from them (Section B.7.1).
- **Enter-Opening**: A modified version of the *docking* motor schema [10, 13]. Given a

detected opening in the environment (e.g., door), the robot attempts to enter the opening by combining ballistic and controlled movements (Section B.7.2).

- **Move-Backward**: The robot moves towards the direction that is opposite of the current heading (Section B.7.3).
- **Move-Forward**: The robot moves towards the same direction it is current heading (Section B.7.4).
- **Move-Leftward**: The robot moves towards the direction that is perpendicularly left of the current heading (Section B.7.5).
- **Move-Rightward**: The robot moves towards the direction that is perpendicularly right of the current heading (Section B.7.6).
- **Move-To-Big-Blob**: A modified version of the *Move-To-Goal* motor schema [10, 11]. The robot moves to a goal perceived as the biggest blob in the field of view (Section B.7.7).
- **Stop**: The robot stops moving (Section B.7.8).
- **Swirl-Obstacle**: An alternative method of negotiating an obstacle as described in [52]. Instead of generating a repulsive vector away from an obstacle, it generates a movement vector tangential to the surface of the obstacle. Based on this vector, the robot circumnavigates the obstacle (Section B.7.9).

<u>Behavior Coordinator</u>

- **Cooperative-Coordinator**: As implemented in Arkin's motor schema based navigation [10, 11], the output vectors of active motor schemata (activated by the anticipatory processor) are linearly summed to generate a single output vector (Section B.8.1).
- **Subsumptive-Coordinator**: Proposed by Brooks [31], motor schemata are organized in

priority-based hierarchical layers (e.g., motor schemata with the highest priority reside in the top layer). As shown in the pseudocode (Section B.8.2), by inspecting the layers from top to bottom, if a layer with a higher number has an active motor schema, the output of the schema is used as the final output, subsuming the outputs from lower layers. If multiple motor schemata are found active in one layer, their outputs are coordinated by the cooperative-coordinator described above. In this particular implementation (Figure 36), there are two layers; the top layer contains the *Stop* schema, and the rest of the motor schemata reside in the bottom layer. In other words, if the *Stop* schema is activated, all other motor schemata are halted.

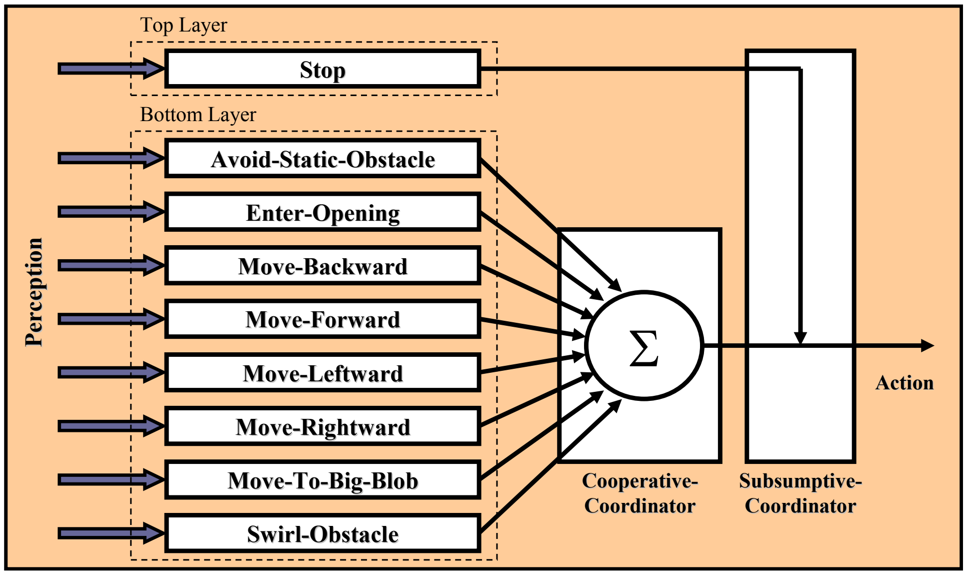


Figure 36: Coordination of the motor schemata. If the *Stop* schema, the sole schema in the top layer, is activated, the output from the bottom layer (holding the rest of the motor schemata) will be suppressed by the subsumptive-coordinator. Otherwise, the behavioral output is a linear summation of the outputs from all active motor schemata in the bottom layer, coordinated by the cooperative-coordinator.

4.2 System Integration

As shown in Figure 37, the Java-based computer program that implements AIR was integrated into a complete robotic framework. More specifically, AIR was interfaced with

HServer [62], a low-level controller program integrated with a collection of hardware drivers. AIR sends action commands to HServer, and HServer relays the commands to the robot hardware. HServer sends perceptual signals, captured by the robot sensors, back to AIR as well. AIR and HServer run as independent processes. While HServer, written in C++, runs only on Linux operating system, Java-based AIR can run on any operating system that supports a Java virtual machine (JVM). The communication between AIR and HServer is done through an internet socket connection, enabling AIR and HServer to exist on separate machines. Three types of configurations are implemented and used in the experiments described in Chapter 5: namely, real robot, Gazebo, and USARSim configurations. The details of these configurations are described in the following subsections.

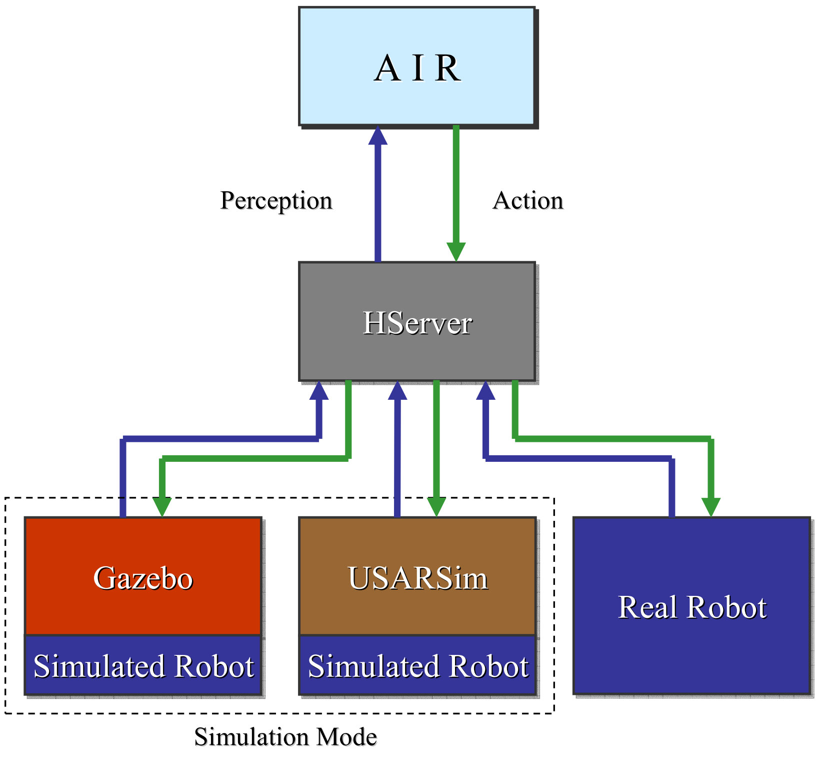


Figure 37: Integration of AIR (Figure 28) into a complete robotic framework.

4.2.1 Real Robot Configuration

A picture of the hardware used in this configuration is shown in Figure 38.

ActivMedia's Pioneer 2 DX robot was integrated with a Sony EVI Camera. The data flow among AIR, HServer, and the robot/sensor is shown in Figure 39. More specifically, in this configuration, HServer (running on a Dell Dimension 4700 with Intel Pentium 4; 3.00 GHz; Red Hat Enterprise Linux WS 4) receives readings from sixteen sonar sensor readings embedded on the Pioneer 2 DX robot and sends back the control commands via a wireless serial connection (FreeWave Data Transeiver; 900 MHz). A stream of analog video images from the camera also arrives at HServer via a wireless video connection (RadioShack A/V Signal Sender; 2.4 GHz) and a frame grabber (Hauppauge WinTV GO).

From the digitized video images, HServer was configured to detect objects in the environment using a classifier [99] implemented in Intel's open source computer vision library (OpenCV) [2]. Specifically for the second experiment[12] in Chapter 5 (Section 5.2), the classifier was trained to detect four types of predefined objects (a baby doll and three signs). From 250 to 500 training images for each object were used to train the classifier. Sample results of the object detection are shown in Figure 40.

From HServer, the sonar sensor readings and detected objects are delivered to AIR (running on Lenovo ThinkPad T61 with Intel Core 2 Duo; 2.0 GHz; Windows XP) and the motor commands computed by AIR are sent back to HServer via a TCP/IP socket connection. This configuration was used in the second experiment in Chapter 5 (Section 5.2).

[12] No other experiments utilized the object detection capability.

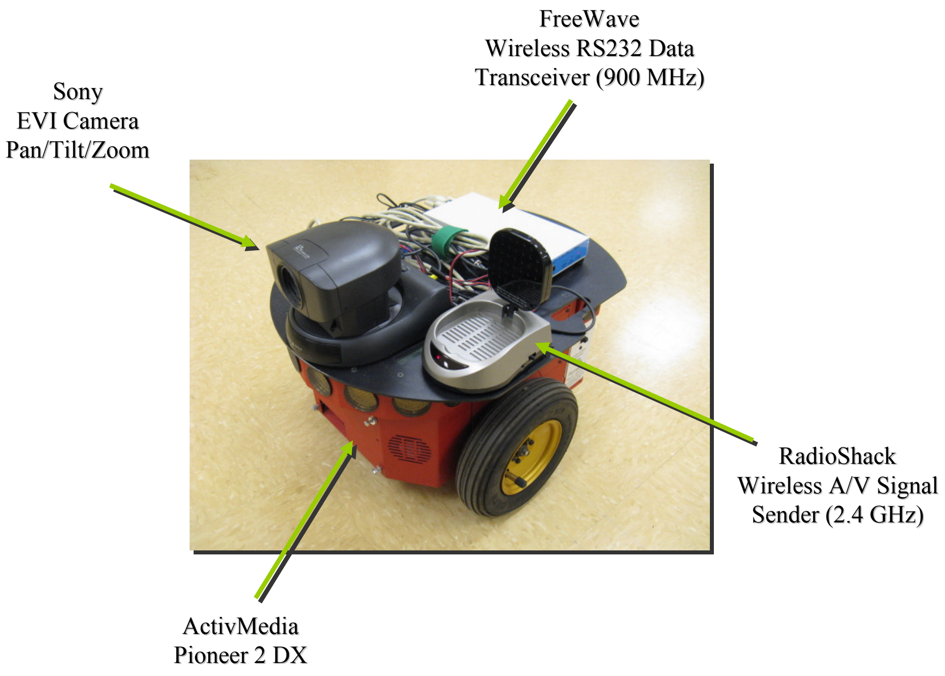


Figure 38: The real robot hardware configuration.

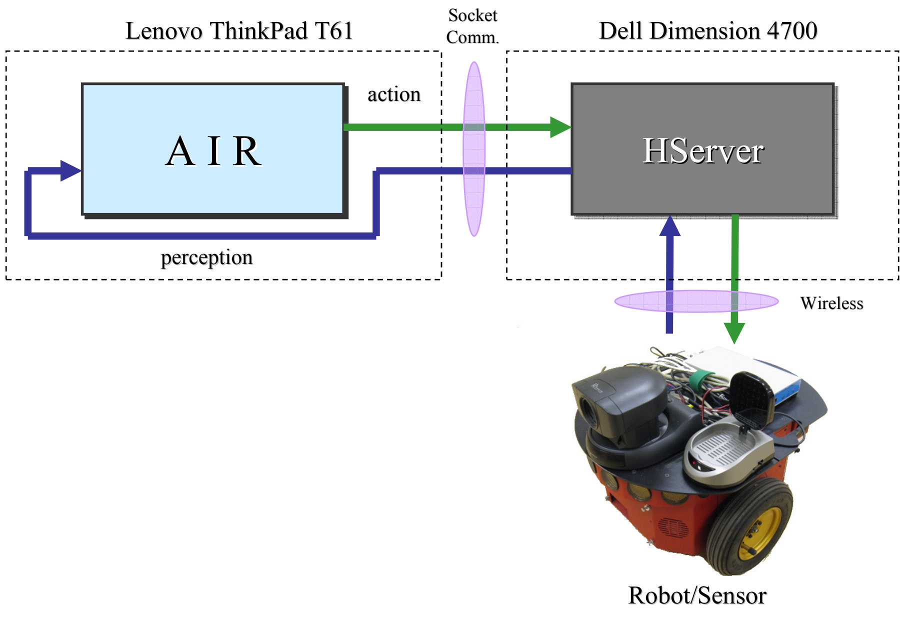


Figure 39: The data flow among AIR, HServer, and the robot/sensor.

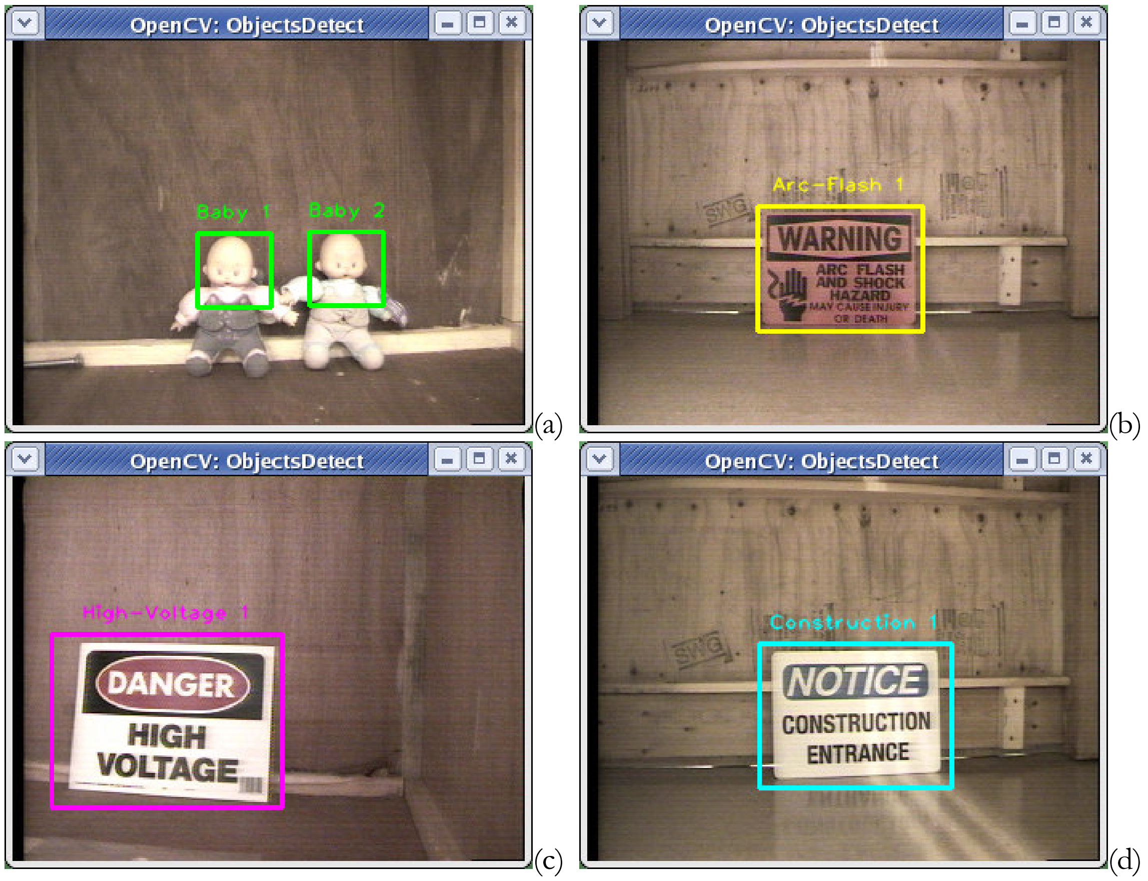


Figure 40: Four images showing objects detected by the OpenCV classifier implemented within HServer: (a) two baby-dolls, (b) the "Arc Flash and Shock Hazard" warning sign, (c) the "High Voltage" danger sign, and (c) the "Construction Entrance" sign.

4.2.2 Gazebo Configuration

In this configuration, AIR was set up to interact with a virtual robot simulated in University of Southern California's Gazebo [87], a high fidelity three-dimensional simulator. More specifically, based on an integrated dynamics engine [164], the dynamics of a simulated robot is constantly computed to reflect the motor commands being received and the type of the virtual environment it is currently in. In this configuration, ActiveMedia Pioneer 2 DX and its sixteen sonar sensors are simulated.

As shown in Figure 41, Gazebo and HServer run on a same desktop machine (Dell Dimension 4700 with Pentium 4; 3.00 GHz; Red Hat Enterprise Linux WS 4), and they interact with each other through a shared memory connection. Relayed by HServer, AIR (running on Dell Latitude X200 with Pentium III; 933 MHz; Red Hat Linux Fedora Core 4) receives the emulated sonar sensor readings from Gazebo and sends back the motor commands. AIR and HServer are connected through a TCP/IP socket connection. This configuration was used in the first and third experiments described in Chapter 5.

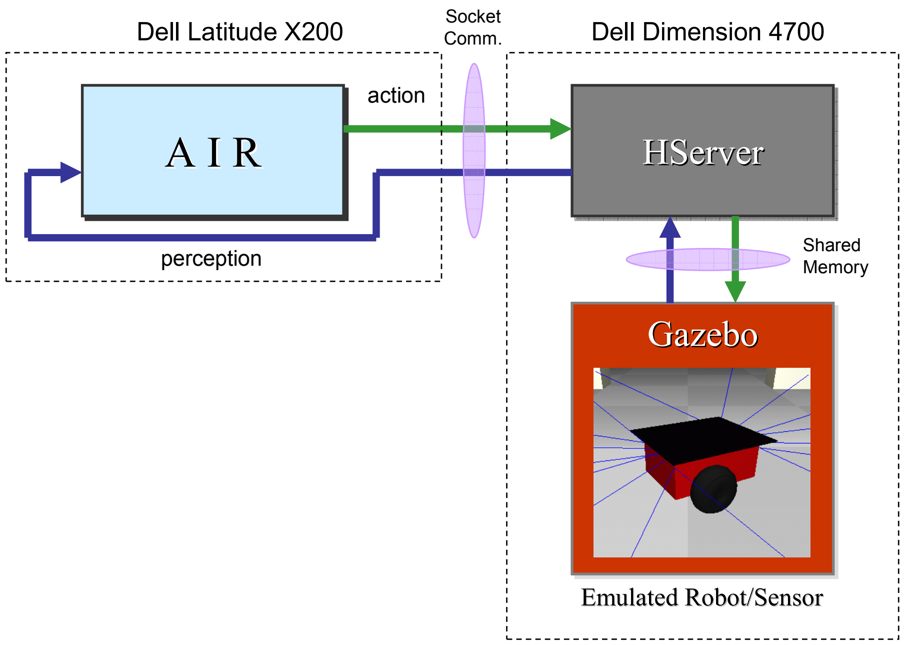


Figure 41: The data flow among AIR, HServer, and the emulated robot/sensor in Gazebo.

4.2.3 USARSim Configuration

While Gazebo is a widely recognized 3D simulator in the robotics community, no adequate toolkit is currently available to design the (virtual) environment that can be simulated; hence, the complexity of the environments is rather limited in Gazebo. On the other hand, USARSim [43], another high fidelity 3D simulator developed by the National Institute of Standards and Technology (NIST), comes with a toolkit to design a fairly complex 3D environment as it is based on Epic Games' 3D gaming technology [38]. USARSim was developed by NIST to simulate search-and-rescue robots in realistic environments. Thus, along with Gazebo, AIR was also interfaced with USARSim.

In the USARSim configuration, iRobot's ATRV-Jr, integrated with SICK LMS200 laser scanner, was simulated. As shown in Figure 42, USARSim runs on a laptop (Lenovo ThinkPad T61 with Intel Core 2 Duo; 2.0 GHz; Windows XP) and communicates with HServer running on a desktop machine (Dell Dimension 4700 with Pentium 4; 3.00 GHz; Red Hat Enterprise Linux WS 4) through a TCP/IP socket connection. HServer and AIR, running on the same desktop machine, communicates with each other through a TCP/IP socket connection as well. This configuration was used in the third experiment (Section 5.3).

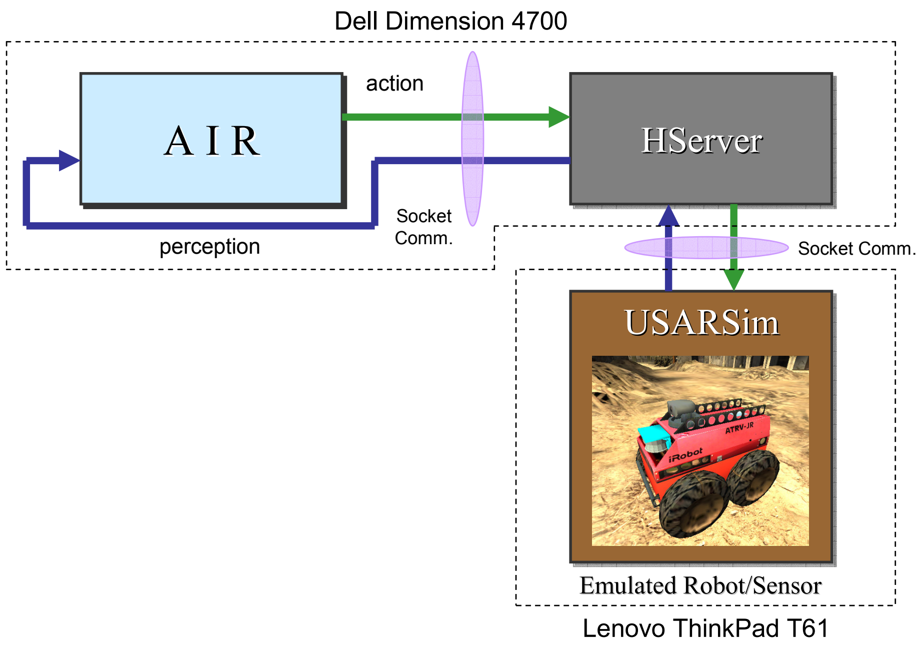


Figure 42: The data flow among AIR, HServer, and the emulated robot/sensor in USARSim.

CHAPTER 5

EVALUATION

In this chapter, the computational model of proactive intelligent behavior for robots discussed in Chapter 3 is evaluated through a set of three experiments. As summarized in Table 31, the first experiment determines the efficiency of the foundational data structure. The second experiment elucidates the characteristics of proactive behavior with respect to somatic markers. Finally, the tradeoff between promptness of the proactive behavior computation and the quality of the behavior is examined in the third experiment.

Table 31: A summary of the experiments in Chapter 5.

Section	5.1	5.2	5.3	
Evaluation Focus	Efficiency of the foundational data structure	Characteristics of proactive behavior with respect to somatic markers	Tradeoff between promptness of the proactive behavior computation and the quality of the behavior	
Relevant Subsidiary Research Question (Section 1.2.2)	3rd	4th	5th	
Experiment #	1	2	3A	3B
Proactiveness	Anticipation	Anticipation	Anticipation	Improvisation
Configuration	Gazebo	Real	Gazebo	USARSim
Environment	Indoor	Indoor	Indoor	Outdoor
Scenario	Simple Hallway Navigation	Search-and-Rescue	Simple Hallway Navigation	Reconnaissance and Detour
Results	The data structure indeed helps reduce the localization time.	The somatic markers help the robot make advantageous decisions.	The history length can be trimmed to improve promptness without compromising behavioral qualities.	The prompt localization time is crucial for sustaining behavioral qualities.

5.1 Efficiency of the Foundational Data Structure

Recall from Chapter 1 that the third subsidiary research question (Section 1.2.2) seeks to determine how information should be stored in episodic memory:

- *How should past episodes of experience be organized in the memory of a robot in order for them to be* *utilized upon anticipation and improvisation?*

To address this question, the efficiency of the organizational structure to encode the episodic memory (Section 3.1) was evaluated in this experiment. The *efficiency* in this case refers to the computational efficiency upon computing proactive behavior. As discussed in Chapter 1, the applications that require proactiveness (e.g., military, search-and-rescue, etc.) can be very time-sensitive. Regardless of how large the state space becomes, behavioral computation has to be fast enough for a robot to be practical in such situations. In other words, the larger the state space it can process in a given time, the more complex task the robot can effectively handle. Thus, computational efficiency is considered a key metric to evaluate the organizational structure of the episodic memory as opposed to, for example, a required memory size since it is likely solved by additional hardware.

The most computationally expensive step in the proactive behavior computation is the event-matching process (Section 3.2.2). In particular, if implemented naively, the recursive Bayesian filter (Equation 3.16) requires an $O(n^2)$ computation time to compute the full posterior probabilities for every episode comprised with n events because the transition model has to be computed n times for each of the n events (states). Incidentally, localization with respect to a map using a Kalman filter (also Bayesian) requires an $O(n^2)$ computation time [179].

As discussed in Sections 3.1.1 and 3.2.2, however, because the events stored in the episodic memory are organized in a unidirectional temporal linear-chain fashion, a certain assumption can be made to reduce the computational load. More specifically, to implement the transition model, the transition probability between two events is approximated by the Poisson probability density function, computing the probability based on the distance between the two events (Equation 3.19). Furthermore, if the distance between the two events becomes greater than a five-event length, the probability becomes essentially zero (recall Figure 26). Exploiting this property, in this experiment, we empirically show that the amount of time required to compute the event-matching process for each episode can be reduced from $O(n^2)$ to $O(n)$.

5.1.1 Materials and Methods

The computation time required for the event-matching process while exploiting the property of the Poisson distribution (*limited-transitions* case) is compared against the computation time when the property is not exploited (*full-transitions* case). The *computation time* here refers to the time required to compute the event-matching function (Equation 3.16) that is the posterior probability of being at some event in a past episode given the history of the current sequence of observations and executed behaviors. At every time cycle, for each event in an episode, the time to compute the event-matching function were recorded in AIR and averaged over all of the events in the episode. For the limited-transitions case, the transition model in Equation 3.16 was computed for only five relevant events. As mentioned above, since the events in an episode are formed in a unidirectional linear chain, our claim here is that, if the property of the Poisson distribution is exploited, the event-matching process can be computed in an $O(n)$ time.

The Gazebo configuration described in Section 4.2.2 was used in this experiment. In other words, AIR, the Java-based computer program that implements the proactive behavior computation running on a laptop machine (Intel Pentium III), was set up to interact with a virtual robot (Pioneer 2 DX) simulated in the high fidelity 3D simulator, Gazebo, running on a desktop machine (Intel Pentium 4). The virtual indoor environment used in this experiment is shown in Figure 43. Note that, given a fixed number of integrated sensors, the time to compute the event-matching function (Equation 3.16) solely depends on the number of events in an episode. In other words, regardless of environmental types, the time to compute the event-matching process is the same as long as the number of events in an episode is unchanged. Thus, the results gained from this indoor experiment are expected to be generally applicable for other types of environments.

