

ABSTRACT

MU, WEI. A Schematic Representation for Cognitive Tool-Using Agents. (Under the direction of Dr. Robert St. Amant).

In artificial intelligence (AI) research, embodied systems have received increasing attention since the 1990s. How to bridge the gap between raw sensorimotor data and symbolic representation in a robotic agent is still an open question. The research described in this document is inspired by theories in cognitive science, such as concept theory and embodied realism, as well as work in robotics and AI. The general goal of this research is to build a system capable of acquiring and maintaining semantic knowledge for higher-level reasoning, in particular reasoning about the use of tools, from the embodied experience of a cognitive agent in a simulated environment or in the real world. This research addresses cognitive theories of embodiment, the design of a general computational architecture, and the design and implementation of AI techniques for solving tool-using problems. One of the major contributions of this research is to provide a computational architecture for an embodied agent that can capture semantic relations from its interactions with the world, sufficient to support effective tool use both in short-term predictions and plan generation. As a result, we have implemented an example of this architecture in an Action Schema Generator, or ASG, which can automatically generate production rules and symbolic representations from a simulated agent's embodied experience without losing the capability of transferring the knowledge backwards to its original numerical sensorimotor format. We have developed pragmatic methods to evaluate the performance of ASG, at the component level and the system level, in simulated and real scenarios, for tasks with and without tools. We also have compared our design with other robotics and cognitive architectures, including behavior-based robotics, Neuroevolution, and psychologically inspired architectures. We believe that our work can provide a general foundation for embodied agents, and should be useful in future research.

A Schematic Representation for Cognitive Tool-Using Agents

by
Wei Mu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2009

APPROVED BY:

Dr. Robert St. Amant
Committee Chair

Dr. Ronald P. Endicott

Dr. James C. Lester

Dr. R. Michael Young

DEDICATION

To my parents, *Bo Yan* and *Fangsu Mu*,
my wife, *Hong Tian*,
for their endless love and support.

To my soon to be born son.

BIOGRAPHY

Wei Mu was born in Tianjin, China in 1979. He has received his Bachelor degree in the Department of Computer Science and Technology from Tsinghua University in 2001. He continued his study in the same department and has received his Master degree in 2004. Since then, he came to United States for further studies and has been a full-time Ph.D. student in the Computer Science Department of North Carolina State University.

ACKNOWLEDGMENTS

I would like to thank Dr. Robert St. Amant for giving his expert guidance throughout my research. It was my early working experience with him in Information Sciences Institute that led me into a whole new area in the research of Artificial Intelligence. Besides that, he's taught me many valuable things about teaching, learning, research and life that I may not have otherwise learned. I am also very grateful for his kindness, patience, financial support and encouragement.

My thanks also go to Dr. James Lester, Dr. Michael Young, and Dr. Ronald Endicott for their continuous assistance regarding both my research and this dissertation. I learned a lot from their invaluable courses. Their profound knowledge, enthusiasm about research, and hardworking spirit inspire me all the time in pursuing a doctoral degree.

I want to thank all my friends in the Knowledge Discovery Laboratory, for their many helps to my research. I also like to thank all my Chinese friends in Raleigh, NC, for all the happy time we have spent together.

This dissertation would not be possible without their help and support.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES.....	viii
Chapter 1 Introduction.....	1
Chapter 2 A Motivating Example.....	7
Chapter 3 A Cognitive Foundation for Tool Use	14
3.1 Concept Theory: Philosophical Foundations	15
3.2 Concept Theory: Embodied Reasoning	18
3.2.1 Image Schemas	18
3.2.2 Body schemas	21
3.3 Mental imagery	23
3.4 Summary	26
Chapter 4 Cognitive Robotic Architectures.....	27
4.1 Examples of cognitive robotic architectures	27
4.2 Design Challenges for a Cognitive Robotic Architecture.....	30
4.2.1 System Inputs and Outputs	32
4.2.2 System Components	33
Chapter 5 System Description.....	36
5.1 Time Series Segmentation from Sensory Data	36
5.1.1 Piecewise Linear Representation	37
5.1.2 Algorithm Implementation	40
5.1.3 Coarse Selection	44
5.2 Prototype Construction	46
5.2.1 Concept Categorization	46
5.2.2 CLIQUE algorithm.....	47
5.2.3 Modified CLIQUE Algorithm	48
5.2.4 Prototype Definition and Description.....	54
5.3 Action Schema Discovery.....	57
5.3.1 Knowledge for Learning.....	58
5.3.2 Action Schemas	58

5.3.3 Schema Generation.....	59
Chapter 6 Evaluation	63
6.1 Simulation Scenarios.....	65
6.2 Evaluation of Time Series Segmentation.....	70
6.3 Evaluation of Prototype Construction.....	76
6.4 Planning Program and System Evaluation.....	79
6.4.1 Evaluation on Scenario 1 – Approaching to a target	82
6.4.2 Evaluation on Scenario 2 – Turning towards a target.....	85
6.4.3 Evaluation on Combining the Scenario 1 and 2.....	86
6.4.4 Evaluation on Scenario 3 - Pushing a target off a box.....	89
6.4.5 Evaluation on Scenario 4 - Stretching to push a target.....	93
6.4.6 Evaluation on Scenario 5 - Detecting a target	96
6.4.7 Evaluation on Scenario 6 – Sweeping a target.....	98
6.5 Summary	103
Chapter 7 Integration and Comparison.....	105
7.1 Comparison with Behavior-based Robotics.....	105
7.2 Comparison with Neural Networks and Neuroevolution.....	107
7.3 Comparison with ACT-R and Soar	110
Chapter 8 Conclusion	112
Bibliography	115
Appendix	121

LIST OF TABLES

Table 5.1 t-function values	40
Table 5.2 Algorithm of regression analysis.....	41
Table 5.3 Algorithm for the segment test.....	41
Table 5.4 Algorithm of coarse selection.....	45
Table 5.5 Algorithm of CLIQUE	48
Table 5.6 Algorithm of build density space	49
Table 5.7 Algorithm of cluster growth	53
Table 5.8 Prototype Definition.....	55
Table 5.9 An example of prototype data	56
Table 5.10 An example of symbolic representation	57
Table 5.11 Structure of the action schema	59
Table 5.12 Approching action	59
Table 5.13 Symbol representation after differential analysis	60
Table 5.14 Algorithm of building action scehma.....	61
Table 5.15 Example of two action schemas	62
Table 6.1 Summary of data trials	66
Table 6.2 Time series segmentation errors of scenario 1	74
Table 6.3 Validity measures between CLIQUE and modified algorithm	78
Table 6.4 Average validity ratios between modified algorithm and CLIQUE.....	79
Table 6.5 Algorithm of forward planning	81
Table 6.6 Action plan of approaching to a target	82
Table 6.7 Action plan of turning towards a target.....	85
Table 6.8 Action plan of combining approaching and turning.....	87
Table 6.9 Action plan of pushing a target	92
Table 6.10 Maximal distances in succussful plan	93
Table 6.11 Action plan of combining pushing and stretching.....	95
Table 6.12 Action plan of sweeping a target off the corner	100
Table 6.13 Action plan of sweeping a target towards corner	101

LIST OF FIGURES

Figure 2.1 Canis in Breve simulation	8
Figure 2.2 Sony Aibo (ERS-7)	9
Figure 2.3 Canis approaching to poke a ball	10
Figure 2.4 Generation of action schemas from sensorimotor data	13
Figure 3.1 OUT Schema.....	20
Figure 3.2 A robot arm's original icon.....	23
Figure 3.3 Reaching a target with a stick	23
Figure 3.4 Examples of mechanical reasoning problems	24
Figure 3.5 Mental scanning experiment	25
Figure 4.1 Madcat architecture.....	28
Figure 4.2 Demonstration of Madcat.....	29
Figure 4.3 Reasoning consciousness model	30
Figure 4.4 System architecture	35
Figure 5.1 Example of white noise.....	39
Figure 5.2 Primitive analysis on Aibo approaching a ball	42
Figure 5.3 Original sensor data of Aibo approaching a ball.....	43
Figure 5.4 Regression result of Aibo approaching a ball	43
Figure 5.5 Ball's position in original data	44
Figure 5.6 Reducing the details in the regression result.....	45
Figure 5.7 An example of a density space.....	50
Figure 5.8 Contour map on threshold 1.....	51
Figure 5.9 Contour map on threshold 4.....	51
Figure 5.10 Contour map in modified algorithm.....	52
Figure 6.1 Example of raw data	66
Figure 6.2 Simulation pictures of scenario 1.....	67
Figure 6.3 Simulation pictures of scenario 2.....	67
Figure 6.4 Simulation pictures of scenario 3.....	68
Figure 6.5 Simulation pictures of scenario 4.....	69

Figure 6.6 Simulation pictures of scenario 5.....	69
Figure 6.7 Simulation pictures of scenario 6.....	70
Figure 6.8 Approaching the target in center front	71
Figure 6.9 Distance and direction data from scenario 1	72
Figure 6.10 Comparison between raw data and recovered data	73
Figure 6.11 Combination of the raw data and recovered data	74
Figure 6.12 Time series segmentation errors.....	75
Figure 6.13 Time series segmentation errors in Sony Aibo robot.....	76
Figure 6.14 Cohesion and separation	77
Figure 6.15 Density spaces with different partition size	79
Figure 6.16 Action space of Scenario 1.....	83
Figure 6.17 Successful areas of making contact.....	84
Figure 6.18 Successful areas of making contact with closest match	84
Figure 6.19 Successful areas of turning towards the target	86
Figure 6.20 Successful areas of combining approaching and turning	88
Figure 6.21 Pushing the target on a small box	90
Figure 6.22 Pushing the target on a large box.....	90
Figure 6.23 Pushing the target with a stick	91
Figure 6.24 Action space of scenario 3	92
Figure 6.25 Stretch to push the target.....	94
Figure 6.26 Stretch to push the target with stick.....	94
Figure 6.27 Detecting the object in front of the box	97
Figure 6.28 Detecting no object in front of the box	97
Figure 6.29 Action plan of scenario 5	98
Figure 6.30 Sweeping the target off the corner	99
Figure 6.31 Sweeping the target towards the corner	99
Figure 6.32 Successful area of sweeping the ball off the corner	102
Figure 6.33 Successful area of sweeping the ball towards the corner.....	102

Chapter 1

Introduction

Building intelligent agents, whether robots or computational systems, is a core pursuit for many researchers in artificial intelligence. In classical AI research, intelligence is viewed as a matter of abstract symbol processing (Pfeifer and Bongard, 2007). It is the algorithm or program that matters, roughly speaking, in the classical view. However, when the focus shifts to the interaction with the real world, as is necessary for robotic systems, classical approaches have “failed to deepen our understanding of many intelligent processes” (Pfeifer and Bongard, 2007). Difficulties arise in the areas of computer vision, robot object manipulation, locomotion, common sense understanding of natural environments, and many other areas. Ironically, it’s areas that humans normally consider difficult — playing chess, applying rules of logic, probing mathematical theorems — that are considered successes for classical artificial intelligence, but reproducing our natural and effortless everyday behaviors, like seeing, hearing, speaking, riding a bicycle, has proved notoriously hard.

The concept of embodied cognition was developed against this background, in the mid-1980s, in both cognitive science and artificial intelligence. One perspective is represented by a series of books by George Lakoff and Mark Johnson (Lakoff and Johnson, 1980; Lakoff, 1987; Johnson, 1987; Lakoff and Johnson, 1999), which focus on aspects of human cognition not previously recognized as being important. Lakoff and Johnson argue that human rationality is embodied, that the nature of the human mind is largely determined by the form of the human body. Bodily experience influences what and how things can be meaningful for us, the way we comprehend and reason about the world, and the actions we take. The evidence Lakoff and Johnson bring to bear can

be found in studies of human conceptual structure formation, linguistic performance, mental activities, and so forth.

In artificial intelligence, especially in robotics, classical approaches have been heavily reliant on sophisticated engineering models of the outside world. In other words, building a robot system requires programmers to understand the conditions the robot will encounter, and then to spell out all of the relevant information that is needed for the system to generate appropriate responses and actions (Coward, 2006). Unfortunately, this design task is very expensive. Often it is too expensive to be done for what we might call semantically deep tasks. Also, because hand-built systems often lack the capability of representing the “meanings” of their internal models (in the reflective sense that human beings can reason *about* as well as *with* their concepts), robotic systems can find it difficult to check the results and maintain the appropriate correspondences between their internal models and the external world (Cohen and Beal, 2000).

Rodney Brooks brought his robotic embodiment perspective into artificial intelligence around the same time as Lakoff and Johnson’s work, with a series of papers and books on both technology and philosophy (Brooks, 1999). Brooks opposes the conventional approach to pre-storing knowledge in designing robotic agents. Instead, he has built a series of robots based on a so-called subsumption architecture and opened up a new area of designing robotic agents.