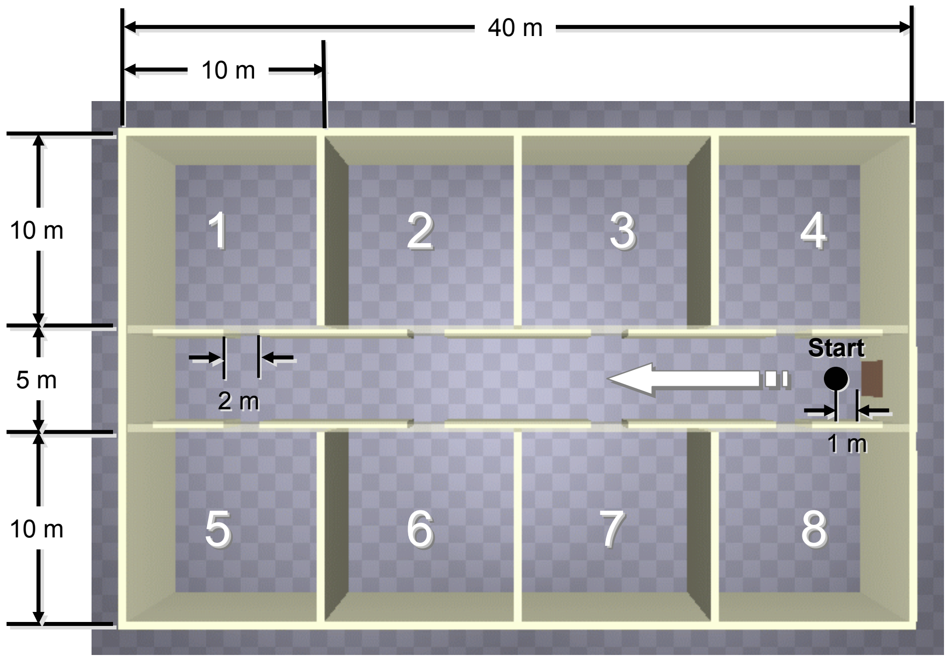


Figure 43: The simulated indoor environment used in the first experiment.

The robot was equipped with 16 sonar sensors, but no odometry information was ever used for the proactive behavior computation. AIR computed the anticipatory behavior based on a single training episode stored in the memory. The training episode was generated by manually instantiating a combination of the *Avoid-Obstacle*, *Move-Forward,* and *Swirl-Obstacle* motor schemata (Section 4.1.3). In this experiment, manual instantiation of behavior is considered adequate since behavioral types do not influence the time to compute the event-matching function[13] (Equation 3.16).

For each case, the size of the training episode in the memory was varied from 20 events to 200 events with the increment of 10 events (i.e., 19 different sizes). For each condition, the testing lasted 10 event-matching cycles, and it was repeated 20 times. Hence, the computation time of the each data point was averaged over 200 measurements. Some of the predefined constants utilized during the computation in this experiment are summarized in Table 37, Appendix C (Section C.1).

5.1.2 Results

The average computation time (over 200 measurements) of each condition with respect to the number of the events in the episode is plotted in Figure 44. The numerical results are shown in Appendix C (Section C.2.1). More specifically, Plot 1 shows the overall computation time for the event-matching process with respect to the number of events in an episode when the property of the Poisson distribution was exploited (i.e., the limited-transitions case). Plots 2 and 3 are also for the limited-transition case, showing the time required to compute the sensor model and transition model, respectively. Plots 4, 5, and 6 show the time required to compute the overall event-matching process, sensor model, transition model, respectively. Each of these plots was also fitted with trend lines using Microsoft Excel. Plots 1, 2, 3, and 5 were best fitted by linear trend-lines while Plot 4 and 6 were best fitted by second-order polynomial trend-lines.

In other words, as expected, when all of the possible transitions were taken into account upon computing the transition model, the computation time increased quadratically

[13] An arbitrary reward value (1.0) was also manually assigned at the end of the episode although its influence on the computation time was null as well.

with respect to the number of events (states). When the computation was broken down to the sensor model and transition model parts, the transition model computation did indeed exhibit the quadratic increase while the increase of the sensor model computation remained linear. On the other hand, in the limited-transitions case, the overall event-matching time was increased only linearly with respect to the number of events, consistent with the $O(n)$ claim. The computations for both sensor and transition models were also expectedly linear.

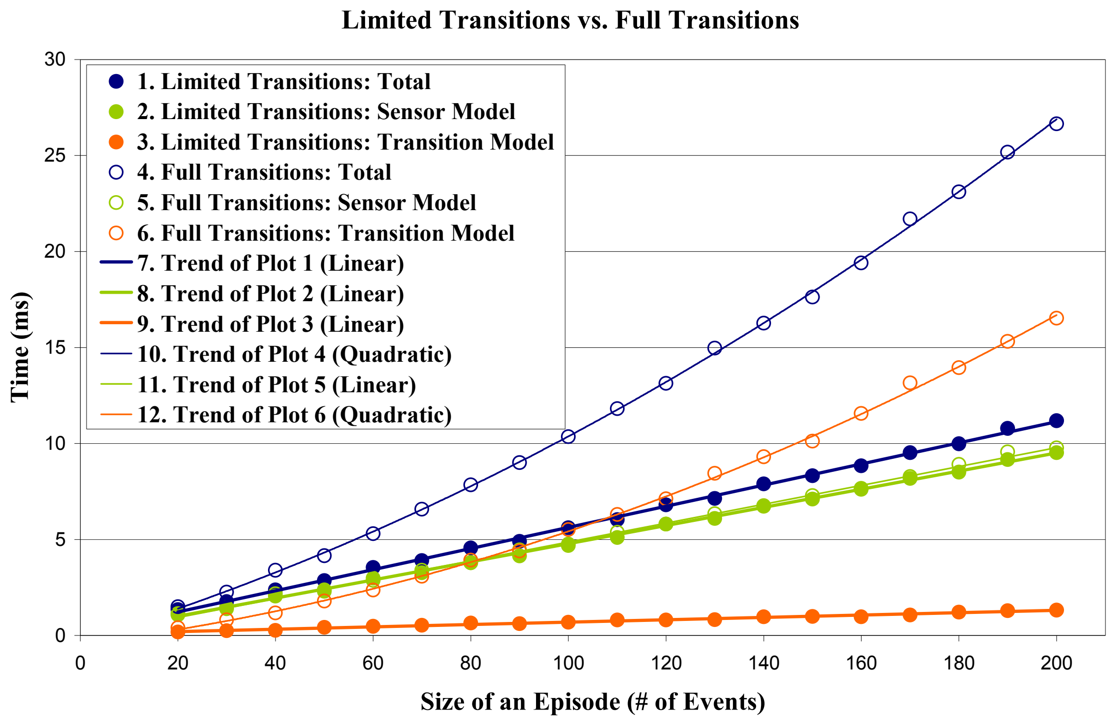


Figure 44: The average computation time required for the event-matching process with respect to the number of events in an episode. (See Table 38 in Appendix C for the numerical values. Also in Appendix C, the constants and the correlation coefficients of the trendlines are shown in Table 39, and the standard error measurements (too small to display here) are reported in Table 40.)

5.1.3 Discussion

This experiment demonstrated the time required for the event-matching process can be indeed reduced from $O(n^2)$ to $O(n)$ if the property of the Poisson distribution is exploited in the transition model computation. As discussed in Section 3.4.2, a combination of the event-matching and behavioral-selection processes is the equivalent of a partially observable Markov decision process (Section 2.4.4). In general, the state estimation part of POMDP, which is the equivalent of the event-matching process in our approach, is the most computationally expensive part. Since the event-matching process is reduced to $O(n)$, our method does indeed computes a POMDP solution efficiently in practice.

5.2 Effectiveness of Somatic Markers

Recall the fourth subsidiary research question (Section 1.2.2) concerning the somatic marker hypothesis (Section 2.1.2):

- *Does a memory with integrated somatic markers help a robot achieve better anticipation and/or improvisation than without them?*

In order to determine this question, the effectiveness of somatic markers integrated within the episodic memory is evaluated. As reviewed in Section 2.1.2, the somatic marker hypothesis [45] asserts that an animal's internal emotional responses to environmental stimuli are physically embedded into the episodic memory and such an element (marker) in the memory is later utilized in the decision making process, allowing the animal to select the most advantageous option. This experiment was adapted from the gambling experiment conducted by Bechara et al. [23] in which the somatic marker hypothesis was examined using patients whose ventromedial prefrontal cortices (presumably a part of the somatic marker circuitry) had been damaged. Although reviewed in Section 3.1, we revisit their experiment in more detail in order to contrast their work with ours.

In their experiment [23], the subjects were asked to draw cards from four types of decks. As summarized in Table 32, two of the four types (A and B) rewarded the subjects with \$100 per card, but the occasional punishments were arranged in a way that after drawing ten cards from these types, the subject would lose $250. On the other hand, the other two decks (C and D) rewarded the subjects with only $50 per card, but it was set up in a way that they would eventually gain $250 after drawing ten cards from these types of the decks.

Table 32: A summary of reward arrangements for the gambling experiment conducted by Bechara et al. [23].

Card Type	A	B	C	D
Reward Value	$100	$100	$50	$50
Avg. Punishment Value	-$250	-$1,250	-$50	-$250
Avg. Frequency of Punishment	1 in 2 cards	1 in 10 cards	1 in 2 cards	1 in 10 cards
Net Gain Per 10 Cards	-$250	-$250	$250	$250

The graphs in Figure 45 show the average distributions of 100 cards picked by the subjects over the four types of decks. The right graph ("Target Subjects") shows the distribution for the subjects whose ventromedial prefrontal cortices were damaged, and the left graph ("Normal Control") is for the subjects with out the damage. According to Bechara et al. [23], the difference between these groups was substantial in terms of selecting the types of the cards. The graph in Figure 46 highlights the difference between the two groups in terms of their advantageous (decks A and B) vs. disadvantageous (decks C and D) choices. More specifically, in 100 trials, the number of times each subject selected from the disadvantageous decks was subtracted from the number of time he/she selected from the advantageous decks, and the value was averaged over all the subjects in the group. In other words, having a positive graph indicates an advantageous trend. In this graph (Figure 46), attention should be paid on the groups labeled "EVR-Type" and "Normal"; the former is the subjects with the ventromedial prefrontal context damages (i.e., broken somatic marker circuitry) and the latter is control subjects without the damage[14]. According to Bechara et al. [23], the subjects without the somatic marker circuit damage were found to make advantageous decisions profoundly compared to the subjects with the damage. In other words, the somatic marker circuitry seems to make a significant contribution upon making advantageous decisions.

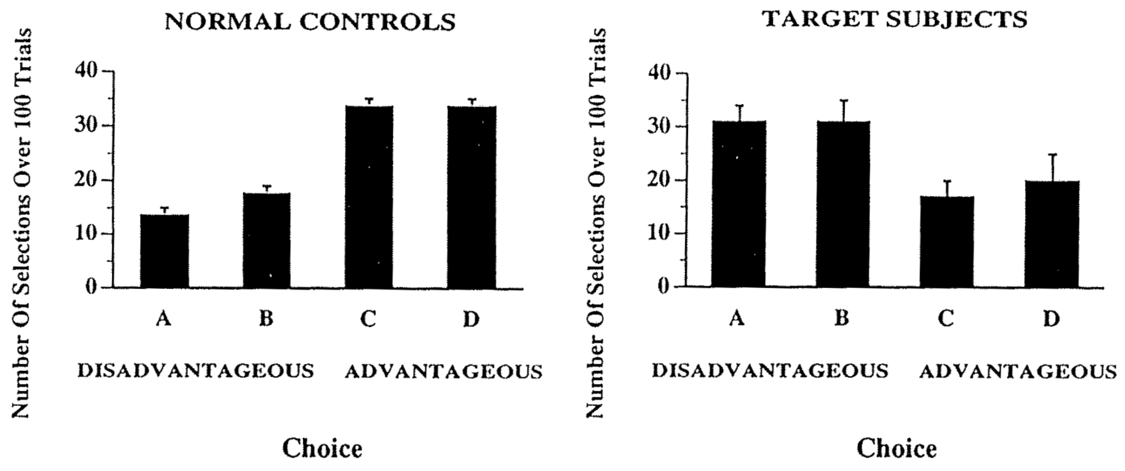


Figure 45: The difference between the group with the somatic-marker circuital damages ("Target Group") and the group without the damage ("Normal Controls") in terms of their choices of cards. The vertical error bars indicate the values of standard error measurements. (Diagram reproduced from [23].)

[14] Regarding the other two groups, the "Brain-Damaged" group is the subjects with other types of brain damages, and the "EVR" group is one stereotypical subject ("Patient E.V.R.") whose ventromedial prefrontal cortex was damaged [23].

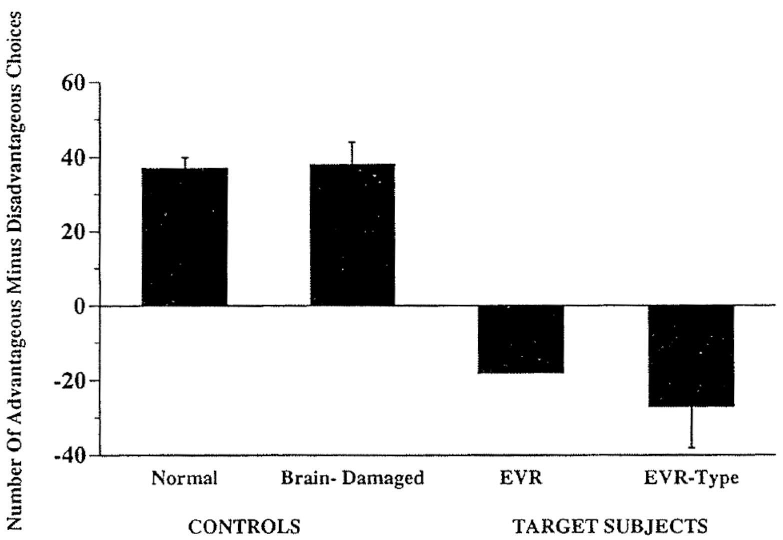


Figure 46: The difference between the group with the somatic-marker circuital damages ("EVR-Type") and the group without the damage ("Normal") in terms of their advantageous choices minus disadvantageous ones. (See Footnote 14 for the descriptions of other two groups.) The vertical error bars indicate the values of standard error measurements (Diagram reproduced from [23].)

5.2.1 Materials and Methods

Recall that, in our framework, a somatic marker refers to the reward signal saved in a sampled event whose value was predominantly determined by the associative rewarding states (Section 3.3.2), and it is utilized in the behavior-selection process by providing expected utilities of to-be-executed behaviors (Section 3.2.3). More specifically, recall the following equation (a copy of Equation 3.35), which is the reward function that determines the current reward value:

$$r_{\text{cur}} = \kappa_g f_{\text{L}}(o_{\text{cur}}, g_{\text{cur}}) + \kappa_o \max_{E \in \Gamma^+(b^*)} f_{\text{L}}(o_{\text{cur}}, o'_{[E]}) + \sum_{\iota \in \text{I}} \kappa_\iota f_{\text{L}}(o_{\text{cur}}, \iota)$$

Here, κ_g, κ_o, and κ_ι are the constants (weights) for the three terms of this equation: namely, similarities of the current perceptual state (o_{cur}) with respect to: 1) the current goal (g_{cur}), 2) predicted observation ($o'_{[E]}$), and 3) associative rewarding states (ι), respectively. Each similarity is calculated by the likelihood function, f_{L} (Equation 3.14). If the value of the third term (the associative rewarding states) becomes substantially larger than other terms, the subsequent reward signal is considered a somatic marker when saved in a sampled event.

The somatic-marker gambling experiment [23] explained above was adapted for a robotic search-and-rescue scenario in this experiment. The real robot configuration (Section 4.2.1) was used. More specifically, ActivMedia's Pioneer 2 DX robot with sixteen sonar sensors and a camera was controlled by the behavior computed by AIR running on a laptop machine (Intel Core 2 Duo). The communication between the robot and AIR was mediated by HServer, running on a desktop machine (Intel Pentium 4).

The experimental area, set up in Georgia Tech Mobile Robot Laboratory, is shown in Figure 47. Instead of the four decks of cards in the gambling experiment, four crate boxes were placed at four different locations in the area (labeled "Box SE", "Box SW", "Box NE", and "Box NW"). The contents of the boxes were hidden from the robot's initial view (marked "Start"). Thus, to inspect inside them, the robot would have to travel to the locations while avoiding obstacles (shaded gray in the figure). The obstacles were detected by the sonar sensors. No odometry information was ever used in this experiment.

During the test, as each card in the gambling experiment was denoted with a certain amount of money to indicate rewards and punishments, each box contained detectable objects that had association with certain reward/punishment values. More specially, as shown in Table 33, if HServer (OpenCV) detected a baby-doll ("survivor") in the incoming video image, 50 points were rewarded (the equivalent of the \$50 low rewarding card). If two baby-dolls were detected simultaneously, 100 points were rewarded (the equivalent of the \$100 high rewarding card). In our framework, awarding of the points was done by assigning appropriate weights (κ_l) for the third term of the reward function (above). In other words, considering the perceptual states of simply detecting these objects to be the associative rewarding states in this case, the values of κ_l for detecting a baby-doll and two baby-dolls were assigned to be 50.0 and 100.0, respectively (assuming the detection does not interfere with each other). Similarly, for punishments, the values of κ_i for detecting the "arc flash and shock hazard" signs, the "high voltage" danger sign, and the "construction entrance" sign were assigned to be -1250.0, -250.0, and -50.0, respectively. The values of these punishments were chosen, so that the robot can be punished with the exactly same frequencies applied in the gambling experiment (Table 32). The frequencies of the punishments used in this experiment are shown in Table 34. For example, if the robot visits the Type A box, the robot is awarded 100 points (two baby-dolls) although once in two visits, -250 points of a punishment (the "high voltage" danger sign) is also hidden in the box. As in the gambling experiment, the rewards and punishments in the Type A and Type B boxes were arranged, so that the robot would experience a net-loss of -250 points if these boxes have been visited 10 time consecutively. On the other hand, if the robot visits the Type C and Type D boxes 10 times in a row, the robot would experience a net-gain of 250 points (i.e., more advantageous choices than the other two boxes). Note that we replicated the experimental setup of the somatic-marker gambling experiment by Bechara et al. [23]; as they had effectively shown the utility of somatic markers, we assumed that the utility of the robotic somatic marker can be also assessed in a similar way.

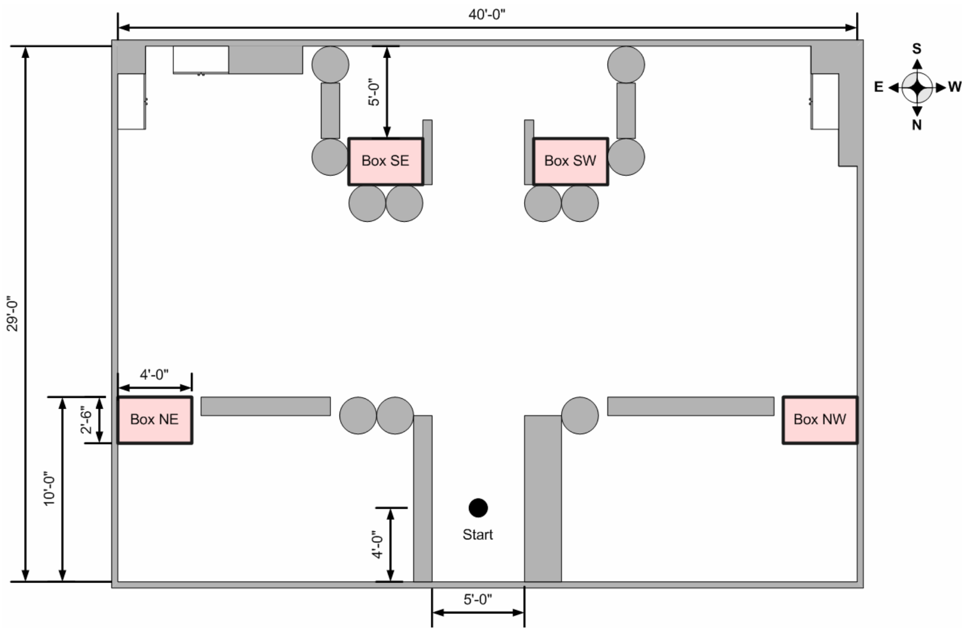


Figure 47: An experimental area used in the somatic marker experiment using a real robot.

Table 33: Assigned associative reward values of predefined objects.

Object Image					
Description	Two Baby Dolls	One Baby Doll	"Arc Flash and Shock Hazard" Warning Sign	"High Voltage" Danger Sign	"Construction Entrance" Notice Sign
Associative Reward Value	100 pts	50 pts	-1250 pts	-250 pts	-50 pts

Table 34: A summary of reward arrangements for the somatic marker experiment (cf. Table 32).

Box Type	A	B	C	D
Reward Value	100 pts	100 pts	50 pts	50 pts
Punishment Value	-250 pts	-1250 pts	-50 pts	-$250
Frequency of Punishment	1 in 2 visits	1 in 10 visits	1 in 2 visits	1 in 10 visits
Net Gain Per 10 visits	-250 pts	-250 pts	250 pts	250 pts

In this experiment, the performance of the robot with somatic markers was compared against the performance of the robot without somatic markers. The latter case was implemented by setting all the associative reward values (κ_i) to be zero. Incidentally, κ_g, the weight for the first term (similarity between the current perceptual state and the current goal) in the reward function, was set to be 25.0 (a half of the weight for the baby doll). Hence, if the somatic marker is enabled, the associative rewarding state would have significant impact on the current reward value. The second weight (κ_o), the similarity between the current perceptual state and predicted perceptual state, was set to be zero, so that it would have no effect during the experiment.

Before testing, for each of the four boxes, three training episodes were generated by manually instantiating a combination of the *Move-Forward*, *Move-Leftward*, *Move-Rightward*, and *Swirl-Obstacle* motor schemata (Section 4.1.3) to make the robot reach the box from the starting position. Although the manual instantiation (supervising) of the behavior may have produced suboptimal solution to reach the destination and may have contained variances among them, by altering the locations of the Types A, B, C, or D boxes through out eight trials as shown in Table 35, the effect of the sub-optimality and the variances was considered nullified. The goal during the testing episodes was set to be the perceptual state detecting two survivors even though no object was hidden in any of the boxes during the training period (i.e., the current reward value was constantly zero during the training).

During a testing episode, the robot was dispatched from the same starting position as in the training, and the same goal (detection of two survivors) was activated by the goal manager. The activation was done manually through the graphical user interface (recall Figure 31) in order to effectively test the proactive behavior computation (instead of, for example, the motivation function to select the goal automatically). Twenty consecutive episodes were recorded as a single test set, and eight test sets were collected for each condition: the robot with somatic markers vs. the robot without somatic markers (i.e., 320 runs total). No improvisation was performed in this experiment. Some of the predefined constants utilized during the computation in this experiment are summarized in Table 37, Appendix C (Section C.1).

Table 35: The reward/punishment schedule types (Table 34) and corresponding box locations (see Figure 47) for each test set.

Test Set Number	Box Type			
	A	B	C	D
1	Box SE	Box NE	Box SW	Box NW
2	Box NW	Box SW	Box NE	Box SE
3	Box SW	Box NW	Box SE	Box NE
4	Box NE	Box SE	Box NW	Box SW
5	Box SE	Box SW	Box NE	Box NW
6	Box SW	Box SE	Box NW	Box NE
7	Box NE	Box NW	Box SE	Box SW
8	Box NW	Box NE	Box SW	Box SE

5.2.2 Results

The graph in Figure 48 shows the rate of the robot taking advantageous choices (i.e., Boxes C and D) through out the 20 trials. The rate was averaged over the eight test sets, and the two conditions (the robot with somatic markers vs. the robot without somatic markers) were compared. The actual sequences of the robot's 20 visits to the boxes are shown in Table 41 (with somatic markers) in Table 42 (without somatic markers) in Appendix C. As it can be observed in the graph, although the difference between the two conditions was not substantial in the beginning, the trajectories of the two gradually diverged as the number of the trials was increased. The robot with somatic markers started taking more and more advantageous choices as episodes were accumulated while the robot without somatic markers was consistently taking disadvantageous choices towards the end. Note that some minimum exposure to the environment (about 11 visits in this case) seems to be necessary before somatic markers become truly effective.

The average numbers of the robot's visits made to each box over the 20 trials are plotted in Figure 49 (with somatic markers) and Figure 50 (without somatic markers). Here, the performance of the robot with somatic markers was found to be substantially better than the one without somatic markers as it visited more advantageous boxes than disadvantageous ones. According to a factorial analysis of variance (ANOVA), presence of somatic markers indeed had significant effect on the box choices ($F(3,56) = 7.03, p < 0.001$). Note that these results are similar to the results found in the actual somatic marker gambling experiment conducted by Bechara et al. [23] (Figure 45) as they both have shown that the subjects/robots with somatic markers make more advantageous choices than the ones without somatic markers. To further visually compare our results with the ones reported by Bechara et al. [23], the number of disadvantageous choices (Types A and B) made by the robot was subtracted from advantageous choices (Types C and D) and plotted in Figure 51, generating a graph compatible to the one in Figure 46. In this case, a positive bar infers an advantageous trend in its choices. According to a one-way ANOVA, the difference between the robot with somatic markers and the robot without them was substantial ($F(1, 14) = 66.98, p < 0.001$); the robot with somatic markers overwhelmingly made more advantageous choices.

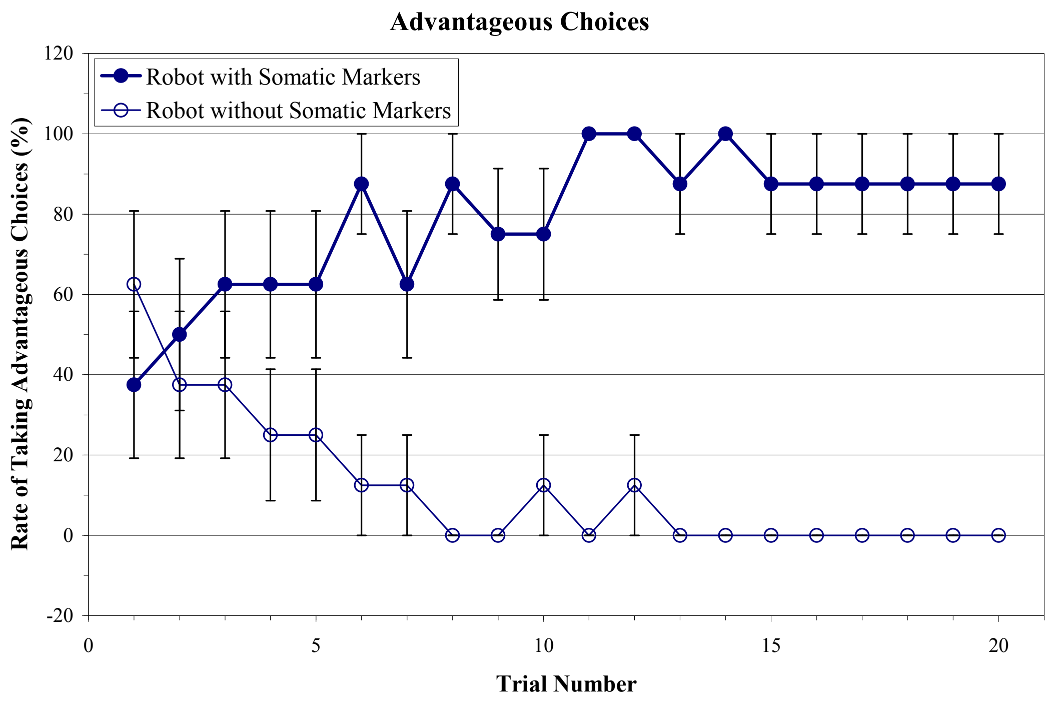


Figure 48: The rate of the robot taking advantageous choices (Boxes C and D) with respect to the number of trials. While there was no substantial distinction between the two conditions in the beginning, at the end of the trials, the robot with somatic markers was found to choose advantageous choices. On the other hand, the robot without somatic markers was found to choose disadvantageous choices. The vertical error bars indicate the values of standard error measurements. (See Table 43 in Appendix C for the numerical values.)

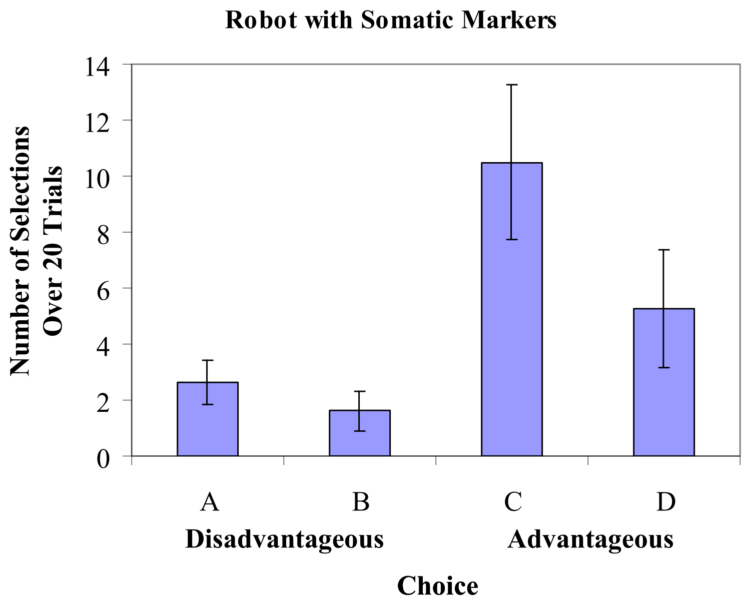


Figure 49: The average distribution of 20 consecutive box-visits over four box types (A, B, C, and D) by the robot with somatic markers. The vertical error bars indicate the values of standard error measurements. (See Table 44 in Appendix C for the numerical values.)

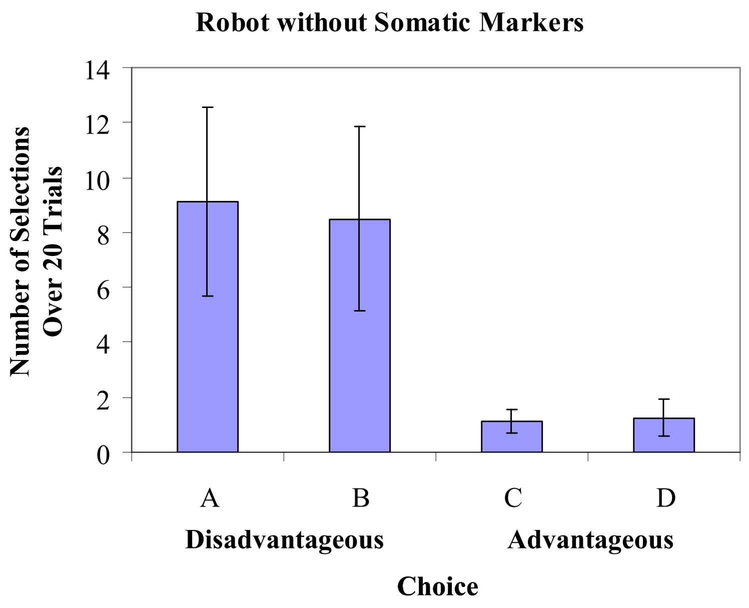


Figure 50: The average distribution of 20 consecutive box-visits over four box types (A, B, C, and D) by the robot without somatic markers. The vertical error bars indicate the values of standard error measurements. (See Figure 43 in Appendix C for the numerical values.)

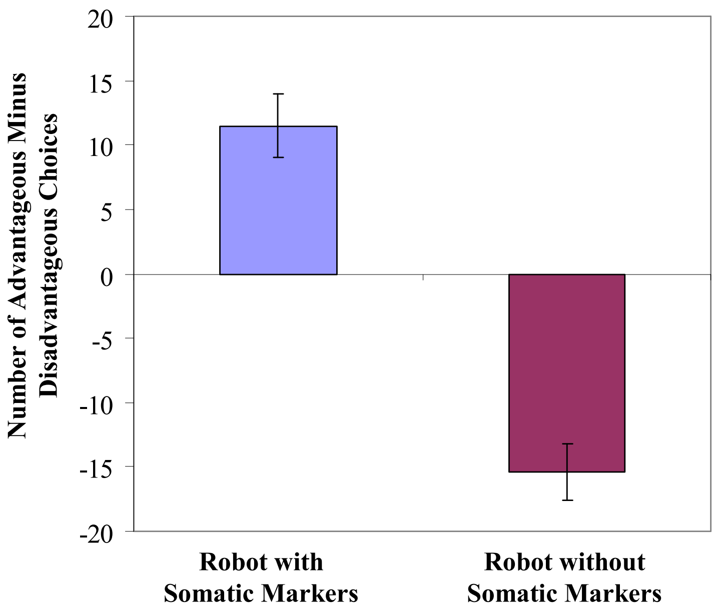


Figure 51: The difference between the robot with somatic markers and the robot without them in terms of its advantageous choices minus disadvantageous ones. The vertical error bars indicate the values of standard error measurements. (See Table 46 in Appendix C for the numerical values.)

5.2.3 Discussion

This experiment demonstrated that the robot with somatic markers indeed made advantageous decisions for itself compared to the one without somatic markers. Algorithmically, the difference between the two conditions is the presence of associative rewarding values (the values were all zeros for the robot without somatic markers). Since the weight (κ_g) in Equation 3.35 for rewarding the robot reaching a current goal (finding two "survivors") had a non-zero value, once two baby dolls were detected in one episode, the robot without somatic markers tended to choose actions based on that episode, resulting in the disadvantageous choices for the robot in this experimental setup. However, if the world is static and hence the punishments never appeared, since it consistently went to the boxes where its goal was, the robot without somatic markers would have made right choices perfectly. Thus, even if it may occasionally lead the robot into disadvantageous circumstances (such as in this experimental setup), the value of κ_g should be kept non-zero.

Although the scenarios differed, this experiment and the gambling experiment conducted by Bechara et al. [23] had similar results as the decisions made by the robot or the subjects with somatic markers were found to be more advantageous than those without somatic markers. It should be noted however that this does not necessary indicate that our approach is exactly how our brains work. This experiment rather proves that a biologically inspired approach can be incorporated into standard machine learning techniques to solve typical artificial intelligence problems such as making advantageous decisions in the long run. Of course, being able to make advantageous decisions in the long run is not novel in the field of machine learning or artificial intelligence. The contribution of this approach can be highlighted by particularly combining the results from the previous experiment: computational efficiency during the event-matching process. Since this experiment has demonstrated that the MDP part (behavioral selection) of the partially observable Markov decision process or POMDP (see Section 2.4.4) does work, it can be concluded that our method indeed computes POMDP solutions computationally efficiently.

5.3 Promptness of Proactive Behavior Computation

This experiment addresses the issues raised by the fifth subsidiary research question (Section 1.2.2) that is to understand the relationship between promptness of the proactive behavior computation and its behavioral quality:

- *What is the trade-off between promptness and the quality of anticipation/improvisation that a robot performs?*

This experiment consists of two parts; the first part attempts to solve it from an anticipatory perspective while the second part deals with the same problem from an improvisational perspective.

In terms of the anticipatory aspect, the first experiment in Section 5.1 demonstrated that the event-matching time in the proactive behavior computation can be reduced to $O(n)$ where n is the number of events in an episode. Even so, recall that if k episodes are found relevant by the recollection process (Section 3.2.1), the total amount of the event-matching time has to be multiplied by k. Naturally, if the robot increases the experience, the k value also increases, resulting the computational time to be $O(kn)$ instead of $O(n)$. As discussed in Section 3.2.1, we proposed that the size of k can be restricted by setting its upper limit or cap. If true, the computation time should remain $O(n)$ instead of $O(kn)$. Although, the factor, k, may seem small as it is a mere linear multiplier (i.e., not polynomial), as our primary interest in this dissertation is to determine how *extended* experience influences the proactive behavioral computation (Section 1.2), it should not be regarded as trivial. Therefore, in the first part of this experiment, we examine how k influences the time required for the event-matching process as well as how it influences the quality of the anticipatory behavior. Furthermore, we also investigate whether imposing the cap can have a negative effect on the computation time or the behavioral performance.

The second part of the experiment deals with an improvisational aspect of the computational promptness with respect to the behavioral quality. In particular, we examine how the number of episodes in the memory influences the time to respond to the unanticipated circumstance (i.e., validation failure) and recall a new set of relevant episodes for a new (intermediate) goal. As in the anticipatory experiment, the event-matching time with respect to the quality of improvisational behavior is also examined.

5.3.1 Materials and Methods

Part A (Anticipatory Aspect)

In this part of the experiment, we examine how the number of relevant episodes selected by the recollection process influences the time to compute event matching as well as how it influences the quality of the behavior being computed. The same method that had been employed in the first experiment to measure the time to compute the event-matching process was also utilized in this experiment (Section 5.1.1) (i.e., the average time to compute the event-matching function, Equation 3.16, for each event in every episode being recalled was recorded as the computation time). The quality of the behavior, performing a navigational task, was measured from two aspects: spatial (path-length) and temporal (duration).

More specifically, as in the first experiment (Section 5.1), this experiment was conducted with the Gazebo configuration (Section 4.2.2). In other words, AIR was executed on a laptop machine (Intel Pentium III) and interacted with the simulated Pioneer 2 DX robot in Gazebo running on a desktop machine (Intel Pentium 4). HServer was set up to relay the perceptual (sonar) and behavioral information between the two processes from the same desktop machine. The same indoor environment used in the first experiment was utilized in this experiment to perform a navigational task (Figure 52).

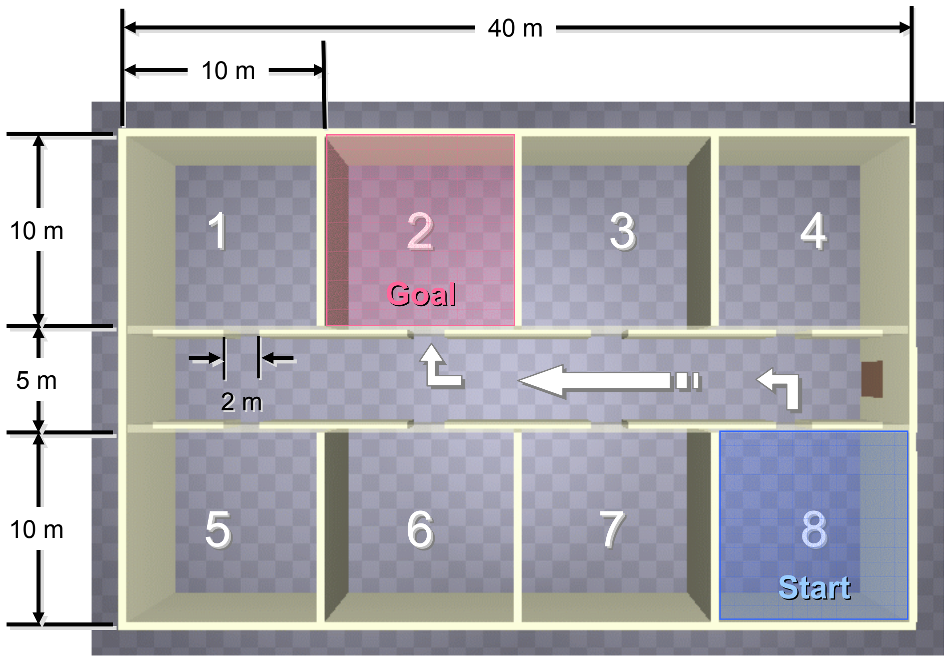


Figure 52: The simulated indoor environment used in the first part of the third experiment.

In a training episode, the robot was dispatched from Room 8, and the combination of the *Avoid-Obstacle*, *Enter-Opening*, *Move-Forward*, *Swirl-Obstacle*, *Move-Leftward*, and *Move-Rightward* motor schemata (Section 4.1.3) were manually instantiated[15] in order to navigate the robot into Room 2 via the hallway. The robot received a fixed reward value (1.0) at the end of the episode (i.e., as soon as entering Room 2), so that episodes with fewer events would be prioritized during behavior selection. Five training episodes were gathered since

[15] Although ideally, by enhancing the motivation and reward functions (Sections 3.3.1 and 3.3.2), the robot should first learn the entire sequence of the behavioral instantiations automatically in a developmental fashion, here, supervised training was used as those functions had not yet been fully developed.

the limited-history case (explained below) could exploit only five recent episodes.

At each test run, the robot was released from the starting position (Room 8), and the run was terminated as soon the robot autonomously navigated itself into the goal (Room 2). The qualities of the behavioral performance were measured in terms of the total distance the robot traveled (path-length) and the time the robot took to reach the goal (duration). During the test, the two conditions were evaluated: namely, the *limited-history* case and the *full-history* case. The condition in which no cap was imposed to limit the number of the relevant episodes being processed in the event-matching process (i.e., trimming the history length) is referred to as the *full-history* case. On the other hand, when the number of the relevant episodes being recalled in the recollection process was restricted by the cap, the condition is referred to as the *limited-history* case. For the limited-history case, the latest five episodes that met the goal condition were selected. As discussed above, the hypothesis here is that, for the full-history case, if the number of the events in each episode is constant, the time to compute event matching increases linearly with respect to the number of the episodes processed during the proactive (anticipatory) computation. Another assumption is that reduction of this computation time can be achieved if the history length is trimmed. Our main question in this experiment (i.e., Subsidiary Research Question 4) is to identify whether such reduction of the computation time would compromise behavioral selection. Thus, the quality of the behavior was evaluated for those two conditions. As discussed in Section 3.2.1, this aspect of the experiment is related to the research conducted by Kira and Arkin [84] in the context of forgetting cases in case-based reasoning. Here, we applied the recency-based elimination strategy to trim the length of the history based on the assumption that the environment would not necessary remain static.

To have a considerable increase in the volume of the episodic memory, the test run was repeated eight times consecutively. In other words, while the number of the available episodes in the memory at the first run was only five (training episodes), at the final run, twelve accumulated episodes were available in its memory (i.e., the full-history case exploited all twelve episodes while the limited-history case exploited only five episodes at the final run). This eight-run-sequence was then repeated four times for each of the two conditions (i.e., 64 runs total). As in the first experiment, the average computation time for the event-matching process was recorded within AIR. To reach Room 2, each run generally required over 300 event-matching cycles; hence, the computation time of the event-matching process for each case was averaged over more than 1200 measurements. Some of the predefined constants utilized during the computation in this experiment are summarized in Table 37, Appendix C (Section C.1).

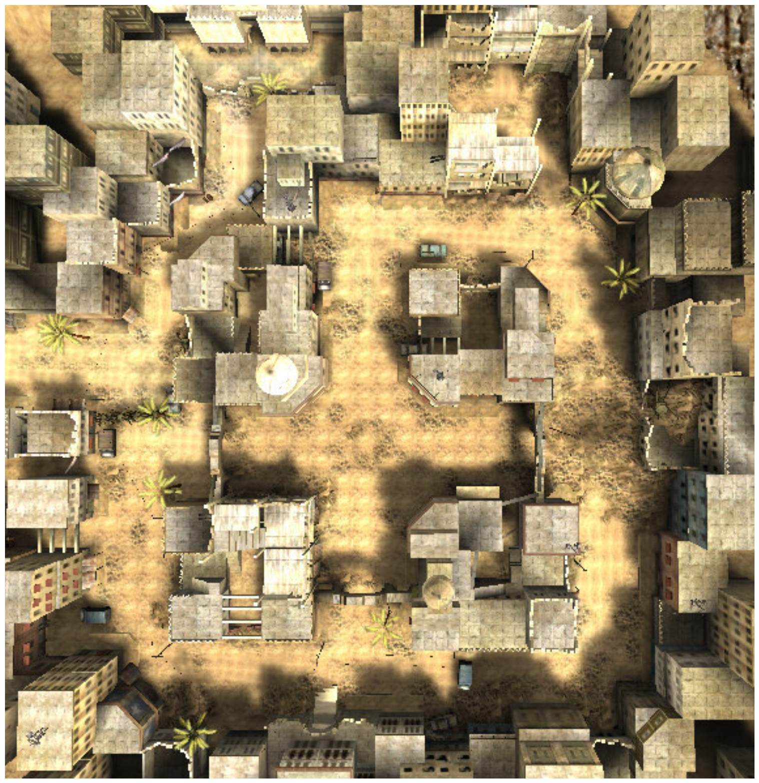
Part B (Improvisational Aspect)

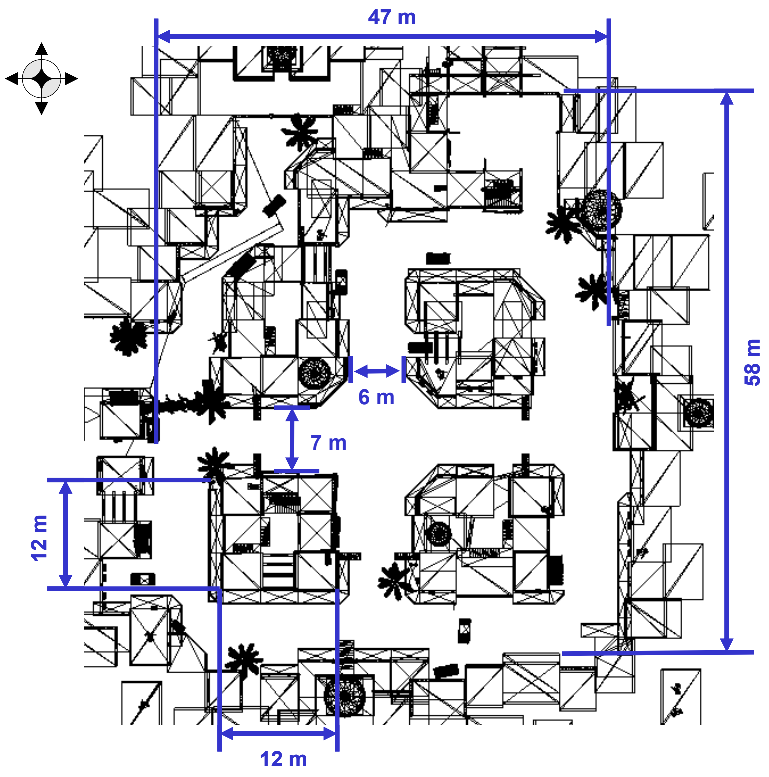
In the second part, the improvisational aspect of the behavioral computation with respect to the behavioral quality is examined through an improvisational detour experiment using USARSim (Section 4.2.3). A virtual village[16] shown in Figure 53 was set up in the USARSim simulation environment. AIR, running on a desktop machine (Pentium 4), was configured to navigate a simulated robot (ATRV-Jr) in this virtual village. USARSim was executed on a laptop machine (Intel Core 2 Duo) while HServer was executed on the same desktop machine as AIR. The sole sensor used in this experiment was an emulated laser scanner (SICK LMS200) mounted in front of the robot, scanning the front 180-degree angle. The readings of the sensor were downsampled to have only 19 measurement points per 180-

[16] The map was adapted from John Falgate's *DM-Blackhawk* map for the Unreal Tournament 2004 game (http://www.mapraider.com/Angelheart).

degree scan (10-degree increment) instead of the original 181 points (1-degree increment) to reduce the computational load. No odometry information was used in this navigational experiment.

As explained in Chapter 3, in our model, what differentiates the computation of improvisation from anticipation are the additional processes: validation (Section 3.2.4) and recovery (Section 3.2.5). More specifically, in anticipation, it is assumed that the relevant episodes recalled by the recollection process (Section 3.2.1) based on the current goal reasonably represent the current state space. Thus, by executing the behavior recorded in those episodes, the goal can be reached in the end. However, the assumption does not always hold in reality. Hence, if the validation process detects a discrepancy between the current state space and the one recorded in the recalled episode, such an episode is removed from the working memory. If there is no more relevant episode left to work with, the recovery process suggests an intermediate goal that would bridge between the current state space and the target state space (containing the goal). The behavior computed from this manner is referred to as improvisation. Although our model can be applied to non-navigational tasks, this experiment was set up to evaluate the promptness of the behavioral computation with respect to the quality of performing a navigational task in which the robot generated an appropriate intermediate goal to reach a target location when an original path became suddenly impassable. The quality of the behavior in this case is measured in terms of the success rate and the operational speed (explained below).


(a)



(b)

Figure 53: The simulated outdoor environment used in the second part of the third experiment: (a) a bird's eye view of the area rendered in USARSim, and (b) the corresponding map and its dimensions of the area.

In this experiment, the training episodes contained the experience of taking different routes in the village. More specifically, three types of training episodes were prepared: namely, *target-episode*, *positive-episode*, and *negative-episode*. The target-episode (Figure 54 (a)) contained the experience of reaching location A the target location that happened to be also the goal[17] specified in the test runs. (A sample image of location A, rendered in USARSim, is shown in Figure 55 (a).) However, during the test, this episode could not be used to reach the goal in its original form since a barrier (Figure 54 (d)) was newly introduced to block the path before the test began. In a target episode, the behavioral sequence of reaching location A from the start position was manually generated by instantiating a combination of the *Move-Forward*, *Move-Leftward*, *Move-Rightward*, and *Swirl-Obstacle* motor schemata (Section 4.1.3). As in the previous experiment, ideally, this sequence should be generated automatically through a developmental-learning process. However, supervised training was employed here since the motivation and reward functions (Sections 3.3.1 and 3.3.2) that would allow such an automatic generation of behavioral sequence had not yet been fully developed.

The episodes for the positive-episode type contained the experience in which the robot took a different route in the village (Figure 54 (b)). More specifically, the robot was released from an alternative starting point, approximately 15 meters west of the releasing point used in the target episode and driven manually by instantiating a combination of the same motor schema types used in the target episode to reach an alternative target location B (Figure 55 (b)). Note that starting point and the goal location of this type coincided with the path the robot took during the target episode. This episode type is considered "positive" as

[17] The target location (goal) was specified in the form of the laser scanner readings.

it allowed the robot to utilize this episode as the bridging episode to bring itself to the ultimate goal eventually when the route in the target episode was found impassable (Figure 54 (d)). Consequently, to distinguish themselves from the episodes from the negative-episode type (explained below), a positive reward value (100) was assigned to every positive episode when the robot reached its goal (location B).

The negative-episode type was prepared, so that it could be verified that the behavior-selection process indeed chooses the advantageous behaviors rather than mere alternative behaviors even when improvising. More specifically, in a negative episode, the robot was dispatched from the same starting position used in the positive episode (Figure 54 (c)), and the goal was set to be also same as the one used in the positive episode (i.e., location B). However, when training (supervised), instead of reaching location B by turning left (towards the north) at the intersection (situated at the center of the village), the robot was deliberately driven straight towards the east, leading the robot to fall into a crater at location C (Figure 55 (c)). In other words, if the behavioral sequence from this type of the episode was chosen, the robot would never be able to reach the intermediate goal (location B) or consequently its final destination (location A). To distinguish itself from the episodes of the positive type, a punishment value (-100) was given to every negative episode when the robot fell into the crater at the end.