Since this work in the 1980s, a great deal of research has been based on an embodied intelligence perspective. Of the work most relevant to this dissertation, Terry Regier used a connectionist network and sequences of pictures to train the system learning the spatial relations concepts, like *in*, *out*, *from*, *to*, *on*, *off*, *front*, *back*, *above*, and *below* (Regier, 1996). Arrabales Moreno and Sanchis de Miguel proposed a computational model that can provide the functions of consciousness, and they implemented some of the functions in a navigation system (Moreno and de Miguel, 2006). Lewis and Luger designed a novel architecture based on Brooks’s subsumption architecture and the Copycat program, which can develop a representation of its environment through a continuing interaction with it (Lewis and Luger, 2000).

This research follows similar steps associated with the embodiment perspective. Specifically, this research takes as a fundamental principle the idea that knowledge about the world should be acquired from the robotic agent’s experience, not given by the designer. In other words, the goal of this research is to design a computational system that can automatically discover useful knowledge from a physical agent’s sensorimotor experience of interacting with the world, in order to support

higher-level reasoning. If this is possible, we can argue that the results of the agent's processing are meaningful to the agent itself, and that this meaning is derived from the agent's embodiment and its interactions with the world. Compared to classical approaches, the aim of this research is shifted from directly engineering the knowledge about the world into the system to the design of a general architecture that can discover knowledge through the agent's embodied experience.

Cognitive agents capable of general problem solving can be enormously complex, even given historical limitations on results that AI systems have achieved. Our goal is more modest than the construction of such an agent. The focus of the work is on a specific range of intelligent behaviors associated with the use of tools. The goal of the project is to have a general-purpose framework that can lead to effective tool-using behavior.

Why tool use? Development of general tool users is one of the key challenges for AI (Nilsson, 1995). Representing, classifying and recognizing tools by their functionality can provide us new opportunities for understanding and eventually improving an agent's interaction with the physical world (Bicici, 2003). Further, because tool use is a capacity of many animals as well as human beings, we have a wide range of possible targets for building a tool-using agent: a robot tool user might have a pre-programmed behavior at its disposal, like some species of wasp (Oswalt, 1973); it might be capable of a small range of tool-using tasks that can be flexibly adapted to the environment, like orangutans (van Schaik et al., 2003); or it might be a general tool user like human beings (St. Amant and Wood, 2005). There are many research projects in AI that have studied embodied intelligence and its application in robotic tool use (Stoytchev, 2003; Acosta-Calderon and Hu, 2005). This research will focus more the relationship between the embodied agent and its representational structures, including what can be learned about general cognition via the development of a tool-using agent in simulation and in a physical environment.

The general approach in this work will be to develop a computational system that will act as a component in some cognitive architecture. For generality we leave this architecture unspecified, but plausible candidates include general-purpose planning and reasoning systems, within the field of AI, cognitive architectures such as Soar and ACT-R, within the field of cognitive science, and other architectures that might be pursued in either of these disciplines. This cognitive system component targets one of the critical aspects of functionality for a realistic cognitive agent: linking physical sensorimotor data to symbolic concepts that describe actions and their effects in an environment.

The system described in this dissertation is designed to meet several criteria that, we believe, make it effective:

- It should reflect a specific theory of concepts. Concepts have a long history and a great deal of empirical information supporting different theories of how they are formed, maintained, and applied.
- It should respect plausible cognitive constraints (Chapman, 1991). Because our work is inspired by comparable behaviors over a wide range of biological “platforms”, our interest is in tool-using capabilities that fall roughly within the same design space as biological organisms.
- Representations of actions should be explainable and, in principle, open to analysis by the agent maintaining such representations. This means that numerical techniques will be judged at least partly on their being amenable to analysis.

To summarize, the main contributions of this research will be with respect to a general component for a computational architecture that can capture embodied relations from the interactions with the world, sufficient to support effective tool use. This is the foundation for the design of any embodied agent, and should be useful for future research. The component will have the following capabilities:

1. The automatic identification of similar patterns of behavior from numerical sensorimotor data over time, in the form of conceptual prototypes.
2. The capacity to apply such prototypes toward the classification of new data; i.e., the component will be capable of providing short-term predictions for its physical interactions.
3. The use of such prototypes in the automatic identification of schematic actions.
4. The capacity to support a higher-level reasoning or planning system to effectively solve problems in a given environment.

The resulting software, which we describe as an *Action Schema Generator*, or ASG, will be validated on a set of robotic tool-use scenarios, mainly in simulation but in some cases using data from a physical robot system. Evaluation is a difficult issue for cognitive agents, especially agents developed to solve problems for which there are no well-established standard computational solutions, and for which there is little to no quantitative data on solutions carried out by humans or other biological agents. For the research problem addressed here, we adopt a two-stage evaluation

process that we believe is common in comparable research projects. First, we carry out an evaluation of the components of the ASG, concentrating on an outline of their capabilities and limitations with respect to performance. Second, we carry out a system-level evaluation, demonstrating that the integration of components in a complete agent is sufficient to solve a range of problems of interest.

One of the critical issues in building a component of a cognitive system is what we can learn about the nature of problem solving during its evaluation. We believe that the following insights into the nature of tool use are worth highlighting.

The scenarios that are part of the system-level evaluation of our work include tasks that require tool use as well as task that do not. Some of the scenarios provide two different versions (tool use and non-tool use) of what is essentially the same problem. We find no hard boundaries between in the representations that the agent derives for the tool use and non-tool use cases. In fact, the use of a tool can generally be seen as imposing a transformation on streams of information about the preconditions or effects of actions. This observation is consistent with conventional views of tool use in human cognitive research, in which a tool is viewed as amplifying some action or as an extension of the body.

The agent described in this work does not carry out an explicit comparison between the representations it develops for tool-using and non-tool-using tasks, in order to make the transformation between pairs of behaviors explicit. Nevertheless there are clear implications that are laid out for a hypothetical higher-level reasoning system to take advantage of. Two different types of effects in tool use can be seen in our evaluation. For an example of the first type, consider the behavior of reaching out, either holding or not holding a stick. The agent we have built generates nearly identical representations for these two behaviors, with the only difference being the distance from a targeted object that can be reached. The implication is that some tools provide a kind of enhancement or transformation of the capabilities of the agent. For an example of the second type, consider the behavior of blind person using a stick to tap the ground while walking. Two analogous behaviors in the agent we have built, simulating two functionalities of a stick tool – probing and striking, are based on representations that the agent is incapable of developing in the absence of a tool. This observation suggests that some tools provide novel capabilities to the agent. Making such distinctions clear can potentially help in designing a higher-level reasoning system of the tool-using agent.

The remainder of this dissertation is structured as follows. Chapter 2 gives a motivating example of the behavior of a simulated tool-using agent we have built, called Canis. Chapter 3 provides an overview of topics in cognitive science related to concept formation and manipulation. Chapter 4 describes relevant work on cognitive architectures, with a focus on the requirements briefly laid out above for the generation of action schemas from quantitative sensorimotor data in a single cognitive architecture component. Chapter 5 describes the design and implementation of the cognitive architecture component, and Chapter 6 runs through its evaluation.

Chapter 2

A Motivating Example

One of the focus points of this research is on tool-using tasks. Tool use is commonly associated with human intelligence, but it can also be seen in a range of animal species with widely differing cognitive capabilities and architectures. If this research is to act as a step toward the development of general tool-using agents, it is reasonable to ask what the scope of capabilities of such agents should be.

A few catalogs of animal tool-using behaviors have been developed, for example for chimpanzees (Whiten et al., 1999) and orangutans (Krutzen et al., 2005); Beck (Beck 1981) provides a book-length survey of tool use across animal species. St. Amant et al. (in preparation) have turned these textual descriptions of behavior into a more formal reference set of tool-using tasks for robotic agents; their set covers the following types of tools, each accompanied by a set of characteristic behaviors for using the tool type:

- Sponging tools.
- Probing tools.
- Collecting tools.
- Pounding tools.
- Wiping tools.
- Levering tools.
- Striking tools.
- Whisking and brushing tools.

- Dabbing tools.
- Tickling tools.
- Combing tools.
- Hooking tools.
- Digging tools.
- Containing tools.
- Perforating tools.
- Protective gear.

One of the most common goals in the use of these tools, even of differing types, is to reach objects that are otherwise inaccessible: they might be out of reach in free space, enclosed in a hole with a small opening, embedded in a surface, placed in a location awkward to reach (e.g., even on the inside of an agent), and so forth.

To motivate the research in this dissertation, it will be useful to walk through an example of tool-using behavior as an illustration, so that different aspects of the behavior can be highlighted along with their relationship to intelligence. We will describe a simple reaching task for a robot agent, with a stick that acts as a surrogate for a probing, collecting, or dabbing tool.

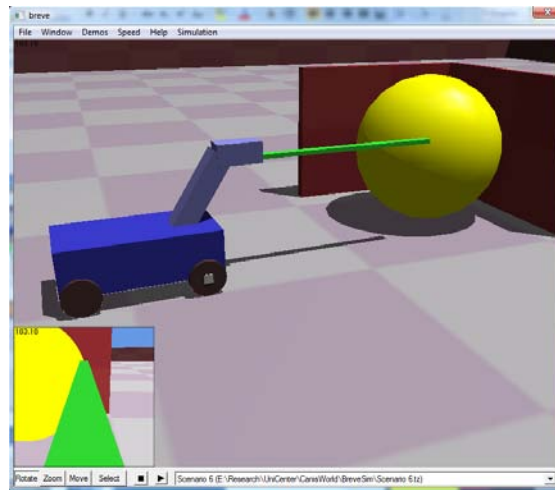


Figure 2.1 Canis in Breve simulation

Most of the research in this project, for practical reasons, has been carried out in simulation. One of the environments we use for training and evaluation is Breve (Klein, 2002), shown in Figure 2.1. Breve is a free, open-source software package that provides 3D simulations of multi-agent systems and artificial life. Breve's simulation engine allows agents to behave in a physically

realistic fashion, which means that agents in the simulated world can be configured to behave largely as real objects do, following the laws of physics.

The agent implemented in the Breve simulation is called Canis. It is designed as a simplified on-wheel version of Sony Aibo (ERS-7), a compact general-purpose dog-like robot (Figure 2.2). Such a design, in the simulation, is to provide compatibility for future research. Although Breve is not a real robotics system, it is an efficient platform for system prototyping and demonstration. Similar to the Aibo, the Canis simulation agent is capable of moving forward and backward, and turning (using its front wheels). Canis also has a head-neck structure that can stretch forward or downward, and it has a simulated camera to capture the first-person view shown in the left-bottom of the simulation window. In the tool-using tasks, a stick can be attached to Canis' head, which simulates a stick being held in the Aibo's mouth. This level of simulation is not detailed enough to capture grasping behavior or fine motor control over manipulated objects, but it should be sufficient for an exploration of tool-using behaviors.



Figure 2.2 Sony Aibo (ERS-7)

In our example scenario¹, a yellow ball has been placed on a brown box in front of Canis, as shown in Figure 2.3. Canis is capable of distinguishing the ball, its target, from the box, which is just an object in the environment. Canis begins in a mode that involves no reasoning; it is carrying out behaviors that allow it to gather and organize information about the effects of its actions on the environment. Over a series of trials, Canis will move directly toward the ball to see what happens.

¹ This is also the second scenario used in the system evaluation described in Chapter 6.

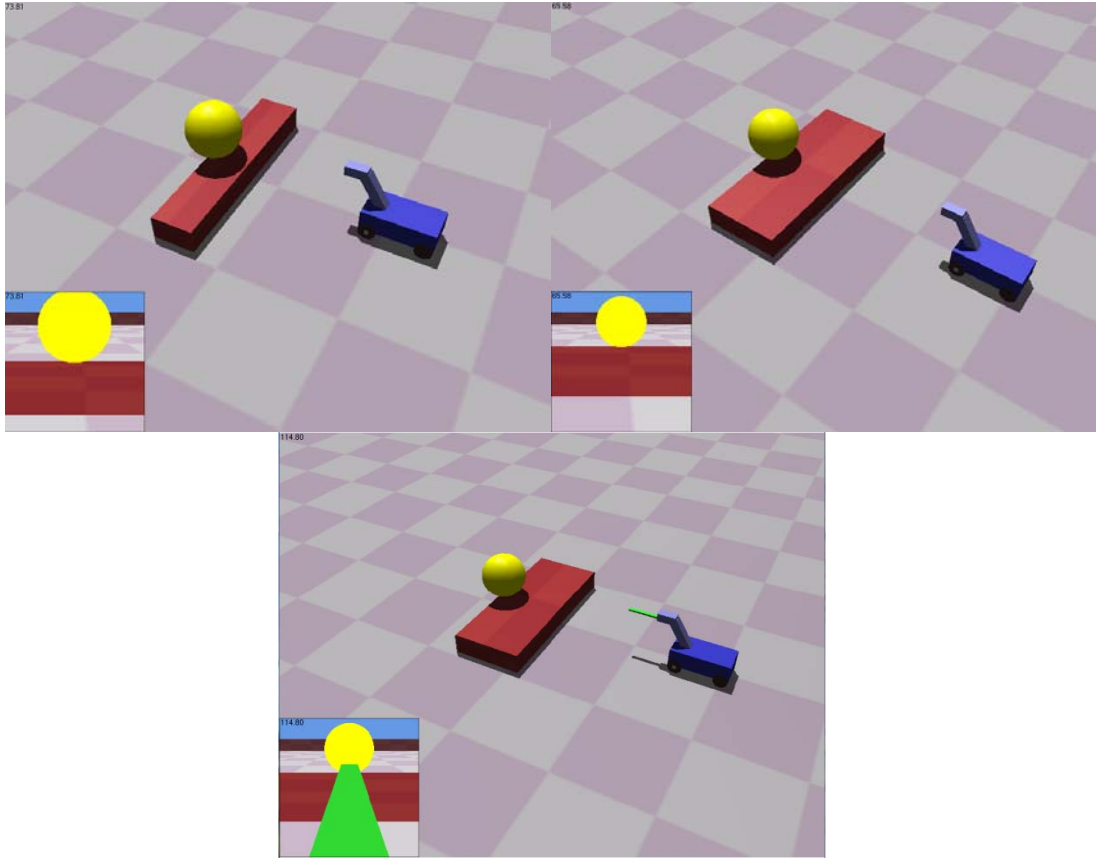


Figure 2.3 Canis approaching to poke a ball

a small box (upper left); a large box (upper right); Canis with stick with a large box (bottom)

There are three different variations in the settings of this scenario; we call each such variation a *situation*. In the first situation, the ball is placed on the box close enough to the edge facing Canis that when Canis approaches it can touch the ball with its nose. When this contact happens, the ball is pushed off the box onto the floor. The trial stops, with this result counted as a success. In the second situation, the ball is sufficiently far from the edge of the box that when Canis approaches, it is blocked from reaching the ball by the edge of the box. Canis is on wheels and cannot climb, and so this trial ends in failure. In the third situation, Canis is given a stick to hold in its mouth, pointing directly ahead. Here when Canis approaches the ball, even if the ball were at the same location as in the second situation, Canis is able to contact the ball with the end of the stick and push it onto the floor, a success. There are two types of minor random variations added for each trial (these do not result in different situations, but can be thought of as variability within a given situation). One type of variation is added intentionally to simulate repeated experiments — for example, Canis begins at

a different distance from the box and acts at a different speed each time; another type of variation is automatically added by the physical engine in Breve, which will produce noise for actions, which sometimes may even affect the outcomes of the simulation.

This might seem to be a trivial set of situations for Canis to represent and distinguish between; it is the kind of thing that human beings do naturally and without thinking about. Nevertheless we put Canis under a number of constraints, not simply to make its job more difficult, but to pose challenges that we believe are comparable to those solved by real animals, even very simple animals, as well as by humans.

First, the information that Canis takes in about these situations comes in the form of streams of un-interpreted continuous data representing both sensory and motor information. There are potentially many such streams in any such situation or scenario. Here there are ten: different control signals are applied; the distances to either the ball or the box are monitored; directions are monitored; and so forth. How should Canis identify relationships across the streams as well as between different parts of single and multiple streams over time?

Second, if Canis is to learn from its experiences in these scenarios then it must have some way of encapsulating general information about them, for future re-use. As with streams of data that may exhibit variation, Canis faces the problem of being able to distinguish between scenarios that are in some sense similar to or different from each other. How should this information be represented and organized?

In answering these questions, we follow the guidance of theoretical and practical experience in cognitive science and artificial intelligence (areas we discuss in more detail in Chapter 3). Broadly speaking, the goal of this research is to translate sensorimotor data into structured representations of Canis's actions and their effects on the environment. This happens in several stages.

First, we must segment continuous signals into separate intervals, and apply appropriate analysis in order to translate them into some form where the intervals can be compared. For example, when Canis is not moving, its distance from a given location remains constant, but if Canis begins to move toward that location, its distance decreases. Such qualitative descriptions of distances as “unchanging” and “decreasing” need to be mapped consistently onto actual data to produce discretized data.

Performing such an analysis on intervals is not enough. It often happens that relationships hold across different sensory streams over a period of time, which means that we must find invariant features over the discretized transformations of individual variables. Here is the idea of *concepts* comes into play. As human beings, we form concepts during repeated experiences in order to categorize, reason about, and communicate the meaning of those experiences. We will rely on clustering techniques for this purpose, without looking deeply into the abstract nature of concepts. By applying appropriate clustering techniques, we should be able to identify the relevant concepts from repeated trials in this scenario, despite variation across the trials.

Automatically generated concepts are not simply for show; they will anchor key points over time in trial data. Translating from discretized trial data into concept-based trial data provides opportunities for the analysis of variation in the data. Essentially, we want to derive what we view as the most important information for any robot agent, knowledge about actions, in a form that the agent can reuse and potentially generalize. We rely on action schemas to represent this information, which captures the pre-conditions and effects of an action, as is common in AI reasoning and planning. Thus the last step of the analysis is to organize concept data into action schemas and identify the most important action schemas in a scenario.

Figure 2.4 illustrates the entire process of translating sensorimotor data into concept-based data and generating action schemas.

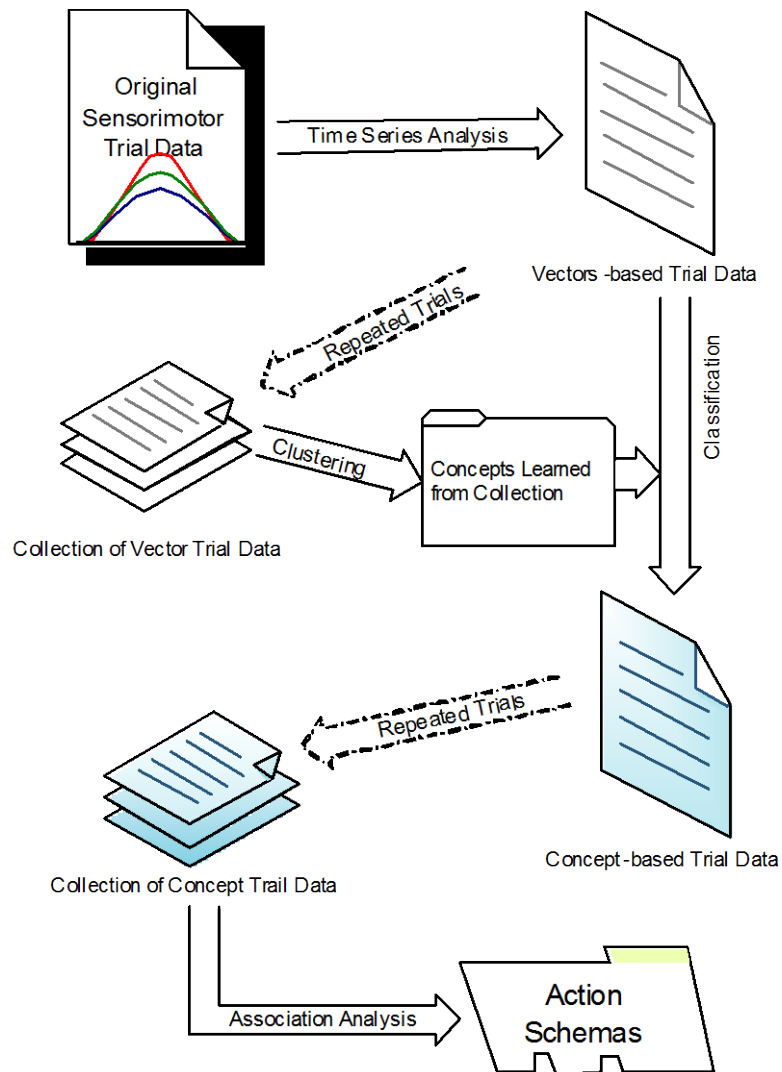


Figure 2.4 Generation of action schemas from sensorimotor data

Chapter 3

A Cognitive Foundation for Tool Use

The ability to form and reason with concepts is critical to intelligent behavior. It is difficult to explain how some decision or course of action is rational without the assignment of objects, properties, and actions to discrete categories or concepts. Even when we simply speak of reasoning about something, we rely on concepts. Unfortunately for us, the philosophical foundations of concepts remain unresolved, even after millennia of consideration. Nevertheless it is possible, even based on a partial and incomplete account, to ground intelligent behavior in the development and manipulation of computational objects that we believe are sufficiently like concepts to act as surrogates.

This chapter gives a brief history and selective overview of concept theory, from a philosophical as well as cognitive science perspective. The goal is to motivate specific design decisions for a cognitive architecture adequate to support intelligent tool-using behavior, by reference to findings in the literature. The relevant observations can be summarized as follows:

1. Concepts need not have explicit (symbolic, propositional) definitions.
2. What a concept represents (its content) may depend on the embodiment and experience of the agent that holds that concept.
3. Some limited set of common schematic relationships may be applicable in a general way over the experiences of an agent.

4. In the application of concepts to new situations, especially in situations that require deciding what to do, simulation of the concept in its application to the situation is a plausible part of understanding the situation.

The remainder of this chapter contains a brief survey of concept theory, expanding on the points above and identifying some of their implications.

3.1 Concept Theory: Philosophical Foundations

The traditional view of concepts can be traced back to Plato and Socrates (Thagard, 2005), in what has come to be known as the *classical theory* of concepts. According to classical theory, concepts are complex mental representations whose structure encodes a specification of necessary and sufficient conditions for their own application. In other words, concepts have a definitional structure. One example is the concept *bachelor*, which can be defined as *unmarried adult male*.

Based on this definitional structure, classical theory explains categorization as a process of checking all the constituents of a concept as applied to an object in question. If every property of the object satisfies the criteria, the concept may be applied; if at least one doesn't, then the concept may not be applied. This theory can also explain the learning process in much the same way as categorization – but it runs in reverse. That is, learning, on this view, is a constructive operation. One simply adds features to the definition in light of new experience.

Most of the problems of classical theory derive from its commitment to definitions. It's obvious that not all concepts have a clear definition. A familiar way to illustrate the difficulty is to ask, "Is the Pope a bachelor?" The Pope, the leader of the Catholic Church, meets the criteria above, but many people would not find the concept of bachelor naturally applicable (Pinker, 1999). Laurence and Margolis (Laurence and Margolis, 2003) go further: "Notice, however, that if a concept has a definition, this definition will strongly constrain theoretical developments in science and place a priori limits on what we are capable of discovering about the world. ... A definition may appear to capture the structure of a concept, but the appearance may only be an illusion which later discoveries help us to see beyond."

Classical theory also predicts that larger definitions should be more difficult for learning and categorization, because of the large number of conditions that need to be considered. This proves to be not always true as in the experiments, *bachelor* is not more difficult to learn than *unmarried*, and *father* is not more difficult to learn than *parent* (Fodor, 1998). Another problem with concept

definition is that people may apply concepts to counterfactual situations where the associated features of classical theory do not apply (Kripke, 1980). For example, people may still apply the concept *gold* to a piece of metal even if it is no longer yellow, in a phenomenon known as *atypical judgment*.

Despite its limitations, classical theory dominated concept theory until the 1970s, when serious alternatives began to be discussed. One result was a new theory, *prototype theory* (Rosch and Mervis, 1975). Prototype theory gives up on the idea that a concept's internal structure provides its definition. Instead, prototype theory holds that a concept structure encodes a set of features that the members of this concept tend to have. In simple form, the feature list is the result of a statistical analysis over the members of that concept. Also in contrast with classical theory, for an object to be in the extension of a concept it need not satisfy each and every property encoded in the concept's structure, as long as it satisfies a sufficient number of them.

Two major advances were made by prototype theory. First, prototype theory doesn't require that concepts have explicit pre-supplied definitions. Instead, a "self-generated" feature list can give the concept the explanatory ability. Because this list can be self-generated from samples of a population of concept members, it provides flexibility. Second, in prototype theory, categorization is taken to be a feature-matching process where an exemplar or individual is compared to a target category for how similar they are. As long as enough features match, the item can be judged to fall into the category. Unlike classical theory, prototype theory doesn't require all the features to be matched (Pinker, 1999).

Prototype theory can solve some problems faced by classical theory. Moreover, it has empirical support from experiments. Among them, experiments show that human perform better in categorization tasks with instances that rank high on a prototypical scale with respect to a given concept. To explain this, Smith and Medin provided an "accumulator model" in which, the item with more features in common will reach the critical value for a concept more quickly, and when the critical value is reached, the system will judge the item belongs to the concept immediately (Smith and Medin, 1981).