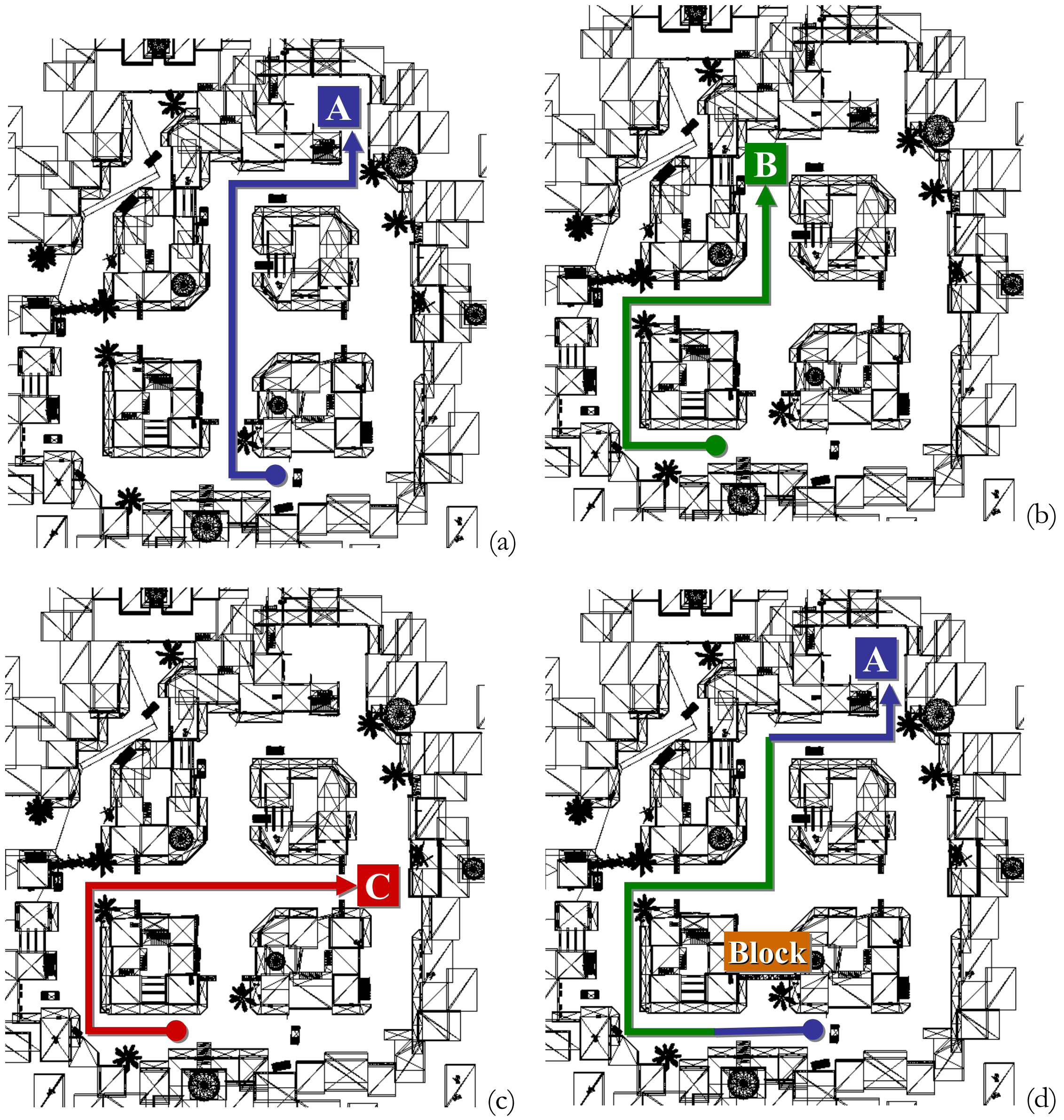


Figure 54: The destinations and the (ideal) paths for four different episode types: (a) the *target-episode* type, (b) the *positive-episode* type, (c) the *negative-episode* type, and (d) testing episodes when successfully detoured.

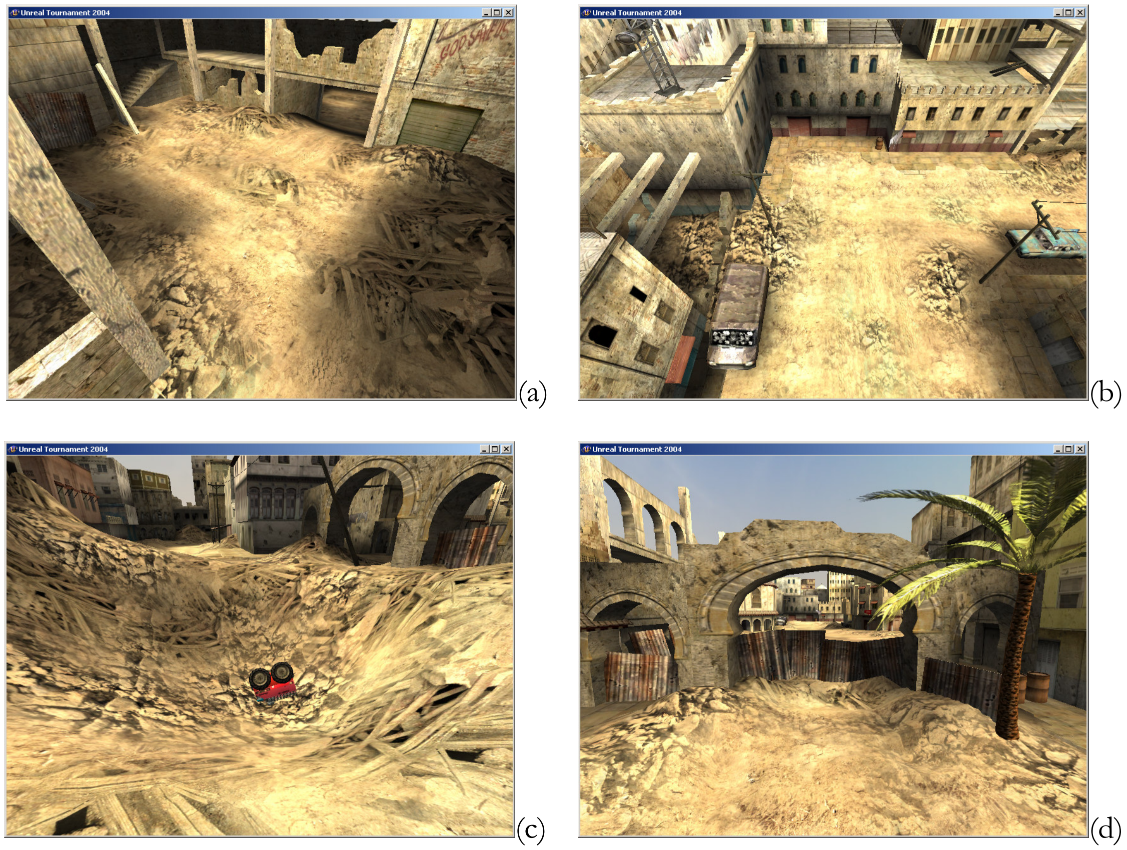


Figure 55: Screen captures of USARSim at the four landmark locations: (a) location A (target), (b) location B (an intermediate point), (c) location C (crater), and (d) the south center-ally entrance (blocked).

During the testing, the robot was released from the same starting position used in the target episode. The observation at location A was used as the goal, so that the robot could initially retrieve the target episode as a relevant one. However, in this case, the south entrance to the center alley located in the middle of the robot's path was set up to be blocked (Figure 55 (d)). Hence, as shown in Figure 54 (d), the robot would have to use a portion of a positive episode to detour around the blocked path and eventually bring itself back to the target episode to reach the goal (location A). In our framework (Section 3.2), this could be done by 1) detecting the anticipatory failure at the blocked entrance, 2) recovering from the failure by generating an intermediate goal (location B), 3) pursuing the intermediate goal by recalling a new set of relevant episodes (positive training episodes), and 4) repursuing the original goal when the intermediate goal is met.

As shown in Table 36, three testing conditions were prepared, so that the behavioral quality could be examined with respect to the three different volumes of the episodic memory. In the first condition (Condition I), when released from the starting position, the robot had three episodes in its memory: namely, one target episode to reach the main goal, one positive episode to provide an alternative route, and one negative episode to ensure that behavior selection is not arbitrary. In Condition II, the robot had one target episode, two positive episodes, and two negative episodes (i.e., five episodes total) in its memory. In Condition III, the memory contained one target episode, three positive episodes, and three negative episodes (i.e., seven episodes total). To measure the promptness of the behavioral computation, the average time the robot took to compute the event-matching process was measured the same way as it was measured in the previous experiments (Experiments 1 and 3A). In other words, the time to compute the event-matching function (Equation 3.16) for an event was averaged over the number of the events in every episode being recalled.

Table 36: The number of episodes in the memory and the testing conditions.

	Number of Episodes in the Memory			
	Target	Postive	Negative	Total
Condition I	1	1	1	3
Condition II	1	2	2	5
Condition III	1	3	3	7

The quality of the improvisational behavior was measured in terms of the success rate and the duration. More specifically, the test run was repeated 12 times for each of the three conditions (i.e., 36 runs total); out of the 12 runs, the number of times the robot successfully reached the goal was measured as the success rate, and among the successful runs, the time to reach the goal was recorded as the duration. Furthermore, the same testing procedure (i.e., the 36 runs) was repeated with the robot operating at half speed (0.5 m/s) (as opposed to the full speed, 1.0 m/s, with which the robot was trained) in order to examine whether the success rate could be improved. The speculation here is that, by reducing the operational speed, the robot could gain more time to process an event sampled in the same environment, improving the event-matching quality and consequently the quality of the overall behavioral performance (success rate). However, obviously, the time to reach the goal was expected to take longer with the slower operational speed. Some of the predefined constants utilized during the computation in this experiment are summarized in Table 37, Appendix C (Section C.1).

5.3.2 Results

Part A (Anticipatory Aspect)

For the hallway navigational task, the robot was able to successfully assess the current state, predict the future consequences of the situation, and execute an action to reach the goal room based on the determined assessment and prediction (i.e., performed anticipation). The graphs in Figure 56 show the averaged computation time required for the event-matching process with respect to the number of episodes in the robot's memory. It can be observed that, if all episodes in the memory were taken into consideration, the overall computation time increased linearly with respect to the number of episodes in the memory. On the other hand, when the cap was imposed on the number of the relevant episodes, the computation time was kept constantly low. A factorial ANOVA confirmed that imposing the cap indeed had significant influence on the computation time with respect to the number of episodes in the memory ($F(7,48) = 343.78, p < 0.001$).

Regarding the quality of the anticipatory behavioral performance, the graph in Figure 57 shows the average length of the path that the robot took to reach the goal with respect to the number of episodes in the memory. Unlike the computation time, the path length was not affected by the number of episodes regardless of the cap ($F(7,48) = 0.11$, $p > 0.1$). The graph in Figure 58 shows the average duration that the robot took to reach the goal with respect to the number of episodes in the memory. Note that, in the full-history case, the arrival to the goal was notably delayed at the last run (when the number of the episodes in the memory was 12). Although another experiment needs to be conducted to identify the cause of this delay, it is speculated that the increase in the computation time for the event-matching process may have caused erroneous event matching during point-turns (i.e., no effect on the path-length), which may have produced oscillations of the robot before making a translational move. However, overall, the average time the robot took to reach the goal was not affected by the number of episodes in the memory regardless of the cap ($F(7,48) = 2.82$, $p > 0.1$).

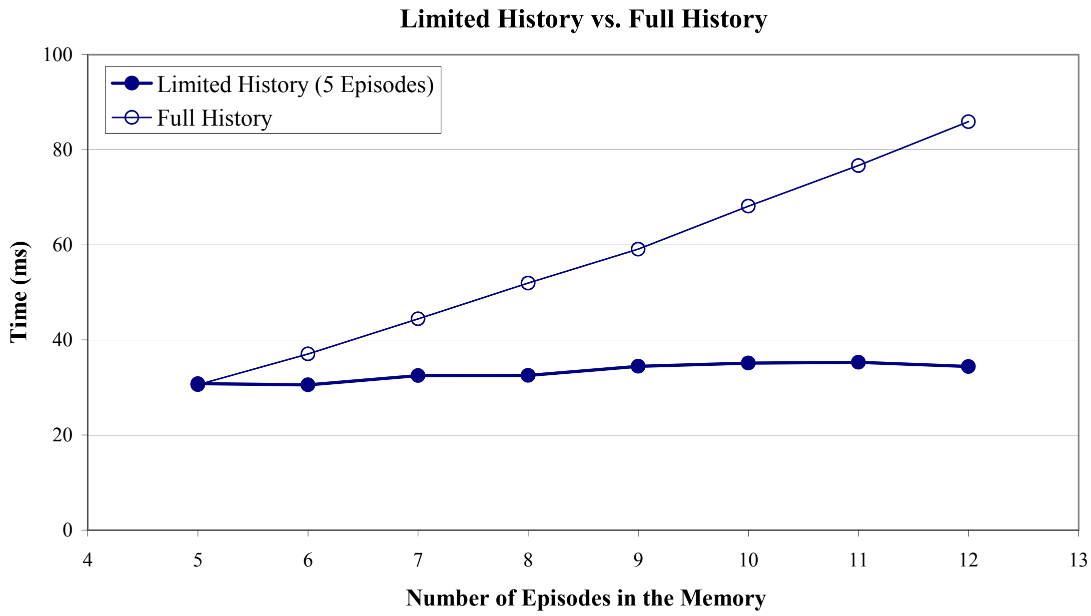


Figure 56: The average computation time required for the event-matching process in the anticipatory experiment with respect to the number of episodes in the memory. When only five recent episodes were used to compute the anticipatory behavior (limited-history), the time to compute event matching stayed constantly low. However, when all episodes in the memory were exploited (full-history), expectedly, the event-matching time increased linearly with respect to the number of episodes. (See Table 47 in Appendix C for the numerical values. The standard error measurements (too small to display here) are also reported in Table 47.)

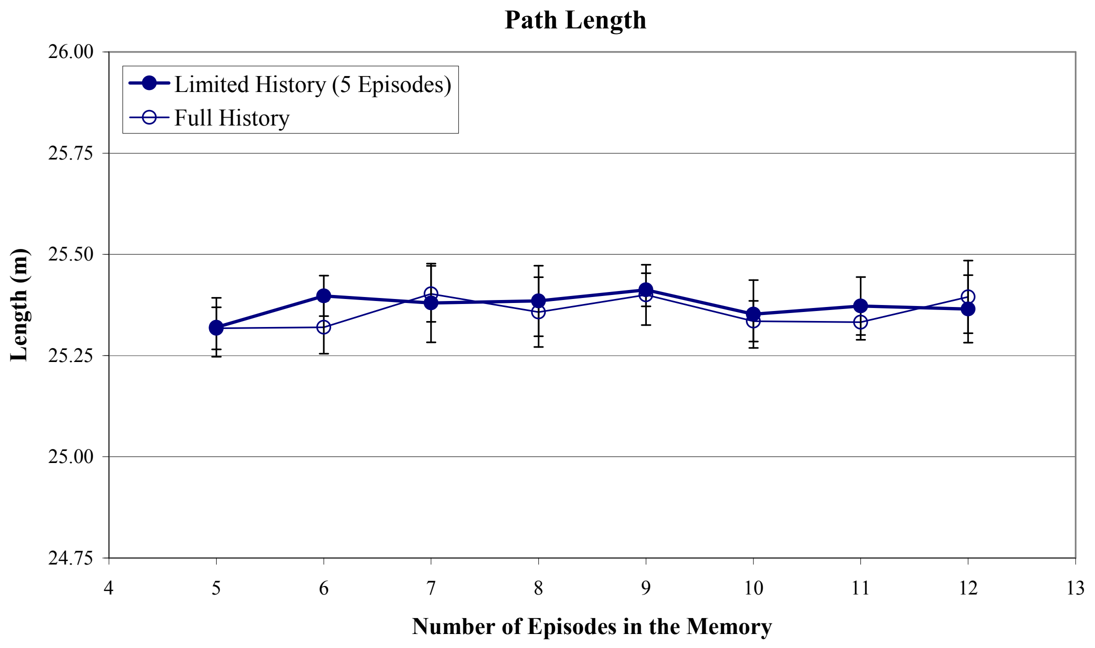


Figure 57: The average path length with respect to the number of episodes in the memory. The vertical error bars indicate the standard error measurements. The path-length did not significantly change regardless of how many episodes were used to compute the anticipatory behavior. The vertical error bars denote the standard error measurements. (See Table 48 in Appendix C for the numerical values.)

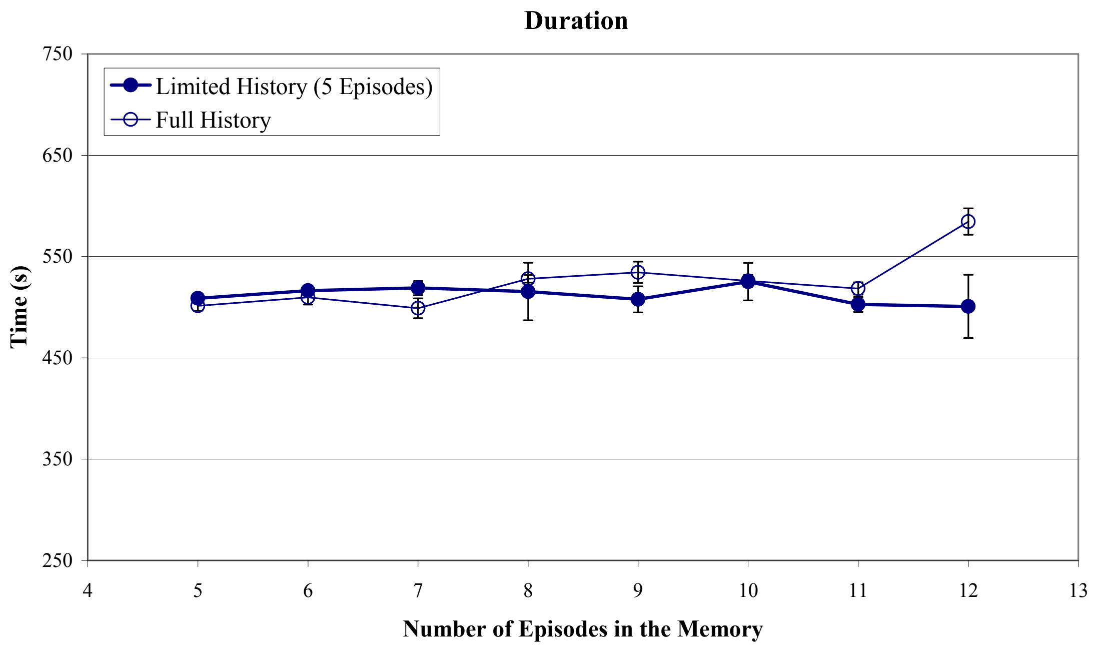


Figure 58: The average duration with respect to the number of episodes in the memory. Overall, as in the path-length graph (Figure 57), the duration did not substantially change regardless of how many episodes were used to compute the anticipatory behavior. The only instance when a considerable difference was recorded was at the last trial for the full-history case. The vertical error bars indicate the standard error measurements. (See Table 49 in Appendix C for the numerical values.)

Part B (Improvisational Aspect)

As shown in Figure 59, in terms of the improvisational behavioral performance (the number of successful runs out of 12 trials), regardless of the robot's speed, the robot was found to perform best in the first condition (three episodes were in the memory) and worst in the third condition (seven episodes in the memory). In any condition, the robot at half speed (0.5 m/s) performed always better than the robot at full speed (1.0 m/s). The best performance (11 out of 12) was achieved by the half-speed robot in the first condition, and the worst performance (0 out of 12) was recorded by the full-speed robot in the third condition. However, as shown in Figure 60, expectedly, the slower speed cost the behavioral quality in terms of the duration as it took substantially longer time to reach the goal when the robot was driven at half speed ($F(1,33) = 239.79, p < 0.001$).

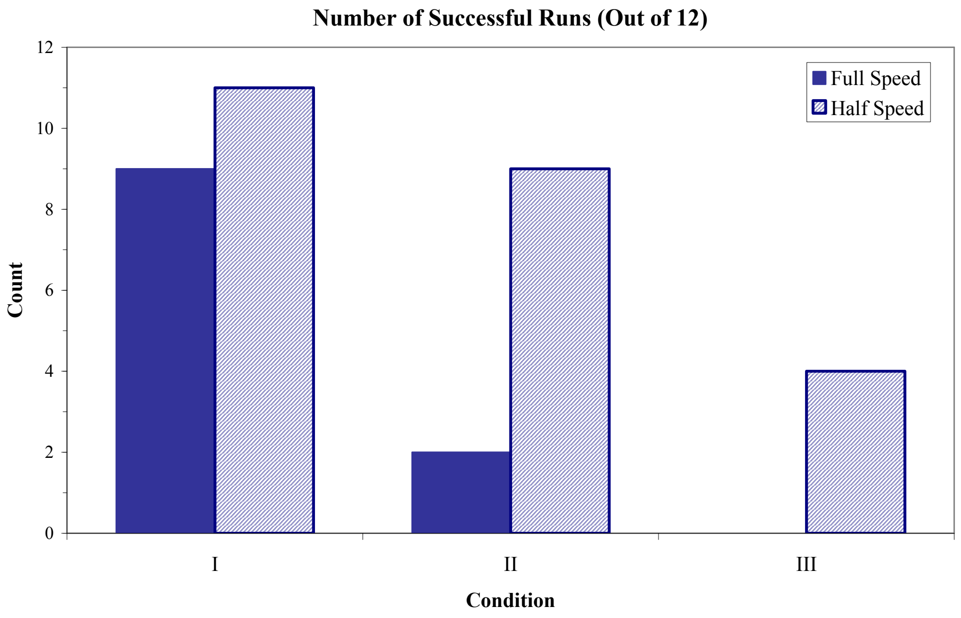


Figure 59: The number of successful runs (out of 12) with respect to the three experimental conditions (Table 36). In all three conditions, the robot at half speed performed better than the robot at full speed. Regardless of its speed, the robot in Condition I (three episodes in the memory) performed better than the other two conditions (9 out of 12 for the full-speed robot and 11 out of 12 for the half-speed robot). In Condition II (five episodes in the memory), when the robot was at full speed, the number of the success runs became substantially small (2 out of 12) while, for the half-speed robot, the number of the successful runs was still high (9 out of 12). In Condition III, the robot performed worst among the three conditions (0 out of 12 for the full-speed robot and 4 out of 12 for the half-speed robot).

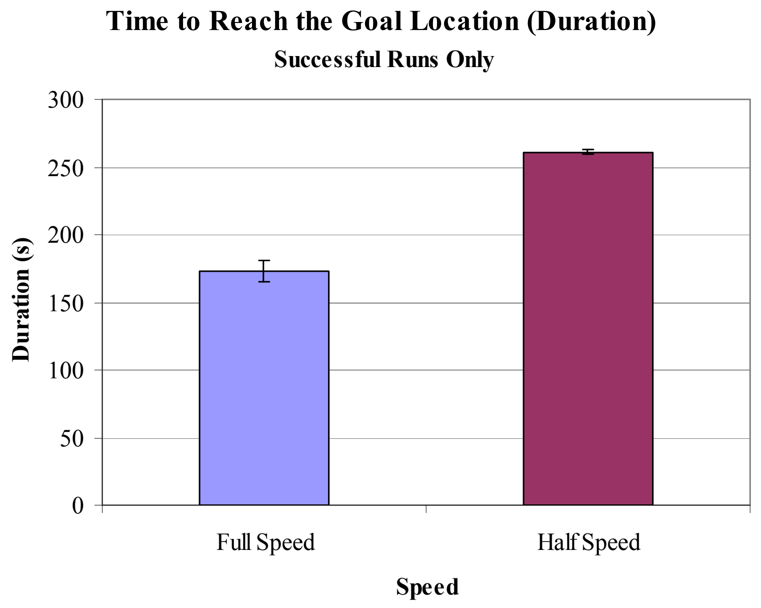


Figure 60: The time to reach the goal location (duration) in successful runs. As expected, when the robot was driven at half speed, the duration was taken longer. The vertical error bars indicate the standard error measurements. (See Table 51 in Appendix C for the numerical values.)

The graphs in Figure 61 show the average time for both the full-speed robot and the half-speed robot to compute the event-matching process with respect to the number of the training episodes in the memory (i.e., the three different conditions). For comparison, the average time the robot took to sample an event during the training is also plotted in the graph. Similar to the trend found in the anticipatory experiment (Figure 56), the computation time required for the event-matching process was found to increase with respect to the number of the episodes in the memory (regardless of the robot's speed). A one-way ANOVA indicates that the number of the episodes in the memory is indeed a strong factor to determine how long the robot takes to compute event matching (full-speed: $F(2,33) = 16.93$, $p < 0.001$; half-speed: $F(2,33) = 82.36$, $p < 0.001$). Unlike the anticipatory experiment, however, this increase in the event-matching time was not precisely linear with respect to the number of the episodes in the memory. In particular, the event-matching time in Condition III (when the number of the episodes in the memory was seven) was less than it would have been if the increase was linear. This is likely caused by the fact that, in Condition III, the robot was often observed to have terminated the intermediate goal prematurely, confusing the right-turn corner at location B (the intermediate goal location) with other right-turn corners found earlier in its path. Once the intermediate goal had been terminated, the number of the relevant episodes since then become just one (the target episode), requiring less time to compute the event-matching process than when the relevant episodes were six (three positive episodes and three negative episodes).

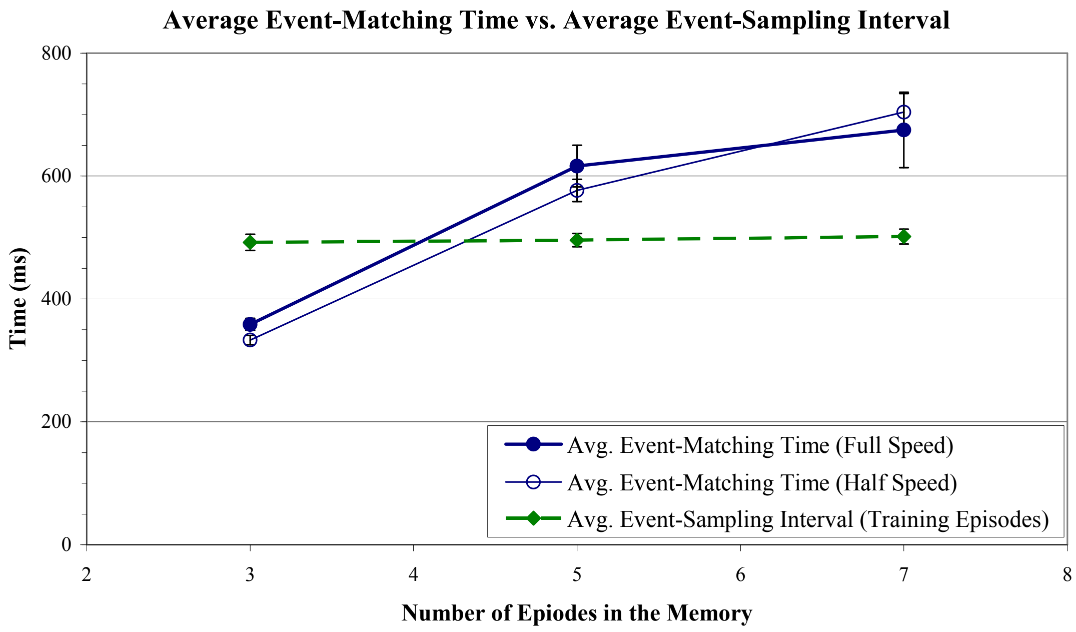


Figure 61: The average computation time required for event matching and the average event-sampling interval in the improvisational experiment with respect to the number of episodes in the memory. Regardless of the robot's speed, the average event-matching time was found to exceed the average sampling interval of the training episodes. The vertical error bars indicate the standard error measurements. (See Table 50 in Appendix C for the numerical values.)

Note that this event-matching error is likely related to how much the event-matching time (during the testing) exceeded the average time the robot took to sample an event during the training (Figure 61), and it is also speculated as the source of the poor performance in Conditions II and III (Figure 59). (Recall from Section 3.1.1 that the event-sampling intervals are determined by the characteristics of the perceptual signal; if the discontinuity in the signal is detected, a new event is sampled.) As shown in Figure 61, the only testing condition in which the robot took less time to compute event matching than the average event-sampling interval during the training was Condition I; in the other two conditions, the average event-matching time was found to exceed the average event-sampling interval. Since the event-matching process has to be performed whenever a new event is sampled (Section 3.1.1), having excessive event-matching time likely affects the quality of event matching. By rearranging the same set of the data used above, as shown in Figure 62, the excessive event-matching time (the average event-matching time minus the average event-sampling interval) for the full-time robot was also plotted against the performance (i.e., success vs. failure). According to a one-way ANOVA, the unsuccessful runs were found to have significantly more excessive event-matching time than successful runs ($F(1,34) = 15.38, p < 0.001$). Note, however, that a more comprehensive experiment needs to be carried out in order verify whether this analysis can be applied for other types of domains/environments.

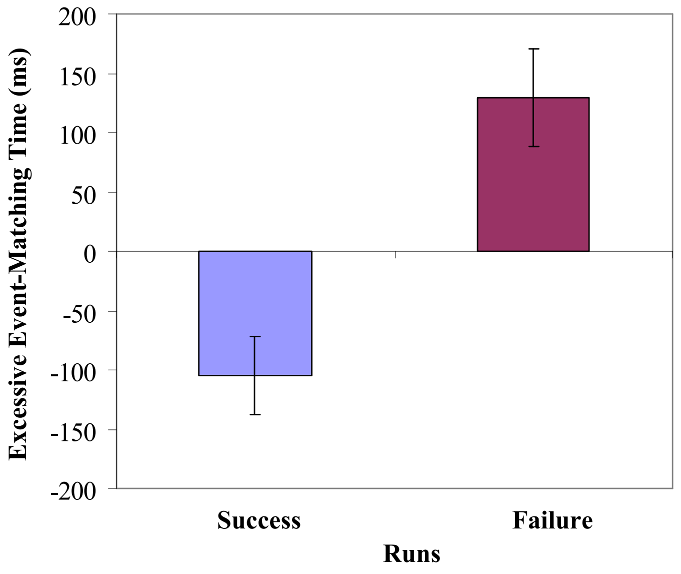


Figure 62: The difference between the successful and unsuccessful runs of the full-speed robot in terms of the excessive event-matching time. When the time to compute the event-matching process exceeded the average event-sampling interval of the episodes in the memory, the performance was found to be overwhelmingly poor. (See Table 52 in Appendix C for the numerical values.)

5.3.3 Discussion

The first part of the experiment (anticipatory aspect) demonstrated that the time to compute the event-matching process linearly increases with the size of the relevant episodes being recalled. Although it is expected to be not as severe as being polynomial, the linear increase can become potentially damaging if the computational resource is limited as seen in the second part of the experiment (discussed below). In the first part of the experiment in which the environment was kept constant, the increase in the number of the episodes did not substantially improve the quality of the behavioral performance whether it is path-length or duration. Hence, by trimming the history length, the computation time required for the anticipatory aspect of the proactive behavior computation can be indeed kept as $O(n)$ instead of $O(kn)$ (where n and k are the number of events in an episode and the number of episodes in the memory, respectively).

However, this result should be compared with the result from the second experiment on the somatic markers (Section 5.2.2). When multiple options are available (e.g., four potential locations to find survivors), the robot has to have enough experience in the environment to start making advantageous decisions. Being able to trim the history-length becomes important when the computational resource is limited. As indicated in the second part of the experiment (improvisational aspect), the increase in the number of the relevant episodes can cause excessive event-matching time (i.e., the time to compute the event-matching process exceeds the event-sampling interval in recalled episodes), and it seems to affect the quality of event matching negatively[18]. This problem is the equivalent of having a

[18] Note that a more comprehensive experiment needs to be conducted in order to verify this claim

POMDP (Section 2.4.4) not being able to estimate the current state accurately. Thus, if the robot cannot accurately find an appropriate matching event, it can lead to poor overall performance. In other words, the robot has to have enough experience in the environment to make advantageous decisions, but when the computational resource is limited, in order to sustain behavioral quality, the history length has to be trimmed, so that the number of the episodes to be processed by the event-matching process can be restricted.

An obvious solution to overcome this limitation is to use a computer with a faster CPU. If the CPU is fast enough to eliminate the excessive event-matching time, the event-matching problem should be resolved. On the other hand, if the CPU is fixed, there are at least two more solutions to work around this problem. The first one is simply to make the robot move slowly as we experimented in the second part of this experiment. By moving at half speed, for example, the allowable time to compute event matching with respect to the current event should be doubled since events are sampled at the same locations with respect to salient features in the environment. Consequently, the performance in terms of the successful rate seems to improve with a slower speed although a slower speed requires more time to complete an assigned task (i.e., this solution is not suitable for a time-critical task).

The other solution is to adjust the transition model dynamically. Instead of assuming event matching is performed every time when a new event is sampled (Section 3.1.1), for example, if the event-matching time is five times slower than an event-sampling interval, it can be assumed that the event matching is performed once per five event-sampling intervals. Accordingly, as shown in Figure 63, this new assumption changes the graph of the Poisson probability function from the one labeled "$\overline{d}$ = 1.0" to "$\overline{d}$ = 5.0", and the transition model (Equation 3.18) should be adjusted based on this change. A subsequent experiment has to be conducted in order to validate this claim.

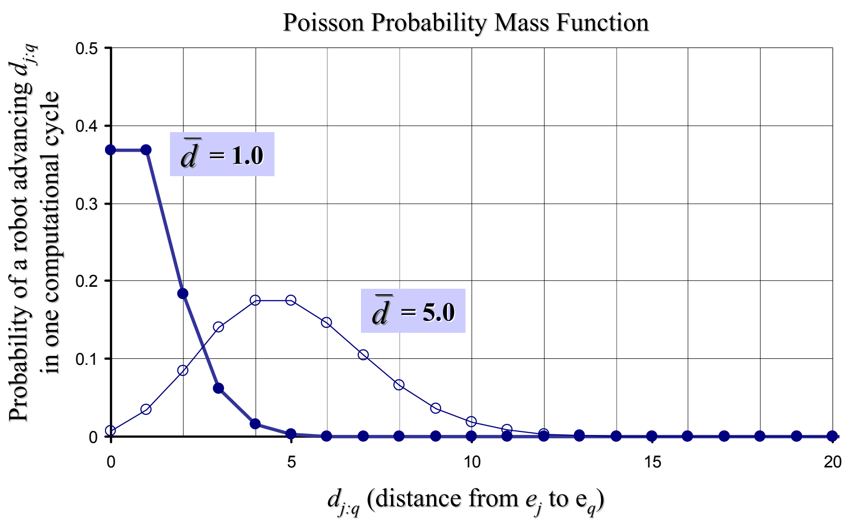


Figure 63: The probability mass function for the Poisson distribution (cf. Figure 26).

Nevertheless, this experiment has successfully demonstrated that, without encoding spatial information (e.g., dead reckoning, GPS, etc.), our framework allows the robot to successfully navigate around a blocked route by promptly detecting an unanticipated circumstance of the situation, finding a fallback solution to deal with the situation, and executing an action to have a desired outcome (i.e., improvisation). How this ability can be applicable to other types of the environment needs to be addressed in future work.

As reviewed in Section 2.2.3, the anticipatory system developed by Schmajuk and Thieme [152] also solves a similar detour problem. While both approaches were inspired by how mammalian hippocampus works, there is a key difference between them; Schmajuk and Thieme's approach models the world using a spatially oriented representation that is a topology of the environment while our approach represents the world using episodes. The spatially oriented representation allows the system to reason about the spatial relationships in the environment (e.g., it can easily find an optimal path). On the other hand, the strength of our approach is that the system is capable of reasoning about the world in terms of episodes. Hence, the application is not limited to just navigational tasks, but it should be also able to handle non-spatial behavioral tasks such as manipulation of objects or communication with humans or other robots.

From a high-level perspective, a hybrid deliberative/reactive control architecture proposed by Goel et al. [64] is relevant to our system as well. More specifically, in their method, the robot executes a series of behaviors specified in a predefined task (e.g., pushing a box), and the behavioral performance is constantly monitored by the model-based deliberative process. If a behavioral failure (e.g., being trapped in obstacles, etc.) is detected, the formal analysis is invoked by the deliberate process in order to identify the cause of the failure and suggest a solution to resolve the failure (e.g., replacing the faulty behavior). This method is based on the assumption that the domain knowledge (model) to detect behavioral failures, analyze the causes of the failures, and fix the failures based on the analysis already exists within the system. However, the question of how to develop such a high-level model automatically for a new domain/environment has not yet fully addressed. On the other hand, our approach does not require development of such a high-level model. The model that the robot has is its own experience (episode) encapsulating a temporal sequence of events, and it is automatically generated by simply interacting with the environment. Another distinction of our approach from Goel et al. [64] is handling of uncertainties. Through a recursive Bayesian filtering (Section 3.2.2), our approach can estimate the current state even if it is not directly observable.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this chapter, the conclusions of this dissertation and the future research direction are discussed. In particular, the first section summarizes the work accomplished in this dissertation. The second section describes the contributions of this research, and the third and final section discusses future work.

6.1 Summary of Work

As discussed in Chapter 1, the primary research question of this dissertation was established to determine how extended experience of a robot affects its ability to behave proactively, that is, to act in an anticipatory and/or improvisational manner (Section 1.2.1). Here, anticipation is to perform an assessment of the current situation, a prediction of the future consequence of the situation, and the execution of an action to have a desired outcome based on the determined assessment and prediction. Improvisation is to promptly detect an unanticipated circumstance of the situation, find a fallback solution to deal with the situation, and then execute an action to have a desired outcome. The primary research question was investigated by exploring the following subsidiary questions (Section 1.2.2):

1. *What common denominator do the processes of anticipation and improvisation for a robot share in terms of recollection and exploitation of past experience?*
2. *What information should a robot extract from a current episode of experience to be remembered in order for it to be utilized upon anticipation and improvisation in the future?*
3. *How should past episodes of experience be organized in the memory of a robot in order for them to be utilized upon anticipation and improvisation?*
4. *Does a memory with integrated somatic markers help a robot achieve better anticipation and/or*

improvisation than without them?

5. *What is the trade-off between promptness and the quality of anticipation/improvisation that a robot performs?*

These questions were addressed during the development of the main computational model (Chapter 3) and the evaluation of the integrated system (Chapters 4 and 5). In the following subsections, while reviewing how these subsidiary research questions were addressed, the computational model and the integrated system are summarized.

6.1.1 Proactive Intelligence

A biologically-inspired computational model of proactive intelligent behavior for robots, which integrates multiple levels of machine learning techniques to accomplish anticipation and improvisation, was developed in Chapter 3. Regarding the first subsidiary research question, this model has shown that the computation of anticipatory and improvisational behaviors do share substantial common denominators: namely, the common foundational data structure (Section 3.1) and the common processes of recollection (Section 3.2.1), event matching (Section 3.2.2), and behavior selection (Section 3.2.3). The most primitive element of the foundational data structure is called an *event*, which also addresses the issue raised by the second subsidiary research question: what information should be extracted from the experience? An event represents the robot's firsthand knowledge of the world and consists of perceptual, behavioral, and reward signals. Here, the perceptual signal is a set of raw sensor readings, the behavioral signal is a set of instantiated motor schemata, and the reward signal is the output of the internal reward function (Section 3.3.2) that determines the desirability of the robot's current state. The reward signal also relates to the fourth subsidiary research question (effectiveness of a somatic marker) and was further investigated by one of the three experiments (reviewed in Section 6.2 below).

Regarding the third subsidiary question (organization of the memory), our foundational data structure was established based on how episodic memories are believed to be organized in the mammalian hippocampus (Section 3.1). An *episode* consists of a series of events, which are arranged in the order that the robot experienced them. As discussed in Section 6.2 below, the behavioral computation was found to benefit significantly from this unidirectional temporal linear-chain formation of events.

Events are sampled whenever there are characteristic changes in the incoming perceptual signal (Section 3.1.1), and episodes are formed whenever there are characteristic changes in the reward signal or goal signal (Section 3.1.2). If there are characteristic changes in the behavioral signal, computational units called *referent nodes* are extracted (Section 3.1.3), and later they are utilized during the computation of improvisation (Section 3.2.5).

Utilizing this foundational data structure, proactive behavior is computed through a series of processes: namely, recollection (Section 3.2.1), event matching (Section 3.2.2), and behavior selection (Section 3.2.3). Recollection is a process in which episodes that are relevant to the current goal are retrieved from memory using instance-based learning (Section 2.4.3). Event matching is a process in which a set of past events in the relevant episodes that best describe the current state (collectively) are identified using a recursive Bayesian filter. In the behavior-selection process, the behavior that is most beneficial for the robot (measured in terms of expected utilities) is selected using a Markov decision process. As noted above, whether it is for anticipation or improvisation, these three are the common processes that are always performed whenever proactive behavior is computed.

The key difference between the anticipatory computation and the improvisational computation is the assumption being made on the episodic contents. In anticipation, the relevant episodes retrieved for the current goal (specified by the motivation function

(Section 3.3.1)) are assumed to represent the current state space fairly accurately. Hence, by executing the action computed based on those episodes, the robot is assumed to reach the goal eventually. On the other hand, improvisation does not assume that the recalled episodes are accurate representations of the current state space; hence, intermediate goals are extracted from one of the relevant episodes in order to bring the robot closer to the main goal state via multiple stages. To switch from the anticipation mode to the improvisation mode, at each time cycle, how a matched event progresses in each episode is monitored by the validation process (Section 3.2.4). When all of the relevant episodes are found invalid (i.e., the event progress is found too slow), the recovery process (Section 3.2.5) is invoked to select the intermediate goals.

6.1.2 AIR

The computational model developed in this dissertation is implemented within a software architectural framework called *AIR* (Section 4.1) and a physically realized robotic system (Section 4.2). AIR, written in Java, comprises two layers: deliberative and reactive subsystems. The deliberative part implements the processes of the proactive behavior computation (Section 4.1.2), and the reactive subsystem translates the behavioral output from the deliberative layer into an appropriate set of motor commands based on the implemented motor schemata (Section 4.1.3).

Three hardware configurations were implemented. In the first configuration, AIR was interfaced with a real robot (ActivMedia's Pioneer 2 DX) (Section 4.2.1). In the second and third configurations, AIR was interfaced with high fidelity three-dimensional simulators: namely, the University of Southern California's Gazebo (Section 4.2.2) and the National Institute of Standards and Technology's USARSim (Section 4.2.3) respectively.

To investigate the third subsidiary research question (organization of the memory), using the Gazebo configuration, the foundational data structure was evaluated in terms of required computational time for event matching with respect to the number of events in an episode (Section 5.1). More specifically, because the events in an episode are organized in a unidirectional temporal linear-chain fashion, an assumption was made to predict the range the event can progress within a time cycle (Section 3.2.2). In other words, to implement the transition model, the transition probability between two events can be approximated by the Poisson probability density function. Since this probability was found to become essentially zero if the distance between the two events is greater than a five-event length, the computation of the transition model can be optimized, subsequently reducing the computational time required for the event-matching process. This experiment has shown that the event-matching time can be indeed reduced from an $O(n^2)$ time to an $O(n)$ time.

The fourth subsidiary question (effectiveness of somatic markers) was investigated by the second experiment (Section 5.2) using the real robot configuration. In the somatic marker hypothesis [45] (Section 2.1.2), an animal's internal emotional responses to environmental stimuli are believed to be physically recorded in the episodic memory, so that they can assist making intelligent decisions in the future. An actual gambling experiment [23] using patients with damaged ventromedial prefrontal cortices (presumably a part of the somatic marker circuitry) to test this hypothesis was adapted for our experiment in the context of a robotic search-and-rescue scenario. In our framework, a somatic marker is the reward signal embedded in a sampled event whose value was predominantly determined by the associative rewarding states (Section 3.3.2). The experiment has demonstrated that the somatic markers indeed help the robot to make advantageous decisions in the long run although the robot has to have some minimal experience in the environment before reliably making such advantageous decisions.

The fifth subsidiary research question (trade-off between computational promptness and quality of proactive behavior) was investigated by the third experiment (Section 5.3), which was broken down into two parts: an anticipatory aspect and an improvisational aspect. In the anticipatory part of the experiment, the Gazebo configuration was used. For a simple navigational task, how the number of relevant episodes influences the time to compute the event-matching process and consequently affects the quality of the behavior was investigated. The experiment showed that the event-matching time did indeed increase linearly with respect to the number of relevant episodes retrieved by the recollection process. On the other hand, the quality of the behavioral performance (measured in terms of the path length and task completion time) was not affected by the increase in the number of the relevant episodes. This led to our conclusion that a cap can be imposed to limit the number of the relevant episodes being processed in the event-matching process (i.e., the history length can be trimmed) in order to reduce the computational time. While a linear increase in the computation time is not as severe as a polynomial increase, being able to reduce the time to compute event matching becomes especially crucial for sustaining behavioral performance when the computational resource is limited as seen in the improvisational detour experiment (discussed below). However, as observed in the somatic marker experiment (discussed above), a minimal exposure to the environment is also necessary for the somatic markers to work effectively in order to make an advantageous decision when there are multiple options to explore. Therefore, the history length needs be selected sensibly in a way that it can fulfill both of the requirements. Automatic trimming of the history length therefore needs to be therefore addressed in future work.

To investigate the improvisational aspect of the fifth subsidiary question, a detour experiment was set up in the USARSim environment. As in the anticipatory experiment (discussed above), the time to compute event matching was found to increase with respect to the number of the episodes being processed in the event-matching process. Furthermore, our preliminary analysis indicates that when the event-matching time exceeds the event-sampling interval recorded in the episodic memories, the quality of the behavioral performance (success rate) was found to be affected negatively. Although reducing the operational speed does help improve the quality by easing the computational schedule, since it substantially delays the overall time to complete a task, this solution is not generally applicable for time sensitive tasks (e.g., search-and-rescue tasks). Hence, as discussed above, processing of an extended amount of episodic memories does not necessary merit proactive behavior; instead, adequate filtering (forgetting) of unwarranted episodes seems to be a key to attain its success. This claim, however, needs to be further verified for a broad spectrum of scenarios in future work.

6.2 Contributions

In addition to the development of the relationship between the extended experience of a robot and its ability to anticipate/improvise (i.e., the primary research question) discussed in the previous section, this dissertation provides several contributions. In this section, these contributions are at first highlighted and then discussed in more detail in a subsequent subsection.

6.2.1 Contribution Summary

- **A computational model of proactive intelligent behavior for robots.** Our computational model (Chapter 3) identifies the necessary computational steps and underlying data structure that allow robots to perform anticipation/improvisation, which should benefit roboticists and artificial intelligence scientists who are interested in implementing such systems. While similar previous methods (e.g., [93, 107, 152]) construct world representations in terms of spatial (topological) maps (see Section 2.2.3), our method develops the world model based on the robot's own participated episodes (see Section 3.1). Potentially applicable tasks are therefore not limited to spatial navigation.

- **An experimental result verifying the computational model.** The successful experimentation in Chapter 5 proves that our computational model is not just theoretical, but it can be actually implemented and work in a real world setting. Although the precursor of our computational model had been evaluated by Endo and Arkin [54] in simulation, it is the first case in which the current version of our computational model was examined by a series of experiments including the one using a real robot. Roboticists and artificial intelligence scientists who are interested in the practicality of our computational model should benefit from this result.
- **An efficient world representation that reduces the POMDP computation load.** As discussed in Section 2.4.4, the current trend of POMDP research in the robotics community is to propose an efficient way of computing POMDP solutions since real robots often provide only limited computational resources. The world representation utilized in our computational method is a contribution to this community since it was shown to reduce the computational time associated with the state-estimation process (Section 3.2.2) from $O(n^2)$ to $O(n)$ (see Section 5.1). Furthermore, the advantage of our method compared to others (e.g., [129, 130, 175, 176]) is that this representation can be automatically constructed from the autonomously-acquired world knowledge using real sensors (see Section 3.1).
- **A robotic system capable of performing proactive navigational tasks without pose**

sensors. As demonstrated by all of the experiments in Chapter 5, since our robot develops the world model based on episodes instead of spatial maps (Section 3.1), it can arrive at a goal location without relying on pose sensors. Hence, roboticists who are dealing with autonomous navigation of robots without adequate pose sensors for the environment should benefit from this system. The advantage of our system over similar previous systems (e.g., [93, 107, 152]) is the incorporation of the POMDP method (Section 2.4.4); it can function even in a environment where the current state is not directly observable.

- **The first robotic implementation of the new hippocampal hypothesis by Eichenbaum et al. [51].** Eichenbaum et al. [51] revolutionized the way of understanding the hippocampal function by challenging the dominant traditional view (e.g., [34, 66, 111, 138, 139, 144, 160]) in which the hippocampus was believed to construct a two-dimensional geocentric map to represent the world; they instead proposed that the hippocampus constructs the world model in terms of discrete episodic memories (see Section 2.1.1). To the best of our knowledge, however, no roboticist had implemented this hypothesis on a robotic system before our implementation (Section 3.1). Biologists who are interested in the relevance of biological theories outside biology and roboticists who are interested in biologically inspired approaches in robotics should benefit from this effort.
- **A novel evaluation method for robotic somatic markers.** Bechara's gambling experiment [23], which evaluated Damasio's somatic marker hypothesis [45] using human subjects (see Sections 2.1.2 and 5.2), is best known for verifying the role of emotion in decision-making (cited by over 1000 papers according to Google). Although there are examples of robots incorporating this hypothesis (e.g., [30, 190]), before our

experiment (Section 5.2), to the best of our knowledge, no roboticist had examined robotic somatic markers through Bechara's gambling experiment. Biologists who are interested in the relevance of biological theories outside biology and roboticists who are interested in biologically inspired approaches in robotics should benefit from this effort.

- **A novel robotic system that performs practical (non-artistic) improvisation.** Although there are robots that perform musical/theatrical improvisation (e.g., [32, 196]), to the best of our knowledge, no robot had yet performed practical (non-artistic) improvisation before ours. (Anderson's Waffler [7] performs practical improvisation in a simulated kitchen, but it is an *AI agent* rather than a *robot* with actual sensors (see Section 2.3.3)). As demonstrated by the detour experiment (Section 5.3), our robot autonomously acquires necessary world knowledge from sensors and processes the information to perform practical improvisation (see Sections 3.2.4 and 3.2.5). Robotics and artificial intelligence scientists who are interested in advancing intelligence and applicability of robots should benefit from this system.
- **A novel way to hybridize CBR and POMDP.** To understand how far the method of case-based reasoning can be extended, today, a number of researchers in the CBR community are interested in hybridization of CBR with other machine learning methods (e.g., [145, 159]). Our computational method is a contribution to this community since it employs a memory-based approach to define the current state space (Section 3.2.1) and a POMDP to find the best action (Sections 3.2.2 and 3.2.3). As discussed in Section 2.4.3, comparing to the similar previous method by McCallum [109, 110], which employs a model-free approach (Q-learning), our method has the advantage of employing a model-based approach (e.g., requiring less experience [81]).

6.2.2 Discussion

The contributions of this dissertation laid out above can be divided into two aspects: practicality and novelty. For example, the general applicability of our computational model can be considered one of the practical contributions. As mentioned above, unlike similar previous methods that had realized anticipatory robot behavior (e.g., [93, 107, 152]) (see Section 2.2.3), our method does not construct the world representation in terms of spatial maps. Instead, the world model is constructed based on episodic memories, which is "a sequence of event representations, with each event characterized by a particular combination of spatial and nonspatial stimuli and behavioral actions," defined by Eichenbaum et al. [51] (see Section 3.1). Hence, the advantage of our computational model is that, theoretically, it can be applied to tasks beyond navigation (to be empirically verified in future work). Moreover, as demonstrated by every experiment in Chapter 5, even for a navigational task, this map-free representation can direct the robot to arrive at a goal location even when no pose sensor is available. This is also a practical contribution to the robotics community since it can further expand the applicability of robots as a pose sensor is not necessary reliable or even available in every situation. On the other hand, the disadvantage of our method in this regard is that, by not incorporating maps, the robot lacks the ability to reason about spatial relationships within the environment, for example, to compute an optimal path. As discussed below (Section 6.3), this shortcoming should be resolved by adding a new layer to the current computational model, which handles symbolic reasoning.

Note that this *spatial map vs. episodic memory* argument bears a resemblance to the similar debate [121] taking place in the neuroscientific community, concerning hippocampal functions. As discussed in Section 2.1.1, one traditional school of thought (e.g., [34, 66, 111, 138, 139, 144, 160]) advocates that, in the hippocampus, the environment is projected into a two-dimensional geocentric map ("cognitive map"), and path-integration is employed to localize the animal with respect to this map. On the other hand, the other school, led by Eichenbaum and his colleagues [51], argues that the hippocampus does not construct such a spatial map; instead the hippocampus constructs a "memory space", which is a collection of discrete episodic memories, collectively modeling the world in terms of the animal's experience (see Section 2.1.1). As discussed in Section 3.4.1, the latter argument sounds more plausible than the former if one considers the fact that various mammals in the wild (monkeys, squirrels, and chipmunks, etc.) effectively navigate in a three-dimensional environment. While some have already implemented the traditional "cognitive map" school of thought on robots (e.g., [35, 114]), to the best of our knowledge, our robotic system is the first attempt in robotics that incorporates Eichenbaum's "memory space" hypothesis within a robotic system. This is a contribution of novelty and should interest biologists who are interested in the relevance of biological theories outside biology and roboticists who are interested in biologically inspired approaches in robotics.

A similar group of researchers (i.e., biologists who are interested in robotic applications and roboticists who are interested in biological inspiration) may also benefit from our novel evaluation method on somatic markers (Section 5.2). The experiment was designed based on Bechara's well-cited (over 1000 papers according to Google) gambling experiment [23] (Sections 2.1.2 and 5.2), which examined the role of emotion in decision making in terms of Damasio's somatic marker hypothesis [45] (Section 2.1.2). In their experiment, patients with damaged ventromedial prefrontal cortices (a part of the presumed "somatic marker circuitry") were compared against the control subjects with normal brains in terms of how their choices (drawing cards from multiple decks) would evolve in the presence of different reward/punishment sequences. At the end, while the subjects with normal brains were able to figure out the truly profitable deck, the patients with defective somatic marker circuitry were found not to be able to identify such an advantageous choice. As shown in Section 5.2, we adapted Bechara's experiment for a robotic search-and-rescue scenario, and a similar result was observed; the robot with somatic markers made more advantageous decisions than the one without somatic markers. Even though some roboticists had already implemented robotic somatic markers (e.g., [30, 190]), to the best of our knowledge, none of the robots was evaluated through Bechara's gambling experiment. Thus, our experiment is considered a contribution of novelty since it is the first attempt in robotics to evaluate robotic somatic markers though the well-regarded experimental method for the subject.