Another source of empirical support for prototype theory is *typicality effects*. Typicality effects are a variety of psychological phenomena related to the fact that people willingly rate subcategories for how typical or representative they are for a given category (Laurence and Margolis, 2003). For example, subjects tend to say that robins are better examples of the category

bird than chickens are; i.e., they say robins are more "typical" birds. According to Eleanor Rosch, categories may have hierarchical structures, which are called super-ordinate, basic, and subordinate categories (Rosch and Lloyd, 1978). The distribution of properties of categories is predicted by independent typicality rankings. The more typical a subordinate is judged to be, the more properties it will share with other exemplars of the same category. For instance, robins are taken to have many of the same properties as other birds, and, correspondingly, robins are judged to be highly typical birds; in contrast, chickens are taken to have fewer properties in common with other birds, and chickens are judged to be less typical birds. Another finding is that typicality has a direct reflection in categorization. In cases where subjects are asked to judge whether an *X* is a *Y*, independent measures of typicality predict the speed of correct affirmatives. Subjects are quicker in their correct response to "Is a robin a bird?" than to "Is a chicken a bird?" Error rates, as well, are predicted by typicality. The more typical the probe, relative to the target category, the fewer errors a subject will make.

However, prototype theory has its own difficulties (Laurence and Margolis, 2003). The first problem is that the prototype theory is subject to the problems of ignorance and error, just like classical theory. People can possess a concept and yet have erroneous information about the items in its extension or lack sufficient correct information to pick them out uniquely. The second problem is that many concepts may simply lack prototypes for many reasons; some concepts may be unfamiliar, for example, while others may simply be un-instantiated (Fodor, 1998). Lacking prototypes means that people may have concepts without any statistical beliefs (feature list) or even members. A third problem is that prototypes don't appear to compose in accordance with the principles of a compositional semantics. Fodor illustrates the argument with the concept *pet fish*. The *pet* prototype encodes properties that are associated with dogs and cats, and the *fish* prototype encodes properties that are associated with things like trout, yet the *pet fish* prototype encodes properties that are associated with goldfish and other small colorful fish. So it's hard to see how the prototype for *pet fish* could be computed from the prototypes for *pet* and *fish*.

Other theories have appeared more recently, such as *concept atomism*, proposed by Jerry Fodor (Fodor, 1998). Concept atomism is contrary to the theories above in that it doesn't admit to concepts having structure. Instead, it treats concepts as semantic primitives. Central to concept atomism is the thesis that a concept's content is not determined by its relation to any other particular concepts. Instead, it is determined by a mind-world relation, that is, a causal or historical relation

between the symbol and what it represents. In concept atomism, a concept is activated when and only when it judges the intentional object to be present. Feature lists are part of the perceptual detection mechanisms that facilitate the process whereby symbol becomes activated in the system, but they are not essential to the symbol. For example, birds might be identified by contingently related features, like *feathers*, *flies*, and *song*, but these features are not part of the concept *bird*.

The most important contribution of concept atomism is that it changes the explanatory role of concepts from internal structures to external relationships. This change means the theory can easily solve the publicity and constancy problem, which indicates that concepts may change from time to time or from person to person by confining idiosyncrasies to the perceptual detection mechanisms and broader theoretical information outside the concept. Concept atomism can also solve the problem of atypical judgments, including factual judgments and counterfactual judgments, by claiming that those features are not part of the essence of the concept.

3.2 Concept Theory: Embodied Reasoning

Various different threads of thought have been based on the ideas and results described in the previous section. Those threads relevant to this dissertation fall under a general category that we may call *embodied reasoning*. By “embodied reasoning” in this context, we mean an account of concepts that recognizes concepts are developed and maintained by human beings who are embodied and situated in the physical world. The centrality of embodiment has implications for a number of cognitive abilities, including representation, learning, and reasoning.

3.2.1 Image Schemas

Mark Johnson and George Lakoff raised the idea that bodily experience influences what and how things can be meaningful for human beings, the way human comprehend and reason about the world, and the actions human take (Lakoff and Johnson, 1980; Johnson, 1987; Lakoff and Johnson, 1999). This opened up a new perspective on human cognition, and has received a good deal of attention in various research areas, including artificial intelligence (Rakova, 2002).

To explain how embodied experience functions in our life, two types of imaginative structures are presented in Johnson and Lakoff’s theory: *image schemas* and *metaphorical projection*. The notion of “schema” is based on Immanuel Kant’s work. For Kant, schemas are structures of the imagination; that is, schemas are fixed templates superimposed onto perceptions and conceptions to render meaningful representations (Oakley, 2007). In cognitive linguistics, the

term “image” implicates perception in all acts of conceptualization. Concepts develop from representation of a perceptual conglomeration of multiple senses, including visual, auditory, haptic, motor, olfactory, and gustatory experiences. Nevertheless, according to Johnson, image schemas are neither propositional, nor rich concrete images, nor mental pictures. An image schema is a recurring, dynamic pattern of our perceptual interactions and motor programs that gives coherence and structure to our experience (Johnson, 1987). Thousands of times each day we feel, see, or manipulate different kinds of image schemas, like containment (houses, cars, and boxes are all containers), like force (which causes a door to stay open or results from contact with others), like up-down orientation (which arises because we live in a gravitational field), and many others.

Mark Johnson introduces the theory of image schemas by first examining the question, “How can anything be meaningful to a person?” In a traditional sense, linguistic meaning, which assumes that only words and sentences have meanings and all of these meanings must be propositional, is the primary issue in the philosophy of language and linguistics. Johnson terms this philosophical view *objectivism*. Objectivism treats all meaning as conceptually and propositionally expressible in literal terms that can correspond to objective aspects of reality, a perspective consistent with classical concept theory. Johnson argues this is not enough because, first, meanings in natural language come with figurative and multivalent patterns that cannot be reduced to literal concepts and propositions, and second, the patterns and their connections are embodied and cannot be reduced to literal concepts and propositions (Johnson, 1987).

For example, one of the most important concepts we recognize from infancy is *force*. Thousands of times we move our body, manipulate other objects, and encounter obstacles; also we feel force exerted on us. From these ceaseless events, we develop repeatable patterns that we can recognize and carry out for interacting forcefully with our environment. These patterns are embodied and give coherent, meaningful structure to our physical experience at a pre-conceptual level, though we are eventually taught names for at least some of these patterns, and we can discuss them in the abstract. But the meaning of such a pattern goes deeper than our conceptual and propositional understanding.

Typically, an image schema will have parts and relations. The parts represent a set of entities, and the relations represent the connection among different parts, like causal relations, temporal sequences, part-whole patterns, etc. Johnson and Lakoff give examples in linguistic expressions that

show evidence of dynamic patterns of recurrent bodily experience. For example, Johnson examines six types of “out” in English (Johnson, 1987):

1. Mary got out of the car.
2. Whenever I’m in trouble, she always bails me out.
3. I don’t want to leave any relevant data out of my argument.
4. Don’t you dare back out of our agreement?
5. Honda just put out its 1986 models.
6. We kicked him out of the club.

He argues that all of them relate to the IN-OUT schema shown in Figure 3.1. He identifies the two parts as “landmark” (LM) and “trajectory” (TR). This schema clearly represents the visual situation of the first sentence. Here the circle represents the car, and Mary moves along the arrow out of the car. Since the car can’t be circular, of course, Mary can’t move along a straight line in leaving the car, so this schema gives us only one idealized image. In the rest of the sentences, Johnson explains the pervasive act of metaphorically extending a schema from the physical to the nonphysical.

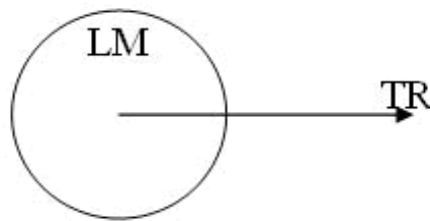


Figure 3.1 OUT Schema

Structure alone is not enough for image schemas to support rational inference. We can perform mental operations on image schemas that are analogs of spatial operations. For example, we can rotate mental image both in 2D and 3D situation, and it seems we can do that at a fixed rate of approximately 60 degree per second, as observed by Shepard and Metzler (Rohrer, 2008). Lakoff also indicates four primary schematic operations, called image-schema transformations (Johnson, 1987),

1. Path-focus to end-point focus: imagine the path of a moving object and then focus attention on the point where it comes to rest or where it will come to rest.
2. Mutiplex to mass: imagine a cluster of objects. Now imagine moving away from the cluster until the individual objects start to appear as a homogenous mass.
3. Trajectory: mentally traverse the path of a continuously moving object.

4. Superimposition: imagine a large sphere and a small cube. Now, increase the size of the cube until the sphere can fit inside it. Now reduce the size of the cube until it fits back inside the sphere.

Although the theory of image schemas is inspired by linguistic and philosophical evidence, along with other research in the neurosciences, more evidence indicates activation patterns like image schemas exist in both animal and human brains (Johnson and Rohrer, 2007). Research from developmental psychology further suggests that infants come into the world with capacities for experiencing image schema structures.

To summarize, image schemas can be characterized more precisely as (Johnson and Rohrer, 2007)

- recurrent patterns of bodily experience,
- that are “image”-like in that they preserve the topological structure of the perceptual experience,
- operating dynamically in and across time,
- realized as activation patterns (or “contours”) in and between topologic neural maps,
- and are structures that link sensorimotor experience to conceptualization and language,
- support “normal” pattern completions that can serve as a basis for inference.

Even though image schemas are generated from physical body experience, they can be applied to many other aspects of our lives, and they can help us to comprehend and make inferences about other experience.

3.2.2 Body schemas

Many researchers believe that humans and other animals maintain an internal representation of their body so that they can perform actions with the physical world without thinking about it. This representation is generally referred to as a “body schema”. Many studies have been done in this area, in both cognitive science and artificial intelligence, but to our knowledge there have been no connections made to explicit concepts.

Head and Holmes first introduced the concept of the body schema in the early 1900s and described it as “(a) the mapping from proprioception and efferent copy (copy of motor command) to body posture and motion, and (b) the mapping from tactile sensation to its originating position on the body surface” (Nabeshima et al., 2005). Other definitions emphasize that the body schema

contains relationships between the body parts and their physical constraints (Acosta-Calderon and Hu, 2005). General, the body schema can be understood as “an unconscious neural map of the spatial relations among the parts of the body, in which multi-modal sensory information (e.g. visual, somatosensory, and tactile) is integrated” (Nabeshima et al., 2005). There also exists a closely related but distinct concept, the “body image”, which is “a consciously manipulable and body-centered version of the body schema” (Nabeshima et al., 2005).

Many fundamental human abilities heavily rely on body schemas. Examples include imitating the actions of other people or even other animals, carrying out body movements, spatial perception, etc. It has also been found that the body schema is not static but can be modified and extended dynamically in very short periods of time. Such extension supports humans in the learning of tool use, as well as more general activities like driving a car. Experiments show that when people drive a car they get the feeling that the boundary of the car is part of their own body.

Research on the concept of the body schema not only helps us in understanding human intelligence, but it has also inspired work on robot system design, which is especially relevant for tool use tasks. Boucard reviews robotics work in using body schemas (Boucard, 2005). One such project is by Verena Hafner and Frederic Kaplan using an AIBO robot. They measure the information distances between all possible pairs of sensors while the robot performs a slow walk and while walking moves its head from side to side. They represent the result in a matrix, called body map. After accumulated enough data, the body map shows that sensors that are informational related are close to each other in a physical sense. In other words, the body map can put highly related sensors into the same category without explicit knowledge about the actual position of these sensors.

In other work, Stoytchev developed an extendable robot body schema based on the Self-Organizing Body-Schema model (Stoytchev, 2003). The main idea is to link the configuration and the sensors of a robot and represent them in one high-dimensional space (called cs-space). The space also represents information about how changes in configuration affect changes in sensor values, and vice versa. Therefore, the system can generate a step-by-step trace from one configuration to another. Stoytchev extended the work by introducing an “offset” vector into the space, so that when applied such a vector, his robot’s actions simulate the effect of using a stick-like tool (see Figure 3.2)

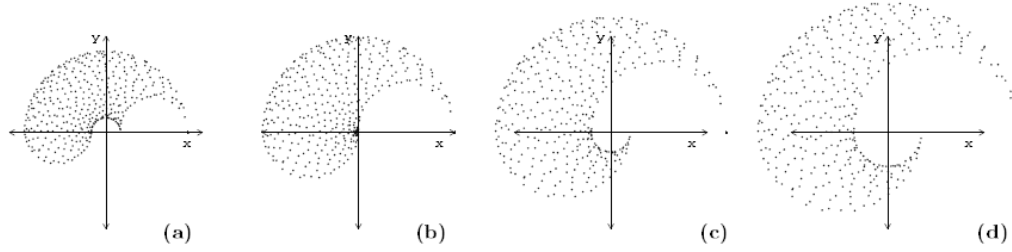


Figure 3.2 A robot arm's original icon

(a) in cs-space, and the extended icons; (b), (c), (d) when different sticks is attached

Nabeshima et al. introduce a model of the body schema that relies on the integration in space and in time of statically updated multisensory information (Nabeshima et al., 2005). They use the combination of two associative memories, a gating neural network (GNN) and a non-monotone neural network (NNN) to record the connection between visual information (a camera on the top of the robot) and tactile information (the joint angles of the robot arms). After accumulating enough experience, the robot executes the following steps in order to reach a target: “(a) generates paths from locations of its hand and a target, (b) internally simulates the paths and selects an executable one using the kinematic controller of its hand, (c) actually moves the hand along the path using the controller, and (d) reaches and touches the target”. They extended the work to the use of a stick by comparing and associating the visual patterns between actions with the stick and the actions without the stick (see Figure 3.3), so that the system can generate plans for using stick with the same steps.

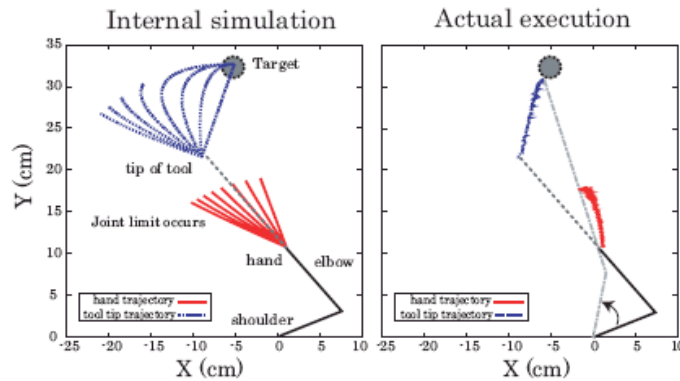


Figure 3.3 Reaching a target with a stick

3.3 Mental imagery

The nature of image schema theory is to provide a mechanism for people connecting the bodily experience with conceptual reasoning. It mainly focuses on where the materials of mental activities come from and how they are interpreted. Research in this area has not yet studied the

mechanism of how the brain actually does the reasoning using those experiences or concepts (Gallese and Lakoff, 2005). Imagine being given the parts of a computer but no instructions for putting the parts together. As a collection of psychological phenomena, mental imagery fills this gap.

Mental imagery can be defined as “the mental invention or recreation of an experience that in at least some respects resembles the experience of actually perceiving an object or an event” (Bertel et al., 2006). Mary Hegarty has studied mechanical reasoning by mental simulation. Hegarty (Hegarty, 2004) claims that a mental model (or situation model) is a representation that is isomorphic to the physical situation that it represents, and the inference processes simulate the physical processes being reasoned about (Figure 3.4). In other words, people will consciously simulate what will happen when they are solving mechanical problems.

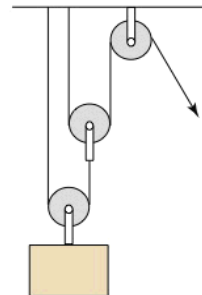
(a) Gear rotation problem



The diagram shows two interlocking gears. Will the knob on the leftmost gear mesh with the groove on the right gear if the gears are rotated inward?

(b) Pulley problem

The diagram depicts a pulley system. When the free end of the rope is pulled, will the lower pulley turn clockwise?



(c) Water pouring problem



The diagram shows two glasses of water. The glasses are the same height and filled to the same water level. If the glasses are tilted, will the water pour out of the two glasses at the same or different angles of tilt? If they are tilted at the same rate, which will pour first?

TRENDS in Cognitive Sciences

Figure 3.4 Examples of mechanical reasoning problems

Hegarty points out three properties in mental simulation:

1. People mentally simulate the behavior of complex mechanical systems piecemeal rather than holistically.

2. Mental simulation is not based purely on visual information, but also incorporates information about invisible entities and properties, such as force and density.
3. Mental simulation may involve motor representations as well as visual representations.

Another important research topic on mental imagery is the nature of mental imagery representations, which has been debated for many years. The center of this debate is whether visual mental images rely on depictive representations or whether they are purely propositional representations (Kosslyn, 1999). This is critical because when researchers began to think about how one could build a computational model to mimic imagery, one must specify a representation with particular properties.

Stephen Kosslyn and Zenon Pylyshyn are two of the most important figures in this area. They have done lot of research in mental imagery and hold opposite views in this debate. Kosslyn designed a series of experiments that focused on the so-called “privileged properties of depictive representations”, which are not shared by propositional representations (Kosslyn, 1999). “Mental Scanning” (Figure 3.5) is one of them. In this experiment, participants learned a map and then were asked to imagine the map, fix their attention on a given landmark, and try to move to a second named landmark in their image. The result shows the greater the distance between the two landmarks, the longer the reaction time a participant needed.

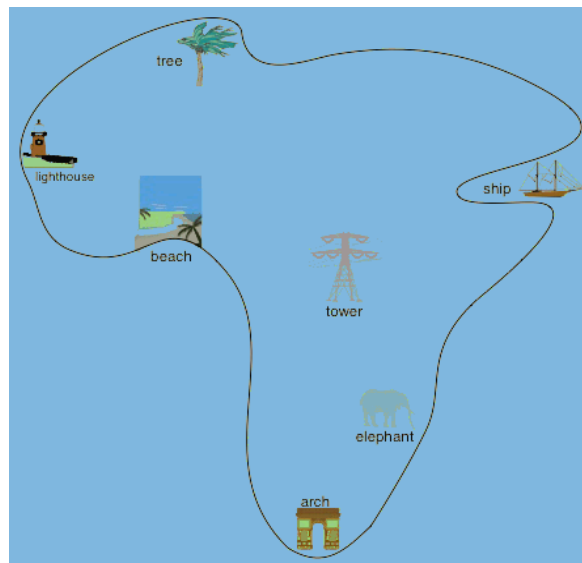


Figure 3.5 Mental scanning experiment

In the latest progress in the debates, many results from neuroscience have been found to support the idea that to have a mental image is to project 2-dimensional moving pictures onto the

surface of your visual cortex (Kosslyn, 1999). This idea has been fostered by the following findings: (1) When a visual pattern is presented to the eye, a homeomorphic (continuously deformed) mapping of retinal activity occurs in visual cortex. (2) Although it remains controversial, it has also been reported that there is increased activity in retinotopically-organized areas of visual cortex during mental imagery.

Pylyshyn (Pylyshyn, 2002) (Pylyshyn, 2003) argued that these findings tell us nothing about the form of the representation, because imagery and vision might involve the very same form of representation without it being pictorial in either case. Furthermore, Pylyshyn was concerned that there may be an infinite regress in pictorial mental imagery: if something is interpreting the image, what inside that something must be interpreting its interpretation of the image, and so on?

Although the debate about the nature of mental imagery is still not entirely resolved yet, it doesn't affect the idea that pictorial depiction is involved in mental imagery. The important thing about this debate is that it helps us to recognize what kinds of properties are involved in depictive representations but not in propositional representations. By analogy, when you program an algorithm on computer, you can ignore the nature of a computer as a process of interactions between electrons. Instead, you focus on a higher and more abstract level. That is, we can ignore the underlying disagreement about whether a depictive representation is based on propositional representations or is primitive; the evidence is compelling that depictive representations play an important role in mental imagery.

3.4 Summary

From this body of literature we take the following guidelines for the research in this proposal.

1. Concepts need not have explicit (symbolic, propositional) definitions.
2. What a concept represents (its content) may depend on the embodiment and experience of the agent that holds that concept.
3. Some limited set of common schematic relationships may be applicable in a general way over the experiences of an agent.
4. In the application of concepts to new situations, especially in situations that require deciding what to do, simulation of the concept in its application to the situation is a plausible part of understanding the situation.

We will see how these guidelines play out in the following sections.

Chapter 4

Cognitive Robotic Architectures

Designing a complete cognitive robotics system means solving problems in many disciplines. Even though this research mainly focuses on an intermediate layer of a robotic system, one that links data to symbols, it requires efforts in the areas of time series analysis, schematic representations, and data mining and machine learning. System architecture design becomes an inevitable and critical issue.

4.1 Examples of cognitive robotic architectures

There are many different frameworks that have been used in robotic systems. Most approaches up until the 1980s made use of a sense-model-plan-act framework. As criticized by Brooks, those systems use representational schemes with fixed and predetermined interpretations (Brooks, 1999). Brooks developed the idea of *behavior-based robotics*, which emphasizes the integration of semi-independent layers that produce behaviors directly from input rather than each contributing to a stage of the sense-model-plan-act framework (Lewis and Luger, 2000).

Brooks's work opposes the conventional representation that explicitly pre-stores the description of the interactions with the world and the intentions of the system. Instead, he claims that all autonomous agents need to be situated and embodied, and he also argues that embodied robots should "use the world as (their) own best representation" (Brooks, 1999). In order to test his ideas, Brooks has built a series of robots based on a decomposition-by-activity subsumption architecture. The robots can wander around in the real MIT lab environment, avoiding static

obstacles, like walls, and moving people, based on low-level considerations. At a higher level, the robots might include goals such as the desire to approach given targets. With the robots and the subsumption architecture, Brooks demonstrated his claim that a robot can continually create and recreate relationships between the system and the world without relying on a central planning facility to dictate commands, or encoding classical representations.

This research direction inspired a variety of new dynamical models of representation. One example is the architecture designed by Joseph Lewis for robot control, an architecture that is both dynamical and embodied.

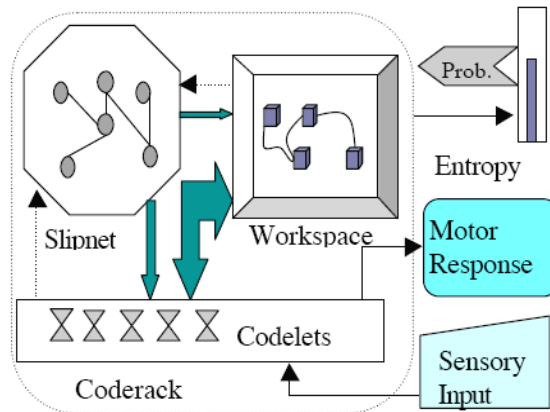


Figure 4.1 Madcat architecture

Lewis’s project, “Madcat”, is inspired by a similar design in a previous project called “Copycat” (Mitchell, 1993). Copycat was one of the first computer programs to attempt to capture the dynamical processes from which symbolic or representation-based behavior can emerge. Copycat solves analogy problems such as If “*abc*” becomes “*abd*” what does “*ijk*” become? One aspect of its novel design is the “slipnet”, “a semantic network organized with spreading activation and multiple kinds of links among its nodes, some of which can change in length”. In other words, the slipnet’s internal topology will keep updating while the system is running. Madcat extends the Copycat architecture to a robotic control system, producing a control architecture capable of ongoing interaction with a dynamic environment. Figure 4.1 shows Madcat’s components in the architecture and the relationships between them. “Codelets” encode four basic rules with high priorities for a given set of sensor readings. The “coderack” is a stochastic priority queue where the choice of the next codelet is made probabilistically with a bias toward the higher urgency codelets. The workspace serves as the locus of structure-building activity of the codelets from the coderack. Activity in the workspace biases codelet choices in the coderack. Similar to Copycat, the slipnet

contains nodes and links that dictate the data to which the codelets respond and the kinds of structures they build. The slipnet topology changes in response to activity in the workspace but its nodes and links remain fixed. The entropy reflects how well emergent structures fit into the data the robot encounters and affects the biases of the system.

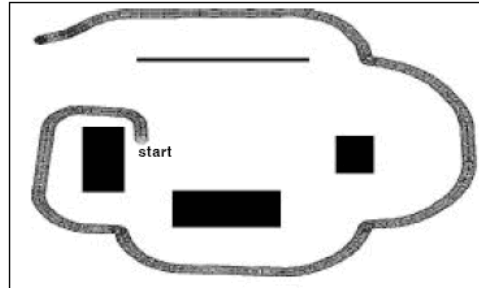


Figure 4.2 Demonstration of Madcat

With this design, the Madcat architecture first demonstrates that certain basic competencies, roughly those of Brooks (Brooks, 1999), could be implemented using this emergent architecture. The chosen behaviors are obstacle avoidance, wandering, and wall-following (Figure 4.2). The second goal of the Madcat architecture is to generate emergent structures correlated with environmental features. These support more effective real-time behavior.

Another interesting architecture design is the machine consciousness approach of Arrabales Moreno and Sanchis de Miguel (Moreno and de Miguel, 2006). They have analyzed the major theories of consciousness and aimed for the design of a machine consciousness model, which they call CERA (Conscious and Emotional Reasoning Architecture). Human consciousness functions, as identified by Baars, include adaptation, learning, contextualization, access to a self system, prioritization, and recruitment of unconscious processors, decision making, error-detection, self-monitoring, and optimization. Accordingly, Moreno and de Miguel have defined a set of basic modules intended to accomplish all of the functionality of reasoning consciousness. The defined modules are as follows:

1. Attention module.
2. Status assessment module.
3. Global search module.
4. Preconscious management module.
5. Contextualization module.
6. Sensory prediction module.

7. Memory management module.
8. Self-coordination module.

Moreno and de Miguel discuss the functionality of every module. For example, “the implementation of the attention module implies a mechanism that allows the robot to pay attention to a particular object or event. This focus point should drive its perception, behavior and explicit learning processes.” Figure 4.3 shows how different modules interact with each other. Also, they consider “emotions” as a measure that provides an agent with a succinct evaluation of its progression in achieving goals.