An important practical contribution of our computational model is the computational efficiency that it provides. Recall that a combination of the event-matching (Section 3.2.2) and behavior-selection (Section 3.2.3) processes implements a partially observable Markov decision process (see Section 2.4.4). While POMDPs are notorious for their computational burden, the world representation utilized in our computational model offers reduction of the computational load through multiple ways. First, instead of using a uniformly discretized map, a state space (episode) is partitioned into discrete states (events) based on the predictability (saliency) of incoming perceptual signals (see Section 3.1.1). Hence, the simpler the environment is, the smaller the state space can be to represent it. Second, it utilizes an abstract notion of action to reduce the action space. As explained in Section 3.1.1, instead of low-level motor commands (velocity, turning angle, etc.), action in our computational model is represented in terms of Arkin's *motor schema* [10, 11] (*Move-To-Goal*, *Avoid-Static-Obstacle*, etc.). Pineau et al. [130], who worked on an elderly assistance robot in a nursing home, argued that such a decomposition of action space is a key to solve high-level decision-making problems using POMDPs. Finally and most importantly, as discussed on several occasions throughout this dissertation (Sections 3.1.2, 3.2.2, 5.1, 5.3.3, and 6.1.2), since the events in an episode are organized in a unidirectional temporal linear-chain fashion, the transition probability between two events can be approximated by the Poisson probability density function (in the Bayesian-based event-matching process). Subsequently, by ignoring the transition between two events that are farther than a five-event length apart (since the probability in such case is known to become zero), the computational time required for the event-matching process was reduced from an $O(n^2)$ time to an $O(n)$ time (Section 5.1). The significance of our approach is that, by interacting with the world (with or without human supervision), a robot can automatically extract necessary information from the experience and construct this efficient world representation without human interpretation. Thus, this representation should benefit roboticists who are interested in applying POMDPs to robotic tasks, especially those with limited computational resources.

As mentioned above, a combination of the event-matching (Section 3.2.2) and behavior-selection (Section 3.2.3) process serves as a POMDP in our computational model. Furthermore, a combination of those two plus the recollection process (Section 3.2.1) can be considered hybridization of CBR (see Section 2.4.2) and POMDP if one views that the recollection process is a case-based reasoner. As discussed in Section 3.2.1, technically, the recollection process is closer to instance-based learning (see Section 2.4.3) than CBR. Nevertheless, our method employs a memory-based approach to define the current state space (episode), and the POMDP determines the most advantageous action for the current juncture. This hybridization effort should interest those in the CBR research community who are interested in how far the method of case-based reasoning can be extended by hybridizing it with other machine learning methods. As reviewed in Section 2.4.2, there are examples of CBR hybridization with reinforce learning, for example, to control simple physical systems (e.g., pendulum) [145] or play a real-time strategy game [159]. As discussed in Section 2.4.3, perhaps the most relevant hybridization effort to ours is McCallum's work [109, 110], in which a memory-based approach (instance-based learning) is applied to identify the current state, and model-free reinforcement learning (Q-learning) is employed to determine the optimal action. Note that our behavior-selection process determines action through a model-based approach. As pointed out by Kaelbling et al. [81], a model-based approach has an advantage over a model-free approach since it requires less experience to attain sufficient performance [81]. On the other hand, a known disadvantage is that it requires more computation time. As discussed above, by imposing a strong assumption on how the state space is formed, our method attempts to ease this problem (Section 3.1.2).

Furthermore, our robotic system, which is able to perform practical (non-artistic) improvisation, can be considered a contribution of both practicality and novelty to the robotics and AI communities. It is practical because such ability can be expected to advance intelligence and applicability of robots, and it is novel since it is the first robot that performs practical improvisation. Recall that, in this dissertation, we have been defining improvisation as *the ability to promptly detect an unanticipated circumstance of the situation, find a fallback solution to deal with the situation, and execute an action to have a desired outcome* (Section 2.3). As reviewed in the Section 2.3.3, in the current field of robotics or even artificial intelligence, examples of robotic systems that can actually perform improvisation are very limited. Even those robots that performed improvisation were limited to some artistic (theatrical/musical) purposes (e.g., [32, 196]). However, to the best our knowledge, no robot had performed improvisation for practical purpose before ours. Anderson's simulated *AI agent* [7], Waffler, performed practical improvisation (making a cup of tea etc.) but was not an integrated robotic system (see Section 2.3.3). On the other hand, our robot is a complete robotic system with sensors (see Section 4.2). The detour experiment in Section 5.3 has demonstrated that our computation framework does allow the robot to perform a special case of practical improvisation; the robot could resolve a sudden obstruction to its initially intended route by detecting the anticipatory failure (blocked route) and applying a fallback solution to the failure (intermediate goal). The robot indeed reached its desired state (goal) when the computational resource was adequately available.

Finally, in addition to providing the detailed description on how to implement the computational model within a working robotic system (Chapter 4), this dissertation reports a successful experimental result in Chapter 5, which confirms that our computational model is not merely hypothetical, but it is certainly realizable as a functioning robot. We consider that this result is a contribution of novelty to roboticists and artificial intelligence scientists who are interested in the practicality of our computational model since nobody else has evaluated it before (except its precursor being evaluated by Endo and Arkin [54]). One of the notable findings is that while accumulation of episodic memories in the environment is necessary for the robot to be able to behave advantageously (via somatic markers), the performance can be negatively affected if processing of those episodes exhausts the computational resource and estimation of the current state could no longer be processed punctually. Although our computational framework attempts to optimize such computational time in various ways as discussed above, this problem becomes inevitable if the robot accumulates the episodes extensively through out its lifetime. As discussed in the previous section (Section 6.1.2), filtering or forgetting of unnecessary episodes therefore would become essential if the robot has to be operated in real-time.

6.3 Future Work

As mentioned above, our robot was able to compute navigational POMDP solutions without ever incorporating odometry information. Hence, it can reasonably assume that our approach is applicable to a variety of mobile platforms or situations in which such sensors are not readily available (e.g., hexapod robot, underwater robot, etc.). This hypothesis should be verified in a subsequent experiment.

Although we have demonstrated that our robot was able to behave in an anticipatory/improvisational fashion autonomously, in order for the robot to be self-sustaining in the environment, several issues need to be addressed. First, our auxiliary functions (Section 3.3) are still incomplete. The motivation function (Section 3.3.1) has to be able to adjust the motivational value automatically using a more sophisticated scheme, so that the current goal can be chosen more intelligently. The reward function (Section 3.3.2) also needs to be enhanced, so that, for example, the weight for the similarity between the current observation and an associative rewarding state can be determined automatically. These enhancements should be made in a way that the robot can automatically acquire a necessary sequence of behaviors to accomplish a complex task solely from prior episodes in which simpler tasks had been solved. In other words, the aspect of developmental learning should be also investigated.

Another enhancement that can be made to the whole system is to add a new layer that handles symbolic reasoning or planning. For example, as discussed in Section 3.1.3, the concept of the computational units called *referents* was designed in a way that some classical planning algorithm can be applied to plan a sequence of actions (behaviors). Allowing the symbolic layer to handle spatial reasoning may also benefit the robot in a way that, for example, it can find an optimal path in a maze-like environment. Furthermore, the symbolic layer should also allow the robot to recognize the human languages; hence, high-level interactions with humans can become part of the episodic memories, enabling more sophisticated proactive-behavior to be performed.

Finally, recall Murphy's Law: *anything that can go wrong will go wrong* [1] (Chapter 1). Our model was developed based on a premise that the only way for a robot to counteract this curse is to act proactively. After enhancing the system by placing all those improvements suggested above, our robot when equipped with the right kind of experience should ultimately be able to handle any critical task through a combination of proper anticipation and improvisation: in the precisely the same way that Capt. Murphy's engineering team had safely completed their extremely risky engineering project.

APPENDIX A

DERIVATION

A.1 Derivation of the Recursive Bayes Filter (Equation 3.16)

Given a sequence of observations (o^τ) and executed behaviors (b^τ), the posterior probability of the robot being at some event (e_q) in the past can be calculated by the recursive Bayes filter (Equation 3.16). As explained in [177] by Thrun, the equation can be derived by applying the Bayes' rule once, the Markov assumptions twice, and the law of the total probability once to the posterior.

$$\begin{aligned}
p(e_q \mid o^\tau, b^\tau) &= p(e_q \mid o_\tau, o^{\tau-1}, b^\tau) \\
&= \eta\, p(o_\tau \mid e_q, o^{\tau-1}, b^\tau)\, p(e_q \mid o^{\tau-1}, b^\tau) && \cdots \text{ Baye's Rule} \\
&= \eta\, p(o_\tau \mid e_q)\, p(e_q \mid o^{\tau-1}, b^\tau) && \cdots \text{ The Markov Assumption} \\
&= \eta\, p(o_\tau \mid e_q) \sum_{e_{\tau-1} \in E} p(e_q \mid o^{\tau-1}, b^\tau, e_{\tau-1})\, p(e_{\tau-1} \mid o^{\tau-1}, b^\tau) && \cdots \text{ The Law of the Total Probability} \\
&= \eta\, p(o_\tau \mid e_q) \sum_{e_{\tau-1} \in E} p(e_q \mid b_\tau, e_{\tau-1})\, p(e_{\tau-1} \mid o^{\tau-1}, b^{\tau-1}) && \cdots \text{ The Markov Assumption}
\end{aligned}$$

where η is a scale (or normalization) factor that ensures the sum of all the possible posteriors becomes 1.

APPENDIX B

PSEUDOCODE

B.1 Goal Manager

B.1.1 Selection of the Current Goal

```
// First, find the motivation with the highest magnitude.
HighestMagnitude = negative infinity
for each Motivation in the list of available motivations
      if Motivation’s magnitude > HighestMagnitude then
            SelectedMotivation = Motivation
      end if
end for


// The current goal is the goal of the selected motivation.
CurrentGoal = SelectedMotivation’s goal
```

B.2 Reward Manager

B.2.1 Computation of the Current Reward Value

```
// First, check the similarity between the current observation
// (CurrentPerception) and the current goal (CurrentGoal), and multiply
// it with a predefined constant (K1). SimilarityFunc() is the
// implementation of Equation 3.14.
Value1 = K1 * SimilarityFunc(CurrentPerception, CurrentGoal)


// Next, check the similarity between the current and predicted
// observations. Multiply it with a constant (K2) as well.
// MathcedRelevantEpisodeList is a list of all relevant episodes
// that are successfully matched (see Section B.5.2).
```

```
HighestSimilarity = negative infinity
for each episode in MatchedRelevantEpisodeList
    Prediction = predicted observation based on the event progress
    Similarity = SimilarityFunc(CurrentPerception, Prediction)
    if Similarity > HighestSimilarity then
        HighestSimilarity = Similarity
    end if
end for
Value2 = K2 * HighestSimilarity


// Check the similarities of all the associative rewarding states
// with respect to the current observation.
Value3 = 0
for each RewardingState in the associative rewarding state list
    tmp = (K3 * SimilarityFunc(CurrentPerception, RewardingState))
    Value3 = Value3 + tmp
end for


// Finally, the current reward value is a sum of the three.
CurrentRewardValue = Value1 + Value2 + Value3
```

B.3 Event Sampler

B.3.1 Sampling a New Event

```
// First, predict the current observation by multiplying the value of
// last sensor readings with the weights computed by TD(λ) (Equation
// 3.4).
Prediction = LastPerception whose readings multiplied by Weights
```

```
// Compute the root-mean-squared differences (CurrentError) between the
// current observation (CurrentPerception) and the predicted one.
CurrentError = RMSDiff(CurrentPerception, Prediction)

if (CurrentError < LastError) and (LastError > LastLastError) then
    // The error peaked. Remember the current state as a new event.
    Event's observation = CurrentPerception
    Event's behavior = CurrentBehavior from the behavior manager
    Event's reward = CurrentRewardValue from the reward manager
    notify the episode compiler module about this new Event
end if

// Finally, update the weights for the next time cycle.
// Note: UpdateWeightsFunc() is the implementation of the TD(λ) update
// rule (Equation 3.5).
Weights = UpdateWeightsFunc(Weights, Perception, Prediction)
```

B.4 Episode Compiler

B.4.1 Purposive Contextualization

```
// Check if the current goal is different from the previous time cycle.
// If different, instantiate a new episode. In this case, the previous
// goal is the episode's context.
if CurrentGoal != LastGoal then
    instantiate a new Episode
    Episode's context = LastGoal
    Episode's events = the recorded events since LastGoal started
    send Episode to the episodic memory repository
end if
```

B.5 Anticipatory Processor

B.5.1 Recollection

```
// First, check the similarity between the current goal (CurrentGoal)
// and the context of each episode. As before, SimilarityFunc() is an
// implementation of Equation 3.14.
Counter = 1
for each Episode in the episodic memory repository
      similarity = SimilarityFunc(CurrentGoal, Episode's context)
      if similarity >= predefined threshold then
            add Episode to RelevantEpisodeList
      end if
      increment Counter
      if Counter > the predefined maximum number then
            // It reached the maximum number.
            end the for-loop
      end if
end for
```

B.5.2 Event Matching

```
// Compute the posterior probability of every event in every relevant
// episode through Equation 3.16.
for each Episode in RelevantEpisodeList
      Index = 0
      // Note: Posteriors and PreviousPosteriors are arrays of double,
      // storing the current and previous posterior probabilities,
      // respectively. The size of the arrays is the number of events
      // in Episode.
      for each Event in Episode
```

```
        // First, get the sensor model (Equation 3.17)
        SM = SimilarityFunc(CurrentPerception, Event’s observation)

        // Next, compute the transition model, and multiply the
value
        // with the previous posterior probability. This code
        // supports the computational optimization method discussed
        // in Section 3.2.2. TransitionFunc() is the implementation
        // of Equation 3.18.
        Sum = 0
        EventDistance = 1
        while EventDistance <= 5
                PastIndex = Event’s index – EventDistance
                PastEvent = the event whose index is PastIndex
                MM = TransitionFunc(
                        Event, CurrrentBehavior, PastEvent)
                Sum = SUM + MM * PreviousPosteriors[PastIndex]
                increment EventDistance
        end while

        // The (unnormalized) posterior probability is the
        // multiplication of the two.
        Posteriors[Index] = SM * Sum
        increment Index
end for // each Event in Episode
normalize Posteriors

// Finally, check the entropy of the posterior probabilities.
// CompEntropy() is the implementation of Equation 3.20. If the
```

```
        // entropy is the below the predefined threshold, the event with
        // the highest posterior probability is the matched event for
        // this episode.
        entropy = CompEntropy(Posteriors)
        if the entropy of Posteriors <= predefined threshold then
              MatchedEvent = Event with the highest posterior value
              add Episode to the list of MatchedRelevantEpisodeList
        end if
end for // each Episode in RelevantEpisodeList
```

B.5.3 Computation of the Utility Values

```
// Compute the utility of every event in a relevant episode. This
// in fact needs to be done only once when the episode is formed.
// Note: Utilities is an array of double, storing the utility values
// of the events. Its size is the number of events (NumEvents) in the
// episode.
Index1 = NumEvents - 1
while Index1 >= 0
      Event1 = the event whose index is Index1
      Behavior = behavior of the event whose index is Index1 + 1
      Index2 = NumEvents - 1
      Value = 0
      while Index2 >= 0
            if Index2 <= Index1
                  end the inner while-loop
            end if
            Event2 = the event whose index is Index2
            Probability = TransitionFunc(Event2, Behavior, Event1)
            Value = Value + Probability * Utilities[Index2]
            decrement Index2
      end while // Index2 >= 0
      Utilities[Index1] = Event’s reward + Value
      decrement Index1
end while // Index1 >= 0
```

B.5.4 Behavior Selection

```
// First, find the expected utility values for each relevant behavior
for each Episode in MatchedRelevantEpisodeList
        ExpectedUtility = 0
        MatchedEvent = the matched event in Episode
        NextBehavior = the behavior executed just after MatchedEvent
        for each Event in Episode
                ExpectedUtility = ExpectedUtility +
                        Utilities[Event’s index] * TransitionFunc(
                                Event, NextBehavior, MatchedEvent)
        end for // each Event in Episode

        // Check to see if this behavior already exists in the list
        // created before
        Index = 0
        BehaviorExists = false
        for each Behavior in BehaviorList
                if Behavior = NextBehavior then
                        // Check how many times this behavior was counted so
                        // far in this routine.
                        NumBehaviors = NumBehaviorsList[Index]

                        // Update the averaged expected utility associated
```

```
                    // with this behavior
                    Value = AveragedExpectedUtilities[Index]
                    Value = ((Value * NumBehaviors) + ExpectedUtility) /
                                (NumBehaviors + 1)
                    AveragedExpectedUtilities[Index] = Value
                    NumBehaviorsList[Index] = NumBehaviors + 1
                    BehaviorExists = true
                    end the for-loop
              end if
              increment Index
        end for // each Behavior in BehaviorList

        if BehaviorExists = false then
              // New behavior. Create new entries in the lists.
              Index = the size of BehaviorList so far
              BehaviorList[Index] = NextBehavior
              AveragedExpectedUtilities[Index] = ExpectedUtility
              NumBehaviorsList[Index] = 1
        end if
  end if // each Episode in MatchedRelevantEpisodeList

  // Find the behavior with the highest expected utility value.
  HighestExpectedUtility = negative infinity
  SelectedBehavior = null
  Index = 0
  while Index < the size of BehaviorList
        ExpectedUtility = AveragedExpectedUtilities[Index]
        if (ExpectedUtility > HighestExpectedUtility) then
              HighestExpectedUtility = ExpectedUtility
```

```
            SelectedBehavior = BehaviorList[Index]
      end if
      increment Index
end while
send SelectedBehavior to the behavior subsystem
```

B.6 Improvisational Reasoner

B.6.1 Validation

```
// If the event progress is delayed according to the schedule recorded
// in the episode, remove the episode from the list.
for each Episode in MatchedRelevantEpisodeList
      LatestEvent = the event sampled most recently
      MatchedEvent = the matched event in Episode
      NextEvent = the event in Episode saved just after MatchedEvent

      // Compute the delay via Equation 3.26.
      delay = -1 + ((CurrentTime – LatestEvent’s timestamp) /
                  (NextEvent’s timestamp – MatchedEvent’s timestamp))
      if delay <= the predefined threshold then
            remove Episode from MatchedRelevantEpisodeList
      end if
end for

// If no episode is left in the list, start the recovering process.
if MatchedRelevantEpisodeList = empty then
      invoke the recover process
end if
```

B.6.2 Construction of a Referent

```
// Inspect every event in the episode. If the behavior type changes
// from one event to another, a new node is started. The observations
// recorded in the beginning and end of this behavior instantiation are
// the nodal precondition and effect, respectively.
Precondition = the first event's observation
Effect = the first event's observation
Behavior = the first event's behavior
ReferentNodeIndex = 0
EventIndex = 1
while EventIndex < the number of events in Episode
      NextEvent = the event whose index is EventIndex
      NextBehavior = Event's behavior
      if Behavior != NextBehavior then
            // Save the node.
            Node's behavior = Behavior
            Node's precondition = Precondition
            Node's effect = Effect
            Referent's nodes[ReferentNodeIndex] = Node

            // Set up the next node.
            Precondition = Effect
            Behavior = NextBehavior
            increment ReferentNodeIndex
      end if
      Effect = NextEvent's observation
      increment EventIndex
end while
```

B.6.3 Selection of a Primary Referent

```
// The current MatchedRelevantEpisodeList (from Section B.5.2) is
// empty since the validation process (Section B.6.1) eliminated
// invalid episodes (hence improvisation was invoked). Thus, we utilize
// the previous MatchedRelevantEpisodeList.
ExpectedUtility = negative infinity
for each Episode in the previous MatchedRelevantEpisodeList
      ExpectedUtility = 0
      MatchedEvent = the matched event in Episode
      NextBehavior = the behavior executed just after MatchedEvent
      for each Event in Episode
            ExpectedUtility = ExpectedUtility +
                  Utilities[Event's index] * TransitionFunc(
                        Event, NextBehavior, MatchedEvent)
      end for // each Event in Episode

      // If its expected utility value is higher than others, its
      // referent is the primary referent.
      if ExpectedUtility > HighestExpectedUtility then
            PrimaryReferent = the referent that belongs to this Episode
            ExpectedUtility = ExpectedUtility
      end if
end for
```

B.6.4 Selection of an Intermediate Goal

```
// First, identify the active node that coincides with the last
// known matched event's timestamp.
EventTime = the timestamp of the last known matched event
for each Node in the primary referent
```

```
        StartTime = Node’s timestamps[0]
        EndTime = Node’s timestamps[1]
        if (StartTime < EventTime) and (EventTime <= EndTime) then
            ActiveNode = Node
            end the for-loop
        end if
    end for

    // The intermediate goal is the nodal effect of the active node.
    IntermediateGoal = ActiveNode’s effect
    inject IntermediateGoal into the recollection process

    // It is possible that this IntermediateGoal may not find any
    // relevant episode in the memory. In this case, select more
    // intermediate goals from the neighboring nodes.
    sleep for 1 second
    SearchRange = 1;
    while no relevant episode is recalled by the recollection process
        Node1 = the node in the primary referent whose index is the
                ActiveNode’s index + SearchRange
        IntermediateGoal = Node1’s effect
        inject IntermediateGoal into the recollection process

        Node2 = the node in the primary referent whose index is the
                ActiveNode’s index – SearchRange
        IntermediateGoal = Node2’s effect
        inject IntermediateGoal into the recollection process

        sleep for 1 second
        increment SearchRange
end while
```

B.7 Motor Schemata

B.7.1 Avoid-Static-Obstacle

```
// Generate a repulsive vector from each obstacle, and linearly sum
// them up.
OutputX = 0 // X: The direction of the robot's current heading.
OutputY = 0 // Y: Perpendicularly left of the robot's heading.
if SensorType is sonar or laser then
        Index = 0
        while Index < the number of data points in the reading
                Distance = reading's values[Index]
                Angle = reading's phi_angles[Index]

                // Create a vector opposite of the obstacle
                Angle = Angle + 180 degree
                if Distance <= SAFETY_MARGINE then
                        // The obstacle is within a safety margin. Generate
                        // the maximum repulse.
                        Magnitude = MAXIMUM_MAGNITUDE
                else
                        // The repulse is stronger as the robot gets closer
                        // to the obstacle.
                        Magnitude = MAXIMUM_MAGNITUDE *
                                (MAXIMUM_RANGE - Distance) /
                                (MAXIMUM_RANGE - SAFETY_MARGINE)
                end if // Distance <= SAFETY_MARGINE
                // Add the vectors linearly.
                OutputX = OutputX + Magnitude * cos(Angle)
                OutputY = OutputY + Magnitude * sin(Angle)

                increment Index
        end while
end if // SensorType is sonar or laser
```

B.7.2 Enter-Opening

```
// First, group neighboring sonar/laser points together as a segment if
// their distances are close enough.
if SensorType is sonar or laser then
    SegIndex = 0
    SegStartDistance = reading's values[0]
    SegStartAngle = reading's phi_angles[0]
    SegEndDistance = reading's values[0]
    SegEndAngle = reading's phi_angles[0]
    ValueIndex = 1
    while Index < the number of data points in the reading
        Distance = reading's values[ValueIndex]
        Angle = reading's phi_angles[ValueIndex]

        // Check to see if this point belongs to the current
        // segment.
        Diff = Absolute(Distance - SegEndDistance)
        if Diff < MINUMUM_SEGMENT_SEPARATION then
            // It belongs to the current segment
            SegEndDistance = Distance
```

```
                SegEndAngle = Angle
            **else**
                // The point is away form the current segment. Save
                // the current segment, and start a new one.
                SegStartDistanceList[SegIndex] = SegStartDistance
                SegStartAngleList[SegIndex] = SegStartAngle
                SegEndDistanceList[SegIndex] = SegEndDistance
                SegEndAngleList[SegIndex] = SegEndAngle
                increment SegIndex
                SegEndDistance = Distance
                SegEndAngle = Angle
                SegStartDistance = Distance
                SegStartAngle = Angle
            **end if** // Diff < MINUMUM_SEGMENT_SEPARATION
            increment ValueIndex
        **end while**
**end if** // SensorType is sonar or laser

// Next, find two segments that are closest to the robot but not next
// to each other. The opening is between two segments.
OpeningStartDistance = the first closest segment’s ending distance
OpeningStartAngle = the first closest segment’s ending angle
OpeningEndDistance = the second closest segment’s starting distance
OpeningEndAngle = the second closest segment’s starting angle
OpeningCenterDistance = (OpeningStartDistance + OpeningEndDistance)/2
OpeningCenterAngle = (OpeningStartAngle + OpeningEndAngle)/2

// Finally, perform the docking behavior [10, 13].
**if** OpeningCenterDistance > CONTROLLED_RADIUS **then**
```

```
        // Ballistic region
        Magnitude = MAXIMUM_MAGNITUDE * BALLISTIC_GAIN
        Angle = OpeningCenterAngle
        OutputX = Magnitude * cos(Angle)
        OutputY = Magnitude * sin(Angle)
    else
        // Controlled region
        DockAngle = an angle perpendicular to the line drawn from the
                    opening's start point to its end point
        AngleDiff = OpeningCenterAngle – DockAngle
        if Absolute(AngleDiff) < (CONE_ANGLE/2) then
            // Approach zone
            TangentWeight = Absolute/(CONE_ANGLE/2)
            MoveToWeight = 1.0
        else
            // Coercive zone
            TangentWeight = 1.0
            MoveToWeight = OpeningCenterDistance/CONTROLLED_RADIUS
        end if // Absolute(AngleDiff) < (CONE_ANGLE/2)

        // Construct the tangential vector
        Magnitude1 = MAXIMUM_MAGNITUDE * CONTROLLED_GAIN * TangentWeight
        if AngleDiff > 0 then
            Angle1 = OpeningCenterAngle + 90
        else
            Angle1 = OpeningCenterAngle - 90
        end if // AngleDiff > 0

        // Construct a move-to vector
```

```
      Magnitude2 = MAXIMUM_MAGNITUDE * CONTROLLED_GAIN * MoveToWeight
      Angle2 = OpeningCenterAngle

      // Sum the two vectors.
      OutputX = Magnitude1 * cos(Angle1) + Magnitude2 * cos(Angle2)
      OutputY = Magnitude1 * sin(Angle1) + Magnitude2 * sin(Angle2)
end if // OpeningCenterDistance > CONTROLLED_RADIUS
```

B.7.3 Move-Backward

```
// Generate a vector towards the direction that is opposite to the
// current heading.
OutputX = - MAXIMUM_MAGNITUDE
OutputY = 0
```

B.7.4 Move-Forward

```
// Generate a vector towards the current heading.
OutputX = MAXIMUM_MAGNITUDE
OutputY = 0
```

B.7.5 Move-Leftward

```
// Generate a vector towards the direction that is perpendicularly left
// of the current heading.
OutputX = 0
OutputY = MAXIMUM_MAGNITUDE
```

B.7.6 Move-Rightward

```
// Generate a vector towards the direction that is perpendicularly
// right of the current heading.
OutputX = 0
OutputY = - MAXIMUM_MAGNITUDE
```

B.7.7 Move-To-Big-Blob

```
// Generate a vector towards the biggest blob in the field of view.
if SensorType is a blob detector then
    MaxBlobSize = 0
    Index = 0
    while Index < the number of data points in the reading
        BlobSize = reading's values[Index]
        if BlobSize > MaxBlobSize then
            Angle = reading's phi_angles[Index]
            MaxBlobSize = BlobSize
        end if
        increment Index
    end while
    OutputX = MAXIMUM_MAGNITUDE * cos(Angle)
    OutputY = MAXIMUM_MAGNITUDE * sin(Angle)
end if // SensorType is a blob detector
```

B.7.8 Stop

```
// Generate a vector whose elements are all zeros.
OutputX = 0
OutputY = 0
```

B.7.9 Swirl-Obstacle

```
// Generate a vector in a direction that is perpendicular to the
// direction of the closest obstacle (tangential to the surface of the
// obstacle).
if SensorType is sonar or laser then
    Index = 0
    ClosestDistance = infinity
```

```
while Index < the number of data points in the reading
      Distance = reading's values[Index]
      if Distance < ClosestDistance then
            Angle = reading's phi_angles[Index]
      end if // Distance < ClosestDistance
      increment Index
end while

// Create the tangential vector
if Angle > 0 then
      Angle = Angle - 90 degree
else
      Angle = Angle + 90 degree
end if // Angle > 0

if Distance <= SAFETY_MARGINE then
      // The obstacle is within a safety margin. Make the vector
      // have the maximum strength.
      Magnitude = MAXIMUM_MAGNITUDE
else
      // The vector is stronger as the robot gets closer to the
      // obstacle.
      Magnitude = MAXIMUM_MAGNITUDE *
            (MAXIMUM_RANGE - Distance) /
            (MAXIMUM_RANGE - SAFETY_MARGINE)
end if // Distance <= SAFETY_MARGINE

// Finally, construct the output vector.
OutputX = Magnitude * cos(Angle)
```

```
    OutputY = Magnitude * sin(Angle)
end if // SensorType is sonar or laser
```

B.8 Behavioral Coordinator

B.8.1 Cooperative Coordinator

```
// Linear sum the output vector from each active motor schema. Save
// the output in the output data structure, Action (Table 24).
Action's speed_x = 0
Action's speed_y = 0
for each MotorSchema in the list of the active motor schemata
    Action's speed_x = Action's speed_x + MotorSchema's OutputX
    Action's speed_y = Action's speed_y + MotorSchema's OutputY
end for
```

B.8.2 Subsumptive Coordinator

```
// Assuming motor schemata are organized in hierarchical layers,
// pick the highest layer that contains active motor schemata, and use
// the output from this layer.
LayerNumber = the number of the top layer (i.e., the highest priority)
while LayerNumber >= 0
    Layer = the layer whose number is LayerNumber
    if there is active MotorSchema in Layer then
        // The highest active layer found. Use this output.
        Action's speed_x = sum all active motor schemata's OutputX
                           in this layer (cf. Section B.8.1)
        Action's speed_y = sum all active motor schemata's OutputY
                           in this layer (cf. Section B.8.1)
        end the for-loop
    end if
    decrement LayerNumber
end while
```

APPENDIX C

DATA

C.1 Experimental Constants

Table 37: The predefined constants utilized in the experiments in Chapter 5.