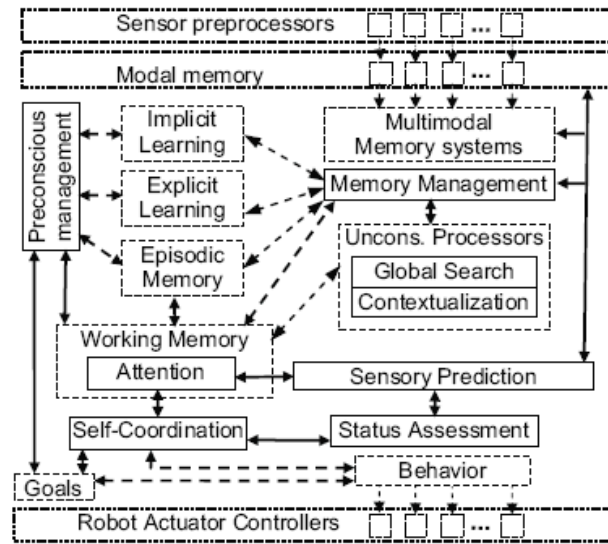


Figure 4.3 Reasoning consciousness model

This architecture allows the integration of different cognitive components into a single autonomous system. It is designed to be a flexible research framework in which different consciousness and emotion models can be integrated and tested. They test the system on the Khepera robot for navigation tasks.

4.2 Design Challenges for a Cognitive Robotic Architecture

In a typical design of a robotic system, a perception module is usually present at the front of the system, responsible for collecting and processing raw sensorimotor information, usually in numerical formats. There might be a central control module, which usually deliberates over the behavior of lower level modules and controls the general behaviors of the entire system through a decision-making process based on the information it has. Symbolic representations, like logic or

schemas, are typically the best fit for this module. However, it's often unclear how the information should best be transformed from numerical format in the perception module to the symbolic format in the high-level reasoning module.

As described earlier, this dissertation focuses on an intermediate layer of a robotic system, more specifically of a robotic system targeted at tool-using tasks. The general goal is to understand how sensory and motor experiences in an embodied physical agent can give rise to semantic primitives and support effective tool-using behavior, and to build an agent that represents such understanding. This intermediate layer focuses neither on low-level perceptual processing, such as computer vision, nor on high-level processing, such as logical reasoning or planning. Instead, it acts as a bridge, one that translates raw sensorimotor information into a structured representation that might be applied directly or used for higher-level reasoning. This component is an *Action Schema Generator*, or ASG.

In order to bridge this gap between raw sensorimotor data and a symbolic representation, the ASG needs to solve three major challenges. First, it needs to break down the continuous sensorimotor data into a discrete format, and make distinctions about when the inputs are quantitatively different but qualitatively similar. Second, it needs to identify frequently appearing pieces of information and organize them appropriately in order to make learning or other high-level reasoning possible. As we have stated earlier, theories of embodied cognition argue that meanings need more than just a definition. Third, then, the ASG, needs to create connections between symbols, especially for symbols that represent actions.

It is also important to limit the expectations we have for an ASG that addresses these challenges in a robotic system. In our design, the ASG doesn't include a long-term symbolic memory module, nor does it include a symbolic memory management module. These constraints on the design can be motivated by practical reasons: a larger range of scenarios or environments can be accommodated if the ASG does not need to maintain potentially domain-specific representations of its input over time. Further, the structure of memory is sometimes dependent on assumptions made at a higher cognitive level; by taking this capability out of the ASG, it gains some generality. These constraints also have implications for how the ASG acquires and maintains its knowledge structures: it will be trained on a variety of scenarios, such as the one described in Chapter 2, but it will need a

long-term memory module to accumulate knowledge between such scenarios². Rather than relying on an “ideally” self-motivated continuous learning method, the ASG will depend on a “localized” learning method.

4.2.1 System Inputs and Outputs

It is also necessary to discuss the input data the ASG must deal with. For a human being, most useful information comes from visual processing. This is also true for the systems we are concerned with. However, computer vision is a complicated problem and beyond the scope of this research. This research doesn’t directly work on visual signals. Instead, we assume that necessary information in the visual signal has been separated out and translated into independent signal streams. For example, this information could include an object’s relative position and size, a tool’s position, size, and orientation, the agent’s location, and even information about some parts of the agent’s body, like its forearm or the endpoint of its paw.

The format of the input signal we use is a conventional time series, a sequence of numerical data points, measured at successive times, spaced at (often uniform) time intervals. The reason for using time-series data is that it can carry the widest variety of signals in a robotics system. Thus the analysis applied to this data won’t lose generality due, for example, to assumptions about a static environment. The data recorded from simulations and from the robot platform for input to the ASG is sampled every 0.1 seconds. However, there is no limitation or requirement on this rate in either the system design or implementation.

In order to assist the system in processing multiple trials and accumulating information as background experience, we rely on script files to describe scenario settings. These script files mainly contain configuration information for scenario environments and information about the agent’s sensors and controllers. A sample script file is included in an appendix.

The outputs of the ASG are aimed at providing a symbolic representation of sensorimotor experiences as well as embodied knowledge about the actions of a robot agent. The outputs should be able to support a high-level system to make reasoning or planning about the robot’s current status, possible actions, and so forth.

² We demonstrate how to combine knowledge from different scenarios in a system evaluation program in a later chapter.

There are three types of output data that the ASG generates. (All of the information that follows will be described in more detail in a later section; the goal here is to provide an overview, for context.) First is the definition of a data structure we call a *prototype definition*, which connects sensorimotor data with symbolic primitives. This output data not only provides the interpretation for the symbols used by a higher-level system, it also supports the capability of translating symbols back to sensorimotor data. Therefore, a high-level system can recover original sensorimotor data from prototype symbols during the reasoning or planning processes, such as a self-monitoring mechanism, like the “emulator” in Clark’s theory of cognitive robotics (Clark and Grush, 1999). The second type of output is the *prototype description*, which uses prototypes to interpret the input to the ASG in real time. This information is intended to support a real-time reasoning system, and it also provides full coverage of the sensorimotor experiences of the system. The last type of output, which we call *action schemas*, is an important supplement to the first two types of outputs. Action schemas are rule-like specifications of how the system interacts with its environment.

4.2.2 System Components

Based on the system setup and the research goals, as described above, we partition the whole system into three successive stages. Every stage will have a specific task description and goals. Their work flow is briefly described in the following.

Time Series Segmentation

The system starts by taking signals from the sensors and controllers of a robot agent in time series format. The data should contain enough information to describe the current status of the robot and the environment around it. Therefore, the first stage, time series segmentation, deals with time series analysis. The main goal in this stage is to translate every sensorimotor signal stream from time series format into multidimensional vectors to which metrics and further processing can be applied. This translation is needed because although time series data is easy to collect, it’s hard to analyze and compare directly, especially when we are looking for patterns that may be quantitatively different but qualitatively similar.

As a result of time series segmentation, the sensorimotor data will be transformed into a set of vector descriptions. The most important difference between these two types of data is that the latter is model-based. That is, based on different methods we use to translate the data, the vectors will capture different knowledge.

Prototype Construction

In the second stage, prototype construction, vector descriptions of data will be sealed into schema-like structures. To construct a schematic representation, the most important task is clustering the data; prototypes will be used to represent clusters, or groups. The variance between the different data instances within a group will be described by the parameters associated with the prototype. For example, an “increasing” prototype for a time series could capture patterns that show an increase in values but at different rates or starting points.

After performing cluster analysis, the system can classify new sensorimotor information with the prototypes it has generated. This is the key step during the translation of a robot’s raw data to a symbolic representation.

So far, the analysis as described is only applied to individual sensorimotor signals (though with dependence on time in time series data). By applying prototype construction, the system will generate both prototype definitions and descriptions from embodied experience. The information contained in these two outputs provides a real-time symbolic description of the robot agent and its environment. This description can be used independent of the next stage of processing. However, translating sensorimotor experience into symbols alone won’t capture all the embodied knowledge for higher-level reasoning and planning. The important work in the last stage is called action schema discovery.

Action schema discovery

In this stage, the analysis focuses on the embodied connections between prototypes; connections are embodied in the sense that relationships between the patterns seen in different sensors arise because they are part of a single agent. That is, action schema discovery is a process that tries to find frequent connections between multiple schemas, more specifically, for action schemas. If connections along with particular control signals repeatedly recur, these will be the relationships that the system needs to capture. In its analysis, the ASG is designed to recover the preconditions and the effects of an action. Having this information should benefit a higher-level system in reasoning and planning.

A diagram of the system architecture looks like Figure 4.4, where parallelograms represent the data, and rectangles represent the process.

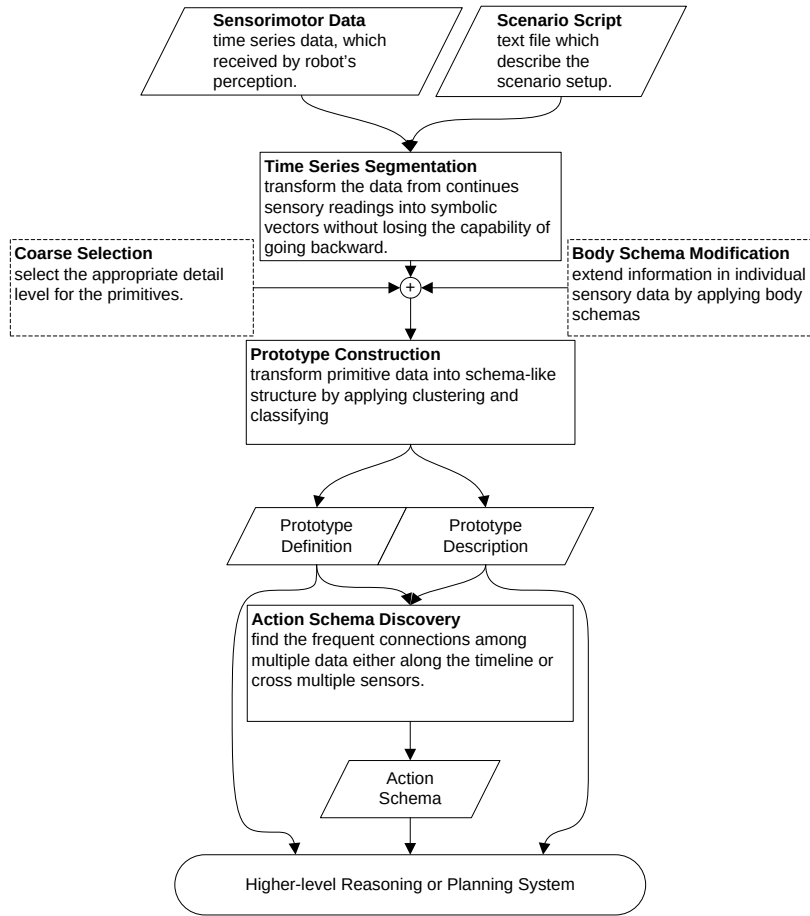


Figure 4.4 System architecture

Chapter 5

System Description

In this chapter, we explain the components of the Action Schema Generator, step by step. We answer questions related to the problems they face, the design and implementation of representations and algorithms to solve these problems, and expectations for the results and performance of these algorithms. The structure of this chapter reflects the breakdown of components in a cognitive robotic architecture given in the previous chapter.

5.1 Time Series Segmentation from Sensory Data

In the first stage of the system, the goal is to transform the agent’s sensory information from continuous readings into measurable primitives without losing the capability of being able to reproduce that sensory information, i.e. to transform the primitives back to “typical” (i.e. prototypical) sensory data. For context, we begin with a brief discussion of related work.

One of the simplest approaches to time series segmentation involves representing n -bits of time-series data by an n -dimension vector, using Euclidean distance to measure the similarity. But clearly this requires unrealistic amounts of computation and storage, and it provides no insight about the data. A more sophisticated and well-known approach in the data mining community is Dynamic Time Warping (Sankoff and Kruskal, 2001). In DTW, sensory data basically keeps its “continuous” format, but the time axis can be moved like an elastic string so that two time series can be compared in a more “essential” sense. However, DTW needs to pre-segment the data into slices before actually applying the comparison. Therefore, in general, the system needs a method which can both

segment the continual data into discrete and use as simple as possible vectors to describe as much as possible.

Minnen's recent research involves discretizing a sequence by symbolic aggregate approximation (SAX) method, and then building a collision matrix to detect the most often repeated sequences (Minnen et al., 2007). This is a very common approach in which the basic primitive is defined by a fixed radius and the frequency of occurrence is the measure for usefulness.

In Rosenstein and Cohen's work, basic primitives are pre-coded patterns (Rosenstein and Cohen, 1999). In an unsupervised fashion, these patterns are used to detect events, and then Euclidean distance is used to compare sequences. As a result, an alphabet is generated that describes sequences. In Oates, Schmill, and Cohen's later work, they use DTW to measure the similarity and clustering algorithm to categorize those robot experiences and build prototypes (Oates et al., 2000). However, one limitation of this approach is that it requires external supervision to divide a time series into shorter series that contain instances of the structures they want to find. The technique cannot accept time series of numerous undifferentiated activities (e.g., produced by a robot roaming the lab for an hour).