Description	Relevant Equation	Symbol	Experiment Number			
			1	2	3A	3B
TD(λ) learning rate	3.5	α	0.001	0.001	0.001	0.001
TD(λ) eligibility trace	3.5	λ	0.1	0.1	0.1	0.1
Recollection similarity function variance	3.14	—	1.0	1.0	1.0	25.0
Recollection similarity function threshold	3.15	θ_ρ	0.95	0.95	0.95	0.95
Sensor model similarity function variance	3.17	θ_Δ	1.0	25.0	1.0	25.0
The behaviroal discount factor for the motion model	3.18	κ_m	0.5	0.9	0.5	0.9
Localizer posterior entropy threshold	3.21	θ_H	10.0	2.5	10.0	2.5
Allowable dealy factor threshold	3.27	θ_Δ	—	5.0	—	2.0 (Full) 4.0 (Half)
Avoid-Obstacle safety margin (m)	—	—	1.0	1.5	1.0	—
Avoid-Obstacle sphere of influence (m)	—	—	5.0	5.0	5.0	—
Drive speed (m/s)	—	—	1.0	0.2	1.0	1.0 (Full) 0.5 (Half)
Steer speed (deg/s)	—	—	15.0	10.0	15.0	30.0 (Full) 15.0 (Half)

C.2 Experimental Results

C.2.1 Efficiency of the Foundational Data Structure

Table 38: Numerical results for the limited transitions vs. full transitions (Figure 44).

# of Events	Computation Time (ms)					
	Limited Transitions			Full Transitions		
	Total	Sensor Model	Trans. Model	Total	Sensor Model	Trans. Model
20	1.35	1.14	0.18	1.50	1.07	0.41
30	1.77	1.47	0.24	2.26	1.40	0.81
40	2.37	2.04	0.27	3.39	2.18	1.17
50	2.86	2.37	0.42	4.16	2.30	1.80
60	3.54	2.98	0.48	5.31	2.87	2.36
70	3.90	3.27	0.52	6.58	3.40	3.08
80	4.55	3.78	0.64	7.84	3.79	3.92
90	4.91	4.14	0.61	9.01	4.46	4.40
100	5.61	4.75	0.68	10.35	4.69	5.52
110	6.03	5.11	0.81	11.82	5.38	6.30
120	6.80	5.79	0.80	13.14	5.80	7.12
130	7.14	6.09	0.82	14.97	6.35	8.44
140	7.89	6.75	0.97	16.26	6.73	9.31
150	8.32	7.11	1.00	17.62	7.29	10.12
160	8.83	7.62	0.97	19.41	7.67	11.57
170	9.52	8.18	1.06	21.69	8.28	13.16
180	9.99	8.50	1.22	23.10	8.90	13.94
190	10.78	9.16	1.29	25.17	9.56	15.32
200	11.19	9.52	1.32	26.65	9.78	16.53

Table 39: The constants and correlation coefficients of the trendlines for the limited transitions vs. full transitions graph (Figure 44).

Term	Trendline Constants (n^2, n, n^0) and Squared Correlation Coefficient (R^2)					
	Limited Transitions			Full Transitions		
	Total	Sensor Model	Trans. Model	Total	Sensor Model	Trans. Model
n^2	0.0000	0.0000	0.0000	0.0003	0.0000	0.0003
n	0.0550	0.0473	0.0061	0.0769	0.0491	0.0315
n^0	0.1201	0.0511	0.0757	-0.2611	-0.0413	-0.4311
R^2	0.9990	0.9988	0.9839	0.9996	0.9980	0.9991

Table 40: Standard error measurements for the limited transitions vs. full transitions (Figure 44).

# of Events	Standard Error Measurement (ms)					
	Limited Transitions			Full Transitions		
	Total	Sensor Model	Trans. Model	Total	Sensor Model	Trans. Model
20	0.053	0.056	0.041	0.038	0.047	0.037
30	0.039	0.050	0.022	0.046	0.058	0.053
40	0.040	0.055	0.033	0.035	0.065	0.075
50	0.021	0.048	0.035	0.029	0.107	0.096
60	0.039	0.057	0.047	0.035	0.081	0.074
70	0.039	0.060	0.058	0.043	0.117	0.111
80	0.035	0.059	0.045	0.028	0.146	0.145
90	0.019	0.054	0.051	0.066	0.153	0.157
100	0.039	0.063	0.046	0.039	0.102	0.102
110	0.031	0.057	0.046	0.072	0.089	0.120
120	0.029	0.066	0.048	0.045	0.254	0.258
130	0.039	0.093	0.069	0.073	0.096	0.113
140	0.031	0.061	0.059	0.141	0.120	0.164
150	0.050	0.074	0.065	0.075	0.125	0.096
160	0.035	0.089	0.062	0.076	0.178	0.196
170	0.056	0.097	0.070	0.166	0.188	0.260
180	0.063	0.130	0.075	0.109	0.210	0.188
190	0.081	0.080	0.094	0.141	0.105	0.175
200	0.063	0.105	0.079	0.102	0.185	0.187

C.2.2 Effectiveness of Somatic Markers

Table 41: Sequences of box choices made by the robot with somatic markers.

Visit #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Test Set 1	A	A	C	B	C	C	C	C	C	C	D	C	D	C	D	D	D	D	D	D
Test Set 2	B	B	D	D	D	D	A	A	A	A	D	D	D	D	D	D	A	D	D	D
Test Set 3	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Test Set 4	A	A	D	D	A	C	A	C	A	A	C	C	C	C	C	C	C	C	C	C
Test Set 5	A	A	B	A	B	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Test Set 6	D	D	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Test Set 7	B	C	B	C	C	C	A	C	C	C	C	C	C	C	C	C	C	C	C	C
Test Set 8	C	C	A	B	B	A	D	C	C	C	C	C	A	D	A	B	C	B	B	B

Table 42: Sequences of box choices made by the robot without somatic markers.

Visit #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Test Set 1	C	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Test Set 2	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
Test Set 3	C	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Test Set 4	D	C	D	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
Test Set 5	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
Test Set 6	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Test Set 7	D	D	D	C	C	A	A	A	A	A	A	C	A	A	A	A	A	A	A	A
Test Set 8	D	C	C	C	D	D	D	B	B	D	B	B	B	A	B	B	B	B	B	B

Table 43: The rate of the robot taking advantageous choices with respect to the number of the trials (visit number) and the standard error measurements (S.E.M.) (Figure 48).

Visit Number		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Robot with Somatic Markers	Advantageous Choices (%)	38	50	63	63	63	88	63	88	75	75	100	100	88	100	88	88	88	88	88	88
	S.E.M.	18	19	18	18	18	13	18	13	16	16	0	0	13	0	13	13	13	13	13	13
Robot without Somatic Markers	Advantageous Choices (%)	63	38	38	25	25	13	13	0	0	13	0	13	0	0	0	0	0	0	0	0
	S.E.M.	18	18	18	16	16	13	13	0	0	13	0	13	0	0	0	0	0	0	0	0

Table 44: The numerical values and the standard error measurements (S.E.M.) of the average distribution of 20 consecutive box-visits over four box types (A, B, C, and D) by the robot with somatic markers (Figure 49).

Type	# of Selection Over 20 Trials	
	Mean	S.E.M.
A	2.63	0.80
B	1.63	0.71
C	10.50	2.74
D	5.25	2.11

Table 45: The numerical values and the standard error measurements (S.E.M.) of the average distribution of 20 consecutive box-visits over four box types (A, B, C, and D) by the robot without somatic markers (Figure 50)

Type	# of Selection Over 20 Trials	
	Mean	S.E.M.
A	9.13	3.41
B	8.50	3.36
C	1.13	0.44
D	1.25	0.67

Table 46: The numerical values of the difference between the robot with somatic markers and the robot without them in terms of its advantageous choices minus disadvantageous ones (Figure 51).

Type	# of Advantageous Minus # of Disadvantageous Choices	
	Mean	S.E.M.
With Somatic Marker	11.50	2.44
Without Somatic Marker	-15.38	2.20

C.2.3 Promptness of Proactive Behavior Computation

Table 47: The numerical values of the average computation time required for the event-matching process (and the standard error measurements) in the anticipatory experiment with respect to the number of episodes in the memory (Figure 56)

# of Episodes	Event-Matching Time (ms)			
	Limited History		Full History	
	Mean	S.E.M.	Mean	S.E.M.
5	30.80	0.48	30.58	0.54
6	30.54	0.53	37.03	0.46
7	32.52	0.39	44.42	0.44
8	32.52	0.48	51.95	0.76
9	34.47	0.68	59.10	0.83
10	35.15	0.72	68.13	0.73
11	35.30	0.35	76.67	1.30
12	34.41	0.55	85.90	0.81

Table 48: The numerical values of the average path length (and the standard error measurements) in the anticipatory experiment with respect to the number of episodes in the memory (Figure 57).

# of Episodes	Path Length (m)			
	Limited History		Full History	
	Mean	S.E.M.	Mean	S.E.M.
5	25.32	0.07	25.32	0.05
6	25.40	0.05	25.32	0.06
7	25.38	0.10	25.40	0.07
8	25.39	0.09	25.36	0.09
9	25.41	0.04	25.40	0.07
10	25.35	0.08	25.34	0.05
11	25.37	0.07	25.33	0.04
12	25.37	0.08	25.40	0.09

Table 49: The numerical values of the average duration (and the standard error measurements) in the anticipatory experiment with respect to the number of episodes in the memory (Figure 58).

# of Episodes	Duration (s)			
	Limited History		Full History	
	Mean	S.E.M.	Mean	S.E.M.
5	508.8	3.4	501.5	4.9
6	516.5	0.1	509.8	7.2
7	519.0	6.9	499.0	9.8
8	515.5	28.3	528.0	3.9
9	507.8	13.0	534.5	10.5
10	525.3	18.4	526.0	5.9
11	502.8	7.3	518.5	6.2
12	500.8	31.2	584.5	13.1

Table 50: The numerical values of the average computation time required for event matching and the average event-sampling interval in the improvisational experiment with respect to the number of episodes in the memory (Figure 61).

Cond.	# of Episodes	Avg. Event-Matching Time (ms)				Avg. Event-Sampling Interval (ms)	
		Full Speed		Half Speed			
		Mean	S.E.M.	Mean	S.E.M.	Mean	S.E.M.
I	3	358.48	9.75	375.15	7.17	492.17	13.18
II	5	616.30	33.90	389.84	15.22	495.79	10.83
III	7	674.97	61.48	408.71	12.93	501.55	12.04

Table 51: The numerical values of the time to reach the goal location (duration) in the improvisational experiment and their standard error measurements (Figure 60).

Duration of Successful Runs (s)			
Full Speed		Half Speed	
Mean	S.E.M.	Mean	S.E.M.
172.82	7.67	260.75	1.68

Table 52: The numerical values of the difference between the successful and unsuccessful runs in terms of the excessive event-matching time (Figure 62).

Runs	Excessive Event-Matching Time (ms)	
	Mean	S.E.M.
Success	-104.77	29.63
Failure	129.26	37.16

REFERENCES

[1] *Merriam-Webster's Collegiate Dictionary*, 10th ed. Merriam-Webster, Springfield, Mass., 1993.

[2] *Open Source Computer Vision Library: Referene Manual*. Intel Corporation, 2001.

[3] J. P. Aggleton and A. W. Young, "The Enigma of the Amygdala: on Its Contribution to Human Emotion," in *Cognitive Neuroscience of Emotion*, R. D. Lane and L. Nadel, eds., Oxford Univ. Press, New York, 2000, pp. 106-128.

[4] P. Agre, "The Dynamic Structure of Everyday Life," Massachusetts Inst. Technology, Technical Report AITR-1085, 1988.

[5] P. Agre and D. Chapman, "What Are Plans for?," in *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, P. Maes, ed., MIT Press, 1991, pp. 17-34.

[6] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, 1991, pp. 37-66.

[7] J. E. Anderson, *Constraint-Directed Improvisation for Everyday Activities*, PhD Dissertation, Dept. Computer Science, Univ. of Manitoba, 1995

[8] J. E. Anderson and M. Evans, "Constraint-Directed Improvisation for Complex Domains," *Proc. the 11th Biennial Conf. the Canadian Soc. for Computational Studies of Intelligence on Advances in Artificial Intelligence*, Toronto, Springer-Verlag, 1995, pp. 1-13.

[9] M. A. Arbib, "Schema Theory," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, ed., MIT Press, Cambridge, Mass., 1998, pp. 830-834.

[10] R. C. Arkin, *Behavior-Based Robotics*. MIT Press, Cambridge, Mass., 1998.

[11] R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *Int'l J. Robotics Research*, vol. 8, 1989, pp. 92-112.

[12] R. C. Arkin, "Moving Up the Food Chain: Motivation and Emotion in Behavior-based Robots," in *Who Needs Emotions: The Brain Meets the Robot*, J. Fellous and M. Arbib, eds., Oxford University Press, 2005, pp. 245-270.

[13] R. C. Arkin and R. R. Murphy, "Autonomous Navigation in a Manufacturing Environment," *IEEE Trans. Robotics and Automation*, vol. 6, 1990, pp. 445-454.

[14] R. C. Arkin, E. Riseman, and A. Hanson, "AuRA: An Architecture for Vision-based Robot Navigation," *Proc. DARPA Image Understanding Workshop*, Los Angeles, 1987, pp. 417-431.

[15] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally Weighted Learning," *Artificial Intelligence Review*, vol. 11, 1997, pp. 11-73.

[16] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally Weighted Learning for Control," *Artificial Intelligence Review*, vol. 11, 1997, pp. 75-113.

[17] D. Bailey, *Improvisation: Its Nature and Practice in Music*. Da Capo Press, 1993.

[18] C. Balkenius and B. Johansson, "Anticipatory Models in Gaze Control: a Developmental Model," *Cognitive Processing*, vol. 8, 2007, pp. 167-174.

[19] R. Bar-On, D. Tranel, N. L. Denburg, and A. Bechara, "Exploring the Neurological Substrate of Emotional and Social Intelligence," *Brain*, vol. 126, 2003, pp. 1790-1800.

[20] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins, "Learning and Sequential Decision Making," in *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, M. Gabriel and J. Moore, eds., MIT Press, Cambridge, Mass., 1990, pp. 539-602.

[21] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The Annals of Mathematical Statistics*, vol. 41, 1970, pp. 164-171.

[22] T. Bear, "Murphy's Law Was Invented Here," *Desert Wings*, 1978, pp. 3.

[23] A. Bechara, A. R. Damasio, H. Damasio, and S. W. Anderson, "Insensitivity to Future Consequences Following Damage to Human Prefrontal Cortex," *Cognition*, vol. 50, 1994, pp. 7-15.

[24] A. Bechara, D. Tranel, H. Damasio, and A. R. Damasio, "Failure to Respond Autonomically to Anticipated Future Outcomes Following Damage to Prefrontal Cortex," *Cerebral Cortex*, vol. 6, 1996, pp. 215-225.

[25] R. E. Bellman, *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[26] P. F. Berliner, *Thinking in Jazz: the Infinite Art of Improvisation*. The Univ. of Chicago Press, 1994.

[27] P. J. Best and A. M. White, "Placing Hippocampal Single-Unit Studies in a Historical Context," *Hippocampus*, vol. 9, 1999, pp. 346-351.

[28] M. Boddy and T. Dean, "Solving Time-Dependent Planning Problems," *Proc. Int'l Joint Conf. Artificial Intelligence*, 1989, pp. 979-984.

[29] C. Breazeal, "A Motivational System for Regulating Human-Robot Interaction," *Proc. Nat'l Conf. Artificial intelligence*, 1998, pp. 54-62.

[30] C. Breazeal, "Robot in Society: Friend or Appliance?," *Proc. Autonomous Agents Workshop on Emotion-Based Agent Architectures*, Seattle, 1999, pp. 18-26.

[31] R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE J. Robotics and Automation*, vol. 2, 1986, pp. 14-23.

[32] A. Bruce, J. Knight, S. Listopad, B. Magerko, and I. R. Nourbakhsh, "Robot Improv: Using Drama to Create Believable Agents," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2000, pp. 4002-4008.

[33] M. Bunsey and H. Eichenbaum, "Conservation of Hippocampal Memory Function in Rats and Humans," *Nature*, vol. 379, 1996, pp. 255-257.

[34] N. Burgess, S. Becker, J. A. King, and J. O'Keefe, "Memory for Events and Their Spatial Context: Models and Experiments," *Philosophical Trans. the Royal Soc.* , vol. 356, 2001, pp. 1493-1503.

[35] N. Burgess, J. G. Donnett, and J. O'Keefe, "Using a Mobile Robot to Test a Model of the Rat Hippocampus," *Connection Science*, vol. 10, 1998, pp. 291-300.

[36] N. Burgess, E. A. Maguire, and J. O'Keefe, "The Human Hippocampus and Spatial and Episodic Memory," *Neuron*, vol. 35, 2002, pp. 625-641.

[37] H. Burianova and C. L. Grady, "Common and Unique Neural Activations in

Autobiographical, Episodic, and Semantic Retrieval," *J. Cognitive Neuroscience*, vol. 19, 2007, pp. 1520-1534.

[38] J. Busby, Z. Parrish, and J. VanEenwyk, *Mastering Unreal Technology: The Art of Level Design*. Sams, Indianapolis, 2005.

[39] M. V. Butz and D. E. Goldberg, "Generalized State Values in an Anticipatory Learning Classifier System," in *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems*, vol. 2684, M. V. Butz, O. Sigaud, and P. Gerard, eds., Springer, Berlin, 2004, pp. 282-301.

[40] M. V. Butz and J. Hoffmann, "Anticipations Control Behavior: Animal Behavior in an Anticipatory Learning Classifier System," *Adaptive Behavior*, vol. 10, 2002, pp. 75-96.

[41] M. V. Butz, O. Sigaud, G. Pezzulo, and G. Baldassarre, "Anticipations, Brains, Individual and Social Behavior: An Introduction to Anticipatory Systems," in *Anticipatory Behavior in Adaptive Learning Systems*, vol. 4520, G. Baldassarre, G. Pezzulo, C. Balkenius, M. V. Butz, and M. Grinberg, eds., Springer, Berlin, 2007, pp. 1-18.

[42] W. H. Calvin, "Evolving Improvisational Intelligence," 1995; http://faculty.washington.edu/wcalvin

[43] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Bridging the Gap Between Simulation and Reality in Urban Search and Rescue," in *RoboCup 2006: Robot Soccer World Cup X*, vol. 4434, G. Lakemeyer, E. Sklar, D. G. Sorrenti, and T. Takahashi, eds., Springer, Berlin, 2007.

[44] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman, "Acting Optimally in Partially Observable Stochastic Domains," *Proc. Nat'l Conf. Artificial Intelligence*, 1994, pp. 1023-1028.

[45] A. R. Damasio, *Descartes' Error: Emotion, Reason, and the Human Brain*. G.P. Putnam, New York, 1994.

[46] A. R. Damasio, "The Somatic Marker Hypothesis and the Possible Functions of the Prefrontal Cortex," *Philosophical Trans. the Royal Soc. of London. Series B, Biological Sciences*, vol. 351, 1996, pp. 1413-1420.

[47] N. Dussault, "iRobot At a Glance," 2007; http://www.irobot.com.

[48] H. Eichenbaum, "Hippocampus: Cognitive Processes and Neural Representations that Underlie Declarative Memory," *Neuron*, vol. 44, 2004, pp. 109-120.

[49] H. Eichenbaum, "Is the Rodent Hippocampus Just for 'Place'?," *Current Opinion in Neurobiology*, vol. 6, 1999, pp. 187-195.

[50] H. Eichenbaum and N. J. Cohen, *From Conditioning to Conscious Recollection: Memory Systems of the Brain.* Oxford Univ. Press, 2001.

[51] H. Eichenbaum, P. Dudchenko, E. Wood, M. Shapiro, and H. Tanila, "The Hippocampus, Memory, and Place Cells: Is It Spatial Memory or a Memory Space?," *Neuron*, vol. 23, 1999, pp. 209-226.

[52] R. B. Emery, T., "Behavior-Based Control of a Non-Holonomic Robot in Pushing Tasks," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2001, pp. 2381- 2388.

[53] Y. Endo, "Anticipatory and Improvisational Robot via Recollection and Exploitation of Episodic Memories," *Proc. 2005 AAAI Fall Symp.: From Reactive to Anticipatory Cognitive Embodied Systems*, 2005, pp. 57-64.

[54] Y. Endo and R. C. Arkin, "Anticipatory Robot Navigation by Simultaneously Localizing and Building a Cognitive Map," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems* 2003, pp. 460-466.

[55] Y. Endo, D. C. MacKenzie, and R. C. Arkin, "Usability Evaluation of High-Level User Assistance for Robot Mission Specification," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 34, 2004, pp. 168-180.

[56] R. Falcone, "MIND RACES: from Reactive to Anticipatory Cognitive Embodied Systems," Information Society Technology, Mind RACES Periodic Management Report N1, 2006.

[57] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, vol. 2, 1971, pp. 189-208.

[58] N. J. Fortin, K. L. Agster, and H. B. Eichenbaum, "Critical Role of the Hippocampus in Memory for Sequences of Events," *Nature Neuroscience*, vol. 5, 2002, pp. 458-462.

[59] N. J. Fortin, S. P. Wright, and H. Eichenbaum, "Recollection-Like Memory Retrieval

in Rats is Dependent on the Hippocampus," *Nature*, vol. 431, 2004, pp. 188-191.

[60] D. J. Foster, R. G. M. Morris, and P. Dayan, "Models of Hippocampally Dependent Navigation, Using the Temporal Difference Learning Rule," *Hippocampus*, vol. 10, 2000, pp. 1-16.

[61] M. Fyhn, S. Molden, S. Hollup, M. Moser, and E. Moser, "Hippocampal Neurons Responding to First-Time Dislocation of a Target Object," *Neuron*, vol. 35, 2002, pp. 555-566.

[62] Georgia Tech Mobile Robot Laboratory, *MissionLab: User Manual for MissionLab 7.0*. College of Computing, Georgia Institute of Technology, Atlanta, Ga, 2006.

[63] A. K. Goel, K. S. Ali, M. W. Donnellan, A. G. d. S. Garza, and T. J. Callantine, "Multistrategy Adaptive Path Planning," *IEEE Expert*, vol. 9, 1994, pp. 57-65.

[64] A. K. Goel, E. Stroulia, Z. Chen, and P. Rowl, "Model-Based Reconfiguration of Schema-Based Reactive Control Architectures," presented at *AAAI Fall Symposium on Model-Directed Autonomous Systems*, Boston, 1997.

[65] M. Grachten, "JIG: Jazz Improvisation Generator," *Proc. Workshop on Current Research Directions in Computer Music*, 2001, pp. 1-6.

[66] A. Guazzelli, M. Bota, and M. A. Arbib, "Competitive Hebbian Learning and the Hippocampal Place Cell System: Modeling the Interaction of Visual and Path Integration Cues," *Hippocampus*, vol. 11, 2001, pp. 216-239.

[67] K. J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, 1989.

[68] M. E. Hasselmo, "The Role of Hippocampal Regions CA3 and CA1 in Matching Entorhinal Input with Retrieval of Associations Between Objects and Context: Theoretical Comment on Lee et al. (2005)," *Behavioral Neuroscience*, vol. 119, 2005, pp. 342-345.