Research in this area generally relies on basic primitives that can be defined in two ways: first, a fixed radius sequence, as in Minnen's work and DTW; second, pre-coded patterns or a sequence which is constructed by pre-coded patterns, as in Cohen's work. Either way, from the functional perspective, a primitive should be used as an atom in the system, which means although there might be some parameters associated with the primitives its internal structure is a black box from the perspective of other parts of the system.

5.1.1 Piecewise Linear Representation

For generality, we want to avoid pre-defined patterns and pre-segmentation of continuous signals in this stage. A commonly used technique, called piecewise linear representation, can provide help for both modeling and segmentation.

In theory, a piecewise linear representation refers to an approximation of a time series T , of length n , with K linear regression vectors. Typically, since K is much smaller than n , this representation makes the storage, transmission and computation of the data more efficient (Keogh et al., 2003). In our research, more than this, linear regression analysis will capture relevant information about a robot's activities.

We refer to a bounded set of contiguous data points in a univariate time series as a *segment*. For each segment in a time series, linear regression converts the numerical data into an equation with a limited number of parameters. The parameters can be treated as a vector to describe the original data set. The value of the regression result will depend on how well the equation fits the data.

$Y = X\beta + \varepsilon$ is the general equation for a linear model, where Y is a column vector that includes the observed outcomes, in this case the signal readings; X is a matrix that includes the observed inputs, in this case the timestamps. For each segment, a linear regression only needs a value of β to determine the approximation. Therefore, the only task left for piecewise linear representation to approximate a time series is to determine where to segment the data.

In the equation above, ε is a general expression of error, typically computed by the sum of squares of the residuals $SS[E] = \sum (y_i - \hat{y}_i)^2$, where y_i is the observed value, and $\hat{y}_i$ is the approximation value. The value of $SS[E]$ is actually a measure of the deviations between original data and the approximation. There are three main causes that can lead to deviations:

1. System noise, caused by simulator sensor noise or external disturbances. It's inevitable to have system noise, and regression is generally stable with noisy data in our simulation scenarios.
2. Un-captured properties of the original data. In other words, a data set may not be correctly described by the regression equation, e.g., if it is non-linear. Since most of the simulation scenarios and trials with the physical robot platform deal with simple physical interactions, the assumption is that linear regression provides enough representational power.
3. Status change. In other words, a robotic system may not always remain in the same state. When a status change happens, the previous equation parameters will not fit the current period. In a piecewise linear representation, this means another segment is needed to represent the new status.

It's not hard to see that identifying status change is the key point for the time series analysis. This information governs how to discretize the raw data for further computation. Therefore, it's very important to find a way to distinguish a status change from system noise.

A common technique in time series analysis is to use “white noise” to model system noise for a physical environment. White noise is a set of uncorrelated random noise with zero mean and a certain variance. An example may look like Figure 5.1.

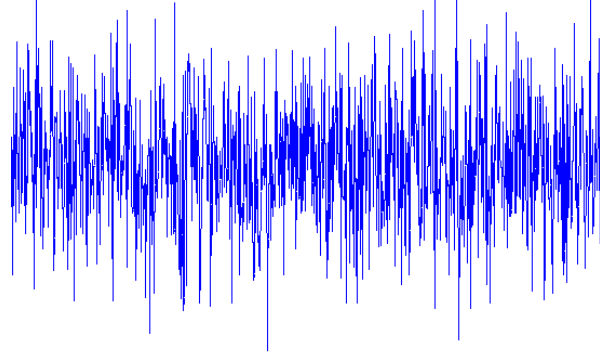


Figure 5.1 Example of white noise

White noise obeys a normal distribution. This is called Gaussian white noise. That is, the noise deviation satisfies the probability equation of the normal distribution - $norm(0, \sigma^2)$.

As discussed above, system noise and status changes are the only two causes we consider relevant for the deviation of the approximation. Therefore, the probability of such a deviation between observed data and approximated data at any time point can be calculated by a statistical hypothesis test — a t -test, in which the hypothesis is “the current deviation is belong to a normal distribution.” In other words, if the probability of such a deviation is larger than the t -test’s threshold, segmentation should be done.

In addition to using the deviation result for the hypothesis test, there are confidence intervals associated with every segment, which indicate possible differences between the real parameters and the approximated ones. The confidence intervals can be computed by

$$[\hat{\beta}_j - \sigma_j t_{m-n; 1-\frac{\alpha}{2}}; \hat{\beta}_j + \sigma_j t_{m-n; 1-\frac{\alpha}{2}}]$$

where $\hat{\beta}_j$ is the estimated parameter, σ_j is the standard deviation, and $t_{m-n; 1-\frac{\alpha}{2}}$ is the t -function based on the difference between the size of data and the degree of the regression, and the possibility threshold. A t -function value table can be found form Table 5.1.

Table 5.1 t-function values

m-n	Probability			
	0.1	0.05	0.01	0.001
1	6.31	12.71	63.66	636.62
2	2.92	4.3	9.93	31.6
3	2.35	3.18	5.84	12.92
4	2.13	2.78	4.6	8.61
5	2.02	2.57	4.03	6.87
6	1.94	2.45	3.71	5.96
7	1.89	2.37	3.5	5.41
8	1.86	2.31	3.36	5.04
9	1.83	2.26	3.25	4.78
10	1.81	2.23	3.17	4.59
infinity	1.65	1.96	2.58	3.29

5.1.2 Algorithm Implementation

In the implementation of the piecewise linear representation, a linear equation $y = \beta_0 + \beta_1 x + \varepsilon$ is used to produce a least squares approximation for each segment of the data. Using $y = \beta_0 + \beta_1 x + \varepsilon$ has two clear advantages. First, it's among the simplest and most intuitive methods appropriate for the simulation data; second, it captures the most fundamental physical aspects of a variable over time, its value and rate of change.

To find the solution of a simple linear regression, the values of $\sum x_i, \sum y_i, \sum x_i y_i, \sum x_i^2, \sum y_i^2$ are computed. In order to avoid low efficiency, a vector class is built to dynamically keep these values updated. Thus, the time complexity of maintaining the data and computing regression result is linear in the size of the data set. Two equation parameters β_0, β_1 and the sum of squares of the residuals $SS[E] = \sum (y_i - \hat{y}_i)^2$ are generated. These parameters can be used to recover the data at any time point. The value of $SS[E]$ is used to compute the possibility that a status change is happened.

Table 5.2 illustrates the algorithm for find the piecewise linear representation based on the linear regression and the segmenting hypothesis t -test. The algorithm basically starts from the beginning of a time series, and scans for every data point. If a segment test result is larger than the threshold, it will segment the data; otherwise it just adds the data point into the current segment until all the data is processed.

Table 5.2 Algorithm of regression analysis

```
Algorithm Seg_TS = getPrimRegression(TimeSeries ts)

time = 1;
while (ts is not finished)
  if no current segment
    create a new segment using linear regression;
  elseif SegmentTest(ts, time) < threshold
    time = time + 1;
    add ts(time) into the current segment using linear regression;
  else split = findBestSegmentTime(time);
    put current segment into Seg_TS;
    time = time + split;
end;
```

Since the algorithm processes data one value at a time, it can be used as an online algorithm, which allows it to be used in a live robotic system. In the implementation of the segment test (Table 5.3), a set of points rather than just one is chosen, to generate a more reliable result. If the probability exceeds a pre-set threshold, then the system will decide to change the status and the segmenting function will be executed to compute the segmenting point. During the computation the threshold is also varied according to the size of the data based on the confidence interval.

Table 5.3 Algorithm for the segment test

```
Algorithm p = SegmentTest(TimeSeries ts, time)

p = 0;
for i = a set of points starting from time
  dat = the linear approximation of ts at time i;
  diff = the deviation between ts and dat;
  p = p + the probability of the deviation as white noise base on the current segment's
  confident interval;
end;

return the average p
```

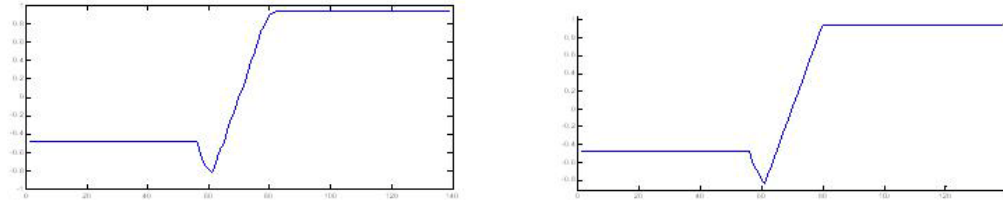
The result of the segmenting function is used to determine whether to separate the data set before and after a time-stamp. If a separation happens, the “after” data set will be used to compute new regression parameters. One more precautionary function implemented is to test the segmentation point in a local neighborhood to find the best result (minimal $SS[E]$ for both “before” and “after” regression).

During the analysis, there are several pre-set parameters: the standard deviation of each signal, the probability threshold, and the minimal length of data for regression. Those constants are all loaded from the scenario script file.

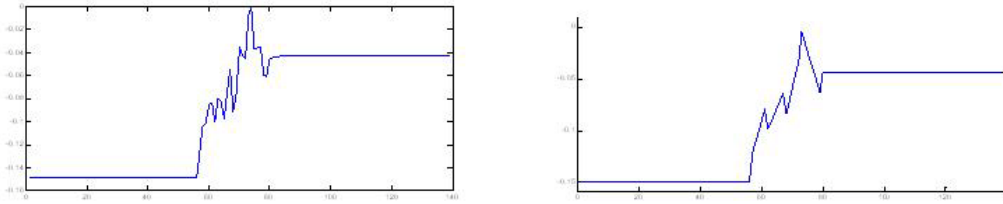
The algorithm has been used on variant scenarios generated from both Breve simulation and Sony Aibo (Figure 2.2), which will generate real physical interaction data with a lot of noise. Figure 5.2 is an example of the analysis, in which Aibo is approaching a ball at the front in our simulation environment.

There are three measures used to monitor the ball from the camera video, placed in the head of Aibo. These are the ball's vertical position in the video, ball-X; the horizontal position in the video, ball-Y; and the size of the ball appeared in the video, ball-Area. On the left side of the following figure, there are the actual sensor readings during the Aibo's approach to the ball. On the right side, there is the approximation by piecewise linear representation.

Ball X after noise filter



Ball Y after noise filter



Ball Area after noise filter

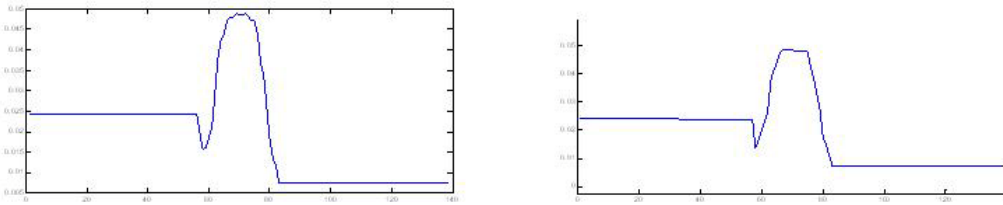


Figure 5.2 Primitive analysis on Aibo approaching a ball

Figures in the left are the original data, right are the regression results.

The next two figures illustrate the similarity between the complete data from Aibo's sensors and approximation results (Figure 5.3 and Figure 5.4).

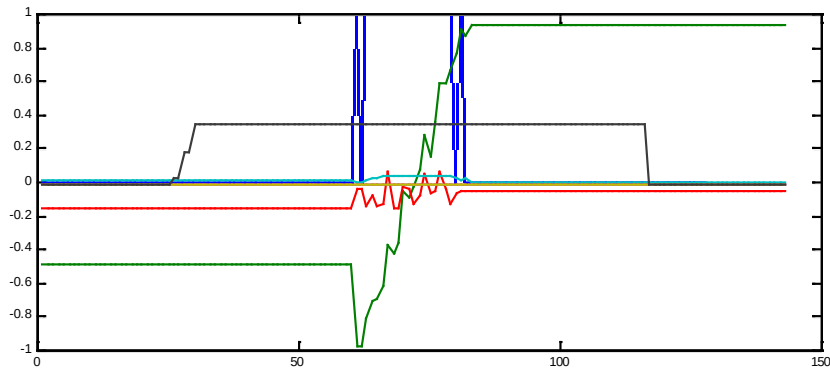


Figure 5.3 Original sensor data of Aibo approaching a ball

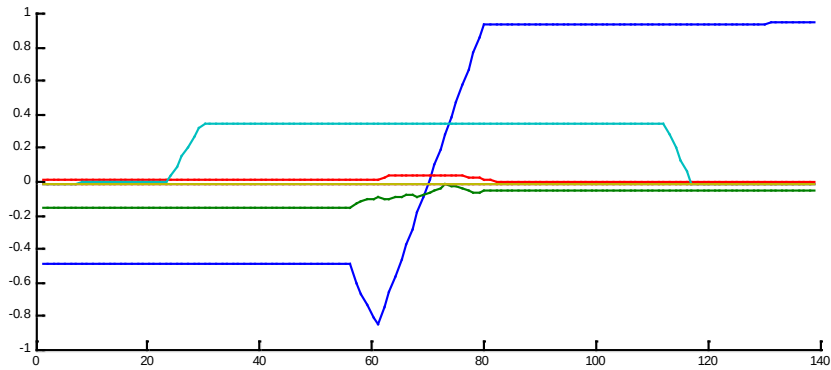


Figure 5.4 Regression result of Aibo approaching a ball

There are several advantages to this approach relative to other approaches such as Minnen’s or Cohen’s work. First, for each segment, there’s no longer a direct association with time. In the other approaches, a predefined time radius is necessary. Comparing different length segments may require re-interpolation (DTW) or it may be impossible. In this approach, the linear model only describes the status of the current data; duration could be a parameter associated with the representation, but it’s not the only possible parameter for detecting a segment. (There’s a minimal time buffer requirement for each segment; however, this is irrelevant to the detection result).