[69] M. E. Hasselmo, E. Schnell, and E. Barkai, "Dynamics of Learning and Recall at Excitatory Recurrent Synapses and Cholinergic Modulation in Rat Hippocampal Region CA3," *J. Neuroscience*, vol. 15, 1995, pp. 5249-5262.

[70] B. Hayes-Roth and L. Brownston, "Multiagent Collaboration in Directed Improvisation," *Proc. the First Conf. Multi-Agent Systems*, San Francisco, 1994, pp. 148-154.

[71] R. Held, "Exposure-History as a Factor in Maintaining Stability of Perception and Coordination," *J. Nervous and Mental Disease*, vol. 132, 1961, pp. 26-32.

[72] J. Hoffmann, "Anticipatory Behavioral Control," in *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems*, vol. 2684, M. V. Butz, O. Sigaud, and P. Gerard, eds., Springer, Berlin, 2003, pp. 44-65.

[73] S. A. Hollup, S. Molden, J. G. Donnett, M. B. Moser, and E. I. Moser, "Accumulation of Hippocampal Place Fields at the Goal Location in an Annular Watermaze Task," *J. Neuroscience*, vol. 21, 2001, pp. 1635-1644.

[74] B. Hommel, J. Musseler, G. Aschersleben, and W. Prinz, "The Theory of Event Coding (TEC): a Framework for Perception and Action Planning," *Behavioral and Brain Sciences*, vol. 24, 2001, pp. 849-878.

[75] R. C. Honey, A. Watt, and M. Good, "Hippocampal Lesions Disrupt an Associative Mismatch Process," *J. of Neuroscience*, vol. 18, 1998, pp. 2226-2230.

[76] M. R. James, S. Singh, and M. L. Littman, "Planning with Predictive State Representations," *Proc. Int'l Conf. Machine Learning and Applications*, 2004, pp. 304-311.

[77] W. James, *The Principles of Psychology*. Dover, New York, 1950.

[78] B. Johansson and C. Balkenius, "An Experimental Study of Anticipation in Simple Robot Navigation," in *Anticipatory Behavior in Adaptive Learning Systems*, vol. 4520, G. Baldassarre, G. Pezzulo, C. Balkenius, M. V. Butz, and M. Grinberg, eds., Springer, Berlin, 2007, pp. 365-378.

[79] B. Johansson and C. Balkenius, "Robots with Anticipation and Attention," in *Advances in Artificial Intelligence in Sweden*, P. Funk, T. Rognvaldsson, and N. Xiong, eds., Malardalen Univ., Vasteras, Sweden, 2005, pp. 202-204.

[80] P. N. Johnson-Laird, "How Jazz Musicians Improvise," *Music Perception*, vol. 19, 2002, pp. 415-442.

[81] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, 1996, pp. 237-285.

[82] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *J. Basic Engineering*, vol. 82, 1960, pp. 35-45.

[83] B. Kernfeld, *What to Listen For in Jazz*. Yale Univ. Press, New Haven, 1995.

[84] Z. Kira and R. C. Arkin, "Forgetting Bad Behavior: Memory for Case-Based Navigation," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems*, 2004, pp. 3145-3152.

[85] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, 1983, pp. 671-680.

[86] K. Kiryazov, G. Petkov, M. Grinberg, B. Kokinov, and C. Balkenius, "The Interplay of Analogy-Making with Active Vision and Motor Control in Anticipatory Robots," in *Anticipatory Behavior in Adaptive Learning Systems*, vol. 4520, G. Baldassarre, G. Pezzulo, C. Balkenius, M. V. Butz, and M. Grinberg, eds., Springer, Berlin, 2007, pp. 233-253.

[87] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems*, 2004, pp. 2149- 2154.

[88] S. Koenig and R. Simmons, "Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models," in *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, D. Kortenkamp, R. Bonasso, and R. Murphy, eds., MIT Press, 1998, pp. 91-122.

[89] J. Kolodner, *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, Calif., 1993.

[90] J. L. Kolodner, *Retrieval and Organizational Strategies in Conceptual Memory: a Computer Model*. Yale Univ., New Haven, Conn., 1980.

[91] J. L. Kolodner and D. Leake, "A Tutorial Introduction to Case-Based Reasoning," in *Case-Based Reasoning: Experiences, Lessons and Future Directions*, D. Leake, ed., MIT Press, Cambridge, Mass., 1996, pp. 31-65.

[92] M. Kruusmaa, "Global Navigation in Dynamic Environments Using Case-Based

Reasoning," *Autonomous Robots*, vol. 14, 2003, pp. 71-91.

[93] B. Kuipers and Y. T. Byun, "A Robot Exploration and Mapping Strategy Based on Semantic Hierarchy of Spatial Representations," *J. Robotics and Autonomous Systems*, vol. 8, 1991, pp. 47-63.

[94] P. R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification and Adaptive Control*. Prentice-Hall, Upper Saddle River, N.J., 1986

[95] J. LeDoux, "Cognitive-Emotional Interactions: Listen to the Brain," in *Cognitive Neuroscience of Emotion*, R. D. Lane and L. Nadel, eds., Oxford Univ. Press, New York, 2000, pp. 129-155.

[96] J. E. LeDoux, *The Emotional Brain: the Mysterious Underpinnings of Emotional Life*. Simon & Schuster, New York, 1996.

[97] I. Lee, M. R. Hunsaker, and R. P. Kesner, "The Role of Hippocampal Subregions in Detecting Spatial Novelty," *Behavioral Neuroscience*, vol. 119, 2005, pp. 145-153.

[98] W. Y. Lee, *Spatial Semantic Hierarchy for a Physical Mobile Robot*, doctoral dissertation, Dep. Computer Sciences, The Univ. of Texas at Austin, Austin, 1996

[99] R. Lienhart and J. Maydt, "An Extended Set of Haar-Like Features for Rapid Object Detection," *Proc. Int'l Conf. Image Processing*, 2002, pp. 900-903.

[100] M. Likhachev, M. Kaess, and R. C. Arkin, "Learning Behavioral Parameterization Using Spatio-Temporal Case-based Reasoning," *Proc. IEEE Int'l. Conf. Robotics and Automation*, Washington, D.C., 2002, pp. 1282-1289.

[101] M. Littman, R. Sutton, and S. Singh, "Predictive Representations of State," *Advances in Neural Information Processing Systems*, vol. 14, 2002, pp. 1555-1561.

[102] M. L. Littman, N. Ravi, E. Fenson, and R. Howard, "An Instance-based State Representation for Network Repair," *Proc. Nat'l Conf. Artificial Intelligence*, 2004, pp. 287-292.

[103] J. Lovell and J. Kluger, *Apollo 13*. Houghton Mifflin, Boston, 2000.

[104] D. M. Lyons and M. A. Arbib, "A Formal Model of Computation for Sensory-Based Robotics," *IEEE Trans. Robotics and Automation*, vol. 5, 1989, pp. 280-293.

[105] E. A. Maguire, D. G. Gadian, I. S. Johnsrude, C. D. Good, J. Ashburner, R. S. J. Frackowiak, and C. D. Frith, "Navigation-Related Structural Change in the Hippocampi of Taxi Drivers," *Proc. Nat'l Academy of Sciences*, vol. 97, 2000, pp. 4398-4403.

[106] S. Massoud Amin and B. F. Wollenberg, "Toward a Smart Grid: Power Delivery for the 21st Century," *IEEE Power and Energy Magazine*, vol. 3, 2005, pp. 34-41.

[107] M. J. Mataric, "A Distributed Model for Mobile Robot Environment-Learning and Navigation," MIT Artificial Intelligence Laboratory, technical report AITR-1228, 1990.

[108] M. J. Mataric, "Navigation with a Rat Brain: a Neurobiologically-Inspired Model for Robot Spatial Representation," *Proc. Int'l Conf. Simulation of Adaptive Behavior*, MIT Press, 1990, pp. 169-175.

[109] A. K. McCallum, *Reinforcement Learning with Selective Perception and Hidden State*, PhD Dissertation, Dept. Computer Science, Univ. of Rochester, 1995

[110] R. A. McCallum, "Hidden State and Reinforcement Learning with Instance-Based State Identification," *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 26, 1996, pp. 464-473.

[111] B. L. McNaughton, C. A. Barnes, J. L. Gerrard, K. Gothard, M. W. Jung, J. J. Knierim, H. Kudrimoti, Y. Qin, W. E. Skaggs, M. Suster, and K. L. Weaver, "Deciphering the Hippocampal Polyglot: the Hippocampus as a Path Integration System," *J. Experimental Biology*, vol. 199, 1996, pp. 173-185.

[112] D. Mendonca, "Decision Support for Improvisation in Response to Extreme Events: Learning from the Response to the 2001 World Trade Center Attack," *Decision Support Systems*, vol. 43, 2007, pp. 952-967.

[113] D. J. Mendonca and W. A. Wallace, "A Cognitive Model of Improvisation in Emergency Management," *IEEE Trans. Systems, Man and Cybernetics, Part A*, vol. 37, 2007, pp. 547-561.

[114] M. J. Milford, G. F. Wyeth, and D. Prasser, "RatSLAM: a Hippocampal Model for

Simultaneous Localization and Mapping," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2004, pp. 403-408.

[115] T. M. Mitchell, *Machine Learning*. McGraw-Hill, New York, 1997.

[116] M. C. Moraes and A. C. da Rocha Costa, "Improvisational Multi-Agent Organization: Using Director Agent to Coordinate Improvisational Agents," *Proc. Int'l Conf. Intelligent Agent Technology*, 2005, pp. 463- 466.

[117] R. Morris, "Developments of a Water-Maze Procedure for Studying Spatial Learning in the Rat," *J. Neuroscience Methods*, vol. 11, 1984, pp. 47-60.

[118] R. G. M. Morris, "Spatial Memory and the Hippocampus: the Need for Psychological Analyses to Identify the Information Processing Underlying Spatial Learning," in *Perception, Memory and Emotion: Frontiers in Neuroscience*, *Pergamon Studies in Neuroscience*, T. Ono, B. L. McNaughton, S. Molotchnikoff, E. T. Rolls, and H. Nishijo, eds., Elsevier Science, Oxford, 1996, pp. 319-342.

[119] E. I. Moser and O. Paulsen, "New Excitement in Cognitive Space: Between Place Cells and Spatial Memory," *Current Opinion in Neurobiology*, vol. 11, 2001, pp. 745-751.

[120] L. Moshkina, Y. Endo, and R. C. Arkin, "Usability Evaluation of an Automated Mission Repair Mechanism for Mobile Robot Mission Specification," *Proc. Proc. the 1st ACM SIGCHI/SIGART Conf. Human-Robot Interaction*, Salt Lake City, Utah, 2006, pp. 57-63.

[121] L. Nadel and H. Eichenbaum, "Introduction to the Special Issue on Place Cells," *Hippocampus*, vol. 9, 1999, pp. 341-345.

[122] J. O'Keefe, "Do Hippocampal Pyramidal Cells Signal Non-Spatial as Well as Spatial Information?," *Hippocampus*, vol. 9, 1999, pp. 352-364.

[123] J. O'Keefe and L. Nadel, *The Hippocampus as a Cognitive Map*. Clarendon Press, Oxford, 1978.

[124] L. E. Parker, "ALLIANCE: an Architecture for Fault Tolerant Multirobot Cooperation," *IEEE Trans. Robotics and Automation*, vol. 14, 1998, pp. 220-240.

[125] I. P. Pavlov, *Conditioned Reflexes: an Investigation of the Physiological Activity of the Cerebral*

Cortex, G. V. Anrep ed. Dover, New York, 1960.

[126] G. Pezzulo, G. Baldassarre, M. V. Butz, C. Castelfranchi, and J. Hoffmann, "From Actions to Goals and Vice-Versa: Theoretical Analysis and Models of the Ideomotor Principle and TOTE," in *Anticipatory Behavior in Adaptive Learning Systems*, vol. 4520, G. Baldassarre, G. Pezzulo, C. Balkenius, M. V. Butz, and M. Grinberg, eds., Springer, Berlin, 2007, pp. 73-93.

[127] G. Pezzulo and G. Calvi, "Schema-Based Design and the AKIRA Schema Language: An Overview," in *Anticipatory Behavior in Adaptive Learning Systems*, vol. 4520, G. Baldassarre, G. Pezzulo, C. Balkenius, M. V. Butz, and M. Grinberg, eds., Springer, Berlin, 2007, pp. 128-152.

[128] J. Piaget, *The Psychology of Intelligence*. Routledge & Paul, London, 1951.

[129] J. Pineau, G. Gordon, and S. Thrun., "Point-Based Value Iteration: An Anytime Algorithm for POMDPs," *Proc. Int'l Joint Conf. Artificial Intelligence*, 2003, pp. 1025-1032.

[130] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards Robotic Assistants in Nursing Homes: Challenges and Results," *Robotics and Autonomous Systems*, vol. 42, 2003, pp. 271-281.

[131] A. Ploghaus, I. Tracey, S. Clare, J. S. Gati, J. N. P. Rawlinsdagger, and P. M. Matthews, "Learning about Pain: the Neural Substrate of the Prediction Error for Aversive Events," *Proc. the Nat'l Academy of Sciences*, vol. 97, 2000, pp. 9281-9286.

[132] J. Pressing, "Cognitive Processes in Improvisation," *Cognitive Processes in the Perception of Art*, 1984, pp. 345-363.

[133] J. Pressing, "Improvisation: Methods and Models," *Generative Processes in Music*, 1987, pp. 129-178.

[134] J. Pressing, "Psychological Constraints on Improvisational Expertise and Communication," in *In the Course of Performance: Studies in the World of Musical Improvisation*, B. Nettl and M. Russell, eds., Univ. Chicago Press, 1998, pp. 47-67.

[135] L. Rabiner and B. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine*, vol. 3, 1986, pp. 4-16.

[136] A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark, "Case-Based Reactive Navigation: a Case-Based Method for On-Line Selection and Adaptation of Reactive Control Parameters in Autonomous Robotic Systems," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 27, 1997, pp. 376-394.

[137] A. Ram and J. C. Santamaria, "Continuous Case-Based Reasoning," *Artificial Intelligence*, vol. 90, 1997, pp. 25-77.

[138] A. D. Redish and D. S. Touretzky, "Cognitive Maps Beyond the Hippocampus," *Hippocampus*, vol. 7, 1997, pp. 15-35.

[139] A. D. Redish and D. S. Touretzky, "Modeling Interactions of the Rat's Place and Head Direction Systems," *Advances in Neural Information Processing Systems*, vol. 8, 1996, pp. 61-67.

[140] A. D. Redish and D. S. Touretzky, "The Role of the Hippocampus in Solving the Morris Water Maze," *Natural Computation*, vol. 10, 1998, pp. 73-111.

[141] E. T. Rolls, S. M. Stringer, and T. P. Trappenberg, "A Unified Model of Spatial and Episodic Memory," *Proc. the Royal Soc. B*, vol. 269, 2002, pp. 1087-1093.

[142] R. Ros, R. Lopez de Mantaras, C. Sierra, and J. L. Arcos, "A CBR System for Autonomous Robot Navigation," in *Artificial Intelligence Research and Development*, vol. 131, *Frontiers in Artificial Intelligence and Applications*, B. Lopez, J. Melendez, P. Radeva, and J. Vitria, eds., IOS Press, Amsterdam, 2005, pp. 299-306.

[143] R. Rosen, *Anticipatory Systems: Philosophical, Mathematical, and Methodological Foundations*, 1st ed. Pergamon Press, Oxford, 1985.

[144] A. Samsonovich and B. L. McNaughton, "Path Integration and Cognitive Mapping in a Continuous Attractor Neural Network Model," *J. Neuroscience*, vol. 17, 1997, pp. 5900-5920.

[145] J. C. Santamaria, R. S. Sutton, and A. Ram, "Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces," *Adaptive Behavior*, vol. 6, 1997, pp. 163-217.

[146] T. Sawada, T. Takagi, Y. Hoshino, and M. Fujita, "Learning Behavior Selection Through Interaction Based on Emotionally Grounded Symbol Concept," *Proc. Int'l Conf. Humanoid Robots*, Los Angeles, 2004, pp. 450-469.

[147] S. Schaal and C. G. Atkeson, "Robot Juggling: Implementation of Memory-Based Learning " *IEEE Control Systems Magazine*, vol. 14, 1994, pp. 57-71.

[148] R. C. Schank, *Dynamic Memory Revisited.* Cambridge Univ. Press., Cambridge, 1999.

[149] R. C. Schank, *Dynamic Memory: a Theory of Reminding and Learning in Computers and People.* Cambridge Univ. Press., Cambridge, 1982.

[150] N. A. Schmajuk, "The Psychology of Robots," *Proc. the IEEE*, vol. 84, 1996, pp. 1553-1561.

[151] N. A. Schmajuk, "Role of the Hippocampus in Temporal and Spatial Navigation: an Adaptive Neural Network," *Behavioural Brain Research*, vol. 39, 1990, pp. 205-229.

[152] N. A. Schmajuk and A. D. Thieme, "Purposive Behavior and Cognitive Mapping: a Neural Network Model," *Biological Cybernetics*, vol. 67, 1992, pp. 165-174.

[153] W. Schultz, "Predictive Reward Signal of Dopamine Neurons," *J. Neurophysiology*, vol. 80, 1998, pp. 1-27.

[154] W. Schultz, P. Dayan, and P. R. Montague, "A Neural Substrate of Prediction and Reward," *Science*, vol. 275, 1997, pp. 1593-1599.

[155] C. E. Schwartz, C. I. Wright, L. M. Shin, J. Kagan, and S. L. Rauch, "Inhibited and Uninhibited Infants 'Grown Up': Adult Amygdalar Response to Novelty," *Science*, vol. 300, 2003, pp. 1952-1953.

[156] W. B. Scoville and B. Milner, "Loss of Recent Memory after Bilateral Hippocampal Lesions," *J. Neurology, Neurosurgery, and Psychiatry*, vol. 20, 1957, pp. 11-21.

[157] C. E. Shannon, "Prediction and Entropy of Printed English," *The Bell System Technical Journal*, vol. 30, 1950, pp. 50-64.

[158] M. L. Shapiro and H. Eichenbaum, "Hippocampus as a Memory Map: Synaptic Plasticity and Memory Encoding by Hippocampal Neurons," *Hippocampus*, vol. 9, 1999, pp. 365-384.

[159] M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. Isbell, and A. Ram, "Transfer

Learning in Real-Time Strategy Games Using Hybrid CBR/RL," *Proc. Int'l Joint Conf. Artificial Intelligence*, Hyderabad, India, 2007, pp. 1041-1046.

[160] P. E. Sharp, "Complimentary Roles for Hippocampal versus Subicular/Entorhinal Place Cells in Coding Place, Context, and Events," *Hippocampus*, vol. 9, 1999, pp. 432-443.

[161] L. Shastri, "A Computational Model of Episodic Memory Formation in the Hippocampal System," *Neurocomputing*, vol. 38-40, 2001, pp. 889-897.

[162] S. Singh, M. R. James, and M. R. Rudary, "Predictive State Representations: A New Theory for Modeling Dynamical Systems," *Proc. Uncertainty in Artificial Intelligence*, 2004, pp. 512-519.

[163] S. Singh, M. Littman, N. Jong, D. Pardoe, and P. Stone, "Learning Predictive State Representations," *Proc. Int'l Conf. Machine Learning*, 2003, pp. 712-719.

[164] R. Smith, " Open Dynamics Engine: v0.5 User Guide," 2006; http://www.ode.org.

[165] L. R. Squire, "Memory and the Hippocampus: a Synthesis from Findings with Rats, Monkeys, and Humans," *Psychological Review*, vol. 99, 1992, pp. 195-231.

[166] L. R. Squire and S. Zola-Morgan, "The Medial Temporal Lobe Memory System," *Science*, vol. 253, 1991, pp. 1380-1386.

[167] W. Stolzmann, "An Introduction to Anticipatory Classifier Systems," in *Learning Classifier Systems: From Foundations to Applications*, P. L. Lanzi, W. Stolzmann, S. W. Wilson, R. E. Smith, A. Bonarini, T. Kovacs, R. L. Riolo, and L. B. Booker, eds., Springer, Berlin, 2000, pp. 175-194.

[168] W. Stolzmann, "Latent Learning in Khepera Robots with Anticipatory Classifier Systems," *Proc. Genetic and Evolutionary Computation Conf. Workshop Program*, 1999, pp. 290-297.

[169] A. Stoytchev and R. C. Arkin, "Combining Deliberation, Reactivity, and Motivation in the Context of a Behavior-Based Robot Architecture," *Proc. IEEE Int'l Symp. Computational Intelligence in Robotics and Automation*, Banff, Alberta, 2001, pp. 290-295.

[170] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine*

Learning, vol. 3, 1988, pp. 9-44.

[171] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an Introduction*. MIT Press, Cambridge, Mass., 1998.

[172] R. S. Sutton and A. G. Barto, "Time-Derivative Models of Pavlovian Reinforcement," in *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, M. Gabriel and J. Moore, eds., MIT Press, Cambridge, Mass., 1990, pp. 497-537.

[173] G. Tesauro, "Programming Backgammon Using Self-Teaching Neural Nets," *Artificial Intelligence*, vol. 134, 2002, pp. 181–199.

[174] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, 1995, pp. 58-68.

[175] G. Theocharous and S. Mahadevan, "Approximate Planning with Hierarchical Partially Observable Markov Decision Process Models for Robot Navigation," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2002, pp. 1347-1352.

[176] S. Thrun, "Monte Carlo POMDPs," *Proc. Advances in Neural Information Processing Systems 12*, MIT Press, 2000, pp. 1064-1070.

[177] S. Thrun, "Probabilistic Algorithms in Robotics," in *AI Magazine*, vol. 21, 2000, pp. 93-109.

[178] S. Thrun, "Robotic Mapping: A Survey," in *Exploring Artificial Intelligence in the New Millenium*, Morgan Kaufmann, 2002.

[179] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, Cambridge, Mass., 2005.

[180] E. C. Tolman, "Cognitive Maps in Rats and Man," in *Behavior and Psychological Man*, Univ. of California Press, 1951.

[181] E. C. Tolman, *Purposive Behavior in Animals and Men*. Univ. of California Press, 1951.

[182] E. C. Tolman and C. H. Honzik, "'Insight' in Rats," *Univ. California Publications in Psychology*, vol. 4, 1930, pp. 215-232.

[183] N. Tomatis, I. Nourbakhsh, and R. Siegwart, "Hybrid Simultaneous Localization and Map Building: a Natural Integration of Topological and Metric," *Robotics and Autonomous Systems*, vol. 44, 2003, pp. 3-14.

[184] E. Tulving, "Episodic and Semantic Memory," in *Organization of Memory*, E. Tulving and W. Donaldson, eds., Academic Press, New York, 1972.

[185] E. Tulving, "How Many Memory Systems Are There?," *American Psychologist*, vol. 40, 1985, pp. 385-398.

[186] E. Tulving, "Neurocognitive Processes of Human Memory," in *Basic Mechanisms in Cognition and Language with Special Reference to Phonological Problems in Dyslexia*, *Wenner-Gren International*, C. v. Euler, I. Lungberg, and R. R. Llinas, eds., Elsevier, Amsterdam, 1998.

[187] E. Tulving, H. J. Markowitsch, F. I. M. Craik, R. Habib, and S. Houle, "Novelty and Familiarity Activations in PET Studies of Memory Encoding and Retrieval," *Cerebral Cortex*, vol. 6, 1996, pp. 71-79.

[188] University of Birmingham, "Hippocampus," 2002; http://www.neuroscience.bham.ac.uk/neurophysiology/research/hippocampus.htm.

[189] H. Van de Water and J. C. Willems, "The Certainty Equivalence Property in Stochastic Control Theory," *IEEE Trans Automatic Control*, vol. 26, 1981, pp. 1080-1087.

[190] J. D. Velasquez, "An Emotion-Based Approach to Robotics," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems*, 1999, pp. 235-240.

[191] M. M. Veloso and J. G. Carbonell, "Derivational Analogy in PRODIGY: Automating Case Acquisition, Storage, and Utilization," *Machine Learning*, vol. 10, 1993, pp. 249 - 278.

[192] O. S. Vinogradova, "Hippocampus as Comparator: Role of the Two Input and Two Output Systems of the Hippocampus in Selection and Registration of Information," *Hippocampus*, vol. 11, 2001, pp. 578-598.

[193] H. Voicu and N. Schmajuk, "Exploration, Navigation and Cognitive Mapping," *Adaptive Behavior*, vol. 8, 2000, pp. 207-224.

[194] E. von Holst, "Relations between the Central Nervous System and the Peripheral Organs," *The British J. Animal Behavior*, vol. 2, 1954, pp. 89-94.

[195] G. Weinberg and S. Driscoll, "Toward Robotic Musicianship," *Computer Music J.*, vol. 30, 2006, pp. 28-45.

[196] G. Weinberg, M. Godfrey, A. Rae, and J. Rhodes, "A Real-Time Genetic Algorithm in Human-Robot Musical Improvisation," *Proc. Int'l Computer Music Conf.*, Copenhagen, 2007, pp. 192-195.

[197] M. A. Wheeler, "Episodic Memory and Autonoetic Awareness," in *The Oxford Handbook of Memory*, E. Tulving and F. I. M. Craik, eds., Oxford University Press, Oxford, 2000, pp. 597-608.

[198] E. R. Wood, P. A. Dudchenko, R. J. Robitsek, and H. J. Eichenbaum, "Hippocampal Neurons Encode Information about Different Types of Memory Episodes Occurring in the Same Location," *Neuron*, vol. 27, 2000, pp. 623-633.

[199] C. I. Wright, H. Fischer, P. J. Whalen, S. C. McInerney, L. M. Shin, and S. L. Rauch, "Differential Prefrontal Cortex and Amygdala Habituation to Repeatedly Presented Emotional Stimuli," *Neuroreport*, vol. 12, 2001, pp. 379-383.

[200] B. Young and H. Eichenbaum, "What Do Hippocampal Neurons Encode?," in *Perception, Memory and Emotion: Frontiers in Neuroscience*, *Pergamon Studies in Neuroscience*, T. Ono, B. L. McNaughton, S. Molotchnikoff, E. T. Rolls, and H. Nishijo, eds., Elsevier Science, Oxford, 1996, pp. 229-249.

[201] S. Zilberstein, "Using Anytime Algorithms in Intelligent Systems," in *The AI magazine*, vol. 17, 1996, pp. 73-83.

[202] S. Zilberstein and S. Russell, "Approximate Reasoning Using Anytime Algorithms," in *Imprecise and Approximate Computation*, vol. 318, S. Natarajan, ed., Kluwer Academic, Boston, 1995, pp. 43-62.