The second advantage is that this approach might provide more insight about the physical environment, like “speed” and “acceleration”. Linear models are essential for preliminary physical interactions. Many physical laws like Newton’s three laws are easily fit with linear models, when well-understood transformations are applied. Most sensory and controller information in the robot tool use tasks we are working with also can be described well by linear models. Therefore, when comparing the sequences, we can expect the algorithm to discover essential relationships. This

approach assumes that having a basic model of the time-series will help a cognitive agent learning about its environment. This assumption is reasonable, according to Rosenstein and Cohen (Rosenstein and Cohen, 1999) who write, “Mandler postulated that to build such a foundation (sensory category), humans make use of an innate mechanism for sensory analysis that searches for regularities and preserves the continuous nature of perception.”

5.1.3 Coarse Selection

The idea of doing coarse selection comes from a very common phenomenon in human thinking, which is ignoring detail and focusing on general trends. Coarse selection addresses an issue in piecewise linear modeling, that of determining the appropriate number of segments for a time series T . For instance, for a part of the observation of Aibo sensory data, we might have a regression result show in Figure 5.5. From time 56 onward, there are three increasing segments. Must it be three? In some cases, we might only want to know it is increasing.

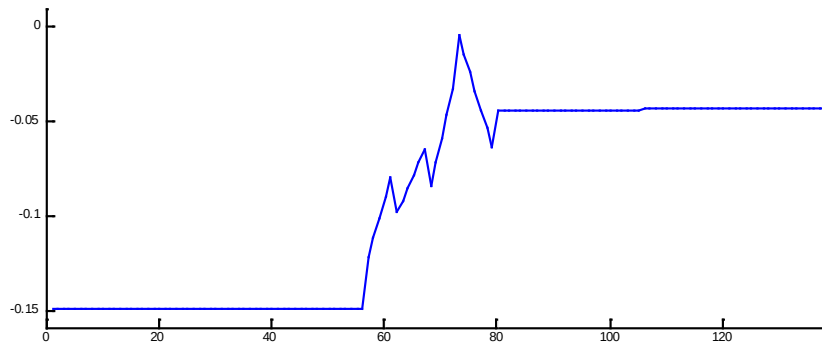


Figure 5.5 Ball’s position in original data

The basic principle of avoiding too much detail is simple: merge the most similar segments into one single segment. The approach may look like the opposite of data segmenting, but it is different in an important way: it is based on global information, while data segmenting is based on local information.

Table 5.4 explains the coarse selection algorithm. When merging the segments, the algorithm is not simply erasing detail, but rather using a hierarchical data structure to keep track of every step, i.e. at the top of this structure are all the segments without clustering information; every time the algorithm moves to the next level, one more segment is merged. The goal of this structure is that for any given tolerance level, this structure can recover a data segment set that ignores all the details

below that level. Figure 5.6 shows the figures for the original regression data and some level of clustering of the results.

Table 5.4 Algorithm of coarse selection

```

Algorithm SegComb = CoarseSelect(SegList)

SegLevel = new Segment List;
for all segments in SegList
    add segment into SegLevel;
end;
add SegLevel into SegComb;

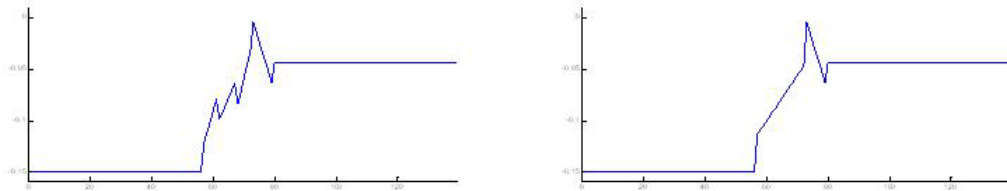
while there's more than one segment in SegLevel
    NewSegLevel = new Segment List;

    copy all segments from SegLevel to NewSegLevel;
    find two adjacent segments with the lowest combination error in NewSegLevel;
    combine these two segments into a new segment;
    remove these two segments in NewSegLevel;
    add combined segment into NewSegLevel;

    add NewSegLevel into SegComb;
    SegLevel = NewSegLevel;
end;

```

Ball Y Regression result



Ball Area Regression result

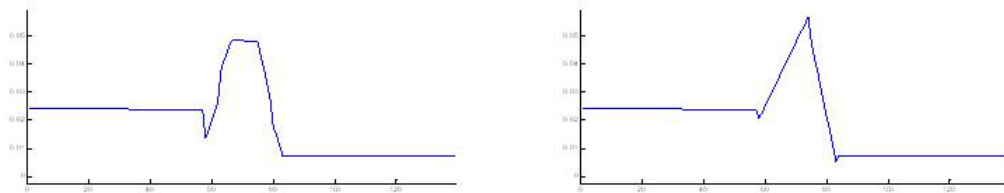


Figure 5.6 Reducing the details in the regression result

5.2 Prototype Construction

After the sensor data has been segmented and translated into a piecewise linear representation, we need to construct symbolic descriptions. By doing so, deeper analysis can be applied and more qualitative relations can be revealed. Also this stage provides the foundation for higher-level reasoning, especially planning.

5.2.1 Concept Categorization

A piecewise linear representation provides an abstraction of sensorimotor signals. Those signals record the activities of the robot system. Learning from this information requires first finding the useful vectors and then categorizing them by their embodied meanings. Categorization is the central and very first task for constructing prototypes.

Categorization needs a measure of utility. There are many ways of evaluating a vector, e.g. the frequency of its appearance, whether it has unexpectedly appeared, its duration, etc. In this research, the most interesting measure is how often the vector appears. In other words, typical vectors will have the focus and priority to be selected. Given this approach, if the most popular vectors are known beforehand, this would be a classification task. However, this research aims to maintain as much flexibility for learning as possible, and thus we treat the task as a clustering task.

Cluster analysis addresses this problem: “Given a collection of n objects individuals, animals, plants etc., each of which is described by a set of p characteristics or variables, derive a useful division into a number of classes. Both the number of classes and the properties of the classes are to be determined” (Everitt, 1995). The most significant difference between classification and clustering is whether the classes are predetermined.

Clustering algorithms have many variations, like k-means and hierarchical clustering. However, most algorithms related to k-means require that the number of clusters be pre-determined. And even though hierarchical clustering doesn’t explicitly require a known number of clusters, it still needs a judgment based on a similar requirement, to decide on which layer to cut. These methods aren’t appropriate for this research, because there’s no way for a designer to know how many kinds of concepts a robot should find in a given scenario.

For this challenge, density based clustering algorithm becomes the best choice for solving the problem. Instead of setting out how many clusters are in a dataset, density-based clustering sets a criterion for a density measure, which means that a cluster will only be formed when there’s enough

data that belongs to it. In other words, the density measure assures that a concept will only be formed when there are enough data samples in close proximity. A similar situation applies in the formation of human concepts, according to prototype theory.

5.2.2 CLIQUE algorithm

Among density-based clustering techniques, CLIQUE (CLustering In QUEst) is a grid-based clustering algorithm that provides an efficient approach specific to multidimensional space (Tan et al., 2006). Grid-based clustering refers to how the algorithm organizes data in a multidimensional space. The idea is to split the possible values of each attribute into a number of contiguous intervals, creating a set of grid cells. Each object falls into a grid cell whose corresponding attribute intervals contain the values of the object. This technique is very useful in handling a continuous data space, as generated by sensor signals.

CLIQUE is also aimed at multidimensional data, because the algorithm not only finds dense areas in a high-dimensional data space, it also reveals dense areas in subsets of the space. The algorithm starts by finding a dense area in each individual dimension of the data space. It splits the dimension into a fixed number of equal volumes, and computes the density of each unit. A unit is considered dense if the fraction of the overall points that it contains is above a user-specified threshold. A cluster is simply a group of collections of contiguous dense units. After finishing the computation for lower dimensions, the algorithm moves to higher dimensions, which are combination of lower dimensions. Instead of checking the entire space, CLIQUE assumes the monotonicity property for density-based clusters: “If a set of points forms a density based cluster in k dimensions (attributes), then the same set of points is also part of a density based cluster in all possible subsets of those dimensions (Tan et al., 2006).”

Table 5.5 explains the procedure of the CLIQUE algorithm that is implemented in our system. Notice that the primitives generated from piecewise linear representation analysis have two parameters and therefore form a two-dimensional data space for CLIQUE.

Table 5.5 Algorithm of CLIQUE

<pre>Algorithm clusters = Clustering(space, min_dense) for all cell in the low dimensional of space if cell is denser than min_dense add cell into cell_list; for all the cell in cell_list Generate the two-dimensional cell2D from the current cell; if cell2D is denser than the min_dense record cell2D into cell2D_list; end find clusters from the union of all adjacent cell2D in cell2D_list;</pre>
--

5.2.3 Modified CLIQUE Algorithm

Even though CLIQUE meets our basic requirements, the algorithm depends on too many preset parameters for clustering for our purposes. We would like a robotic system to make decisions by itself, as much as possible, rather than relying on pre-defined thresholds. There are three parameter-related requirements to be dealt with. First, we want an algorithm that is less affected by the process of partitioning the density space into grids. This becomes an important problem when the size of the data samples is not large compared to the size of the space partition. Second, we want an algorithm to have a consistent density measure no matter how many data are applied and how many grids are generated by the partition. Third, the result of CLIQUE is also largely decided by the density threshold, which is used to screen out the high-density areas; we want an algorithm that is less reliant on such a threshold, because it would be difficult for a designer to foresee the appropriate value for an autonomous robot agent. Based on those requirements, we modified CLIQUE algorithm in the following respects.

The first improvement is to apply the influence of data as a distribution instead of a single point in the density space. This technique is based on so-called kernel density estimation in the area of statistics and pattern recognition (Tan et al., 2006). The central idea is that the overall density function can be expressed as the sum of a kernel function that captures the neighborhood around individual data points. The commonly used kernel function is the Gaussian function,

$$K(y) = e^{\frac{-\text{dist}(x,y)^2}{2\sigma^2}}$$

which is extended in two-dimensional space.

The greatest benefit of introducing a kernel density estimation technique is when there's not enough data for the clustering algorithm to return reliable results. Since the density space is partitioned into grids in the original CLIQUE algorithm, a data point just on one side of a grid line will have no influence on the other side of the line. If there are a lot of data points, this problem won't be an issue because the distribution of data points will be captured by the entire data set. In the scenarios considered in this research, however, the size of the data set may not be significantly larger than the size of the space partition (in most scenarios, the ratio of the size of the data by the size of the space partition is less than 10). Using a Gaussian function could very well make up for problems introduced by the grid partition for a continuous density space.

The second improvement to the original CLIQUE algorithm is to separate the computation of the clustering algorithm from the partition setting and density measure. In other words, the data size and the grid partitions of the space should not influence results. To achieve this, two functions must be dynamically implemented. First, the kernel density function must be able to compute the Gaussian distribution based on size of the grid partition. Second, the density value must be adjusted by the amount of data and the size of the space partition, so that the density value becomes a ratio value rather than a density value. In our system, if the density value of a grid cell is 1, it equals the average value if all the data points are uniformly distributed on the whole space.

These two improvements mainly affect how the density space is built before the clusters are generated. Table 5.6 shows how to build the density space in the modified algorithm.

Table 5.6 Algorithm of build density space

<pre>Algorithm space = buildSpace(data, partition) for all the points in data set compute the scope of the density space; grid = divide the density space into grids by partition; for each point in data compute the Gaussian distribution based on partition size; apply the Gaussian distribution on the density space from the location of the current point; ratio = the sum of the Gaussian distribution / the grid size; for each cell in grid compute the adjusted density rate;</pre>
--

Figure 5.7 illustrates an example of a density space after the distribution functions of all the data points have been applied. The data is the distance signal that comes from the Breve simulation, when Canis is trying to approach a ball from the front (in a scenario similar to the ones described in Chapter 2). The horizontal plane in the figure represents the scope of the signal variation. The plane has been partitioned into a 25x25 grid. The height value of each cell represents the density in that location; zero values represent no data in that location. This is a 3D figure of the space before applying the density threshold to screen out the high-density areas. Four separated areas can be found in the figure. Two of the areas contain more than one local maximum.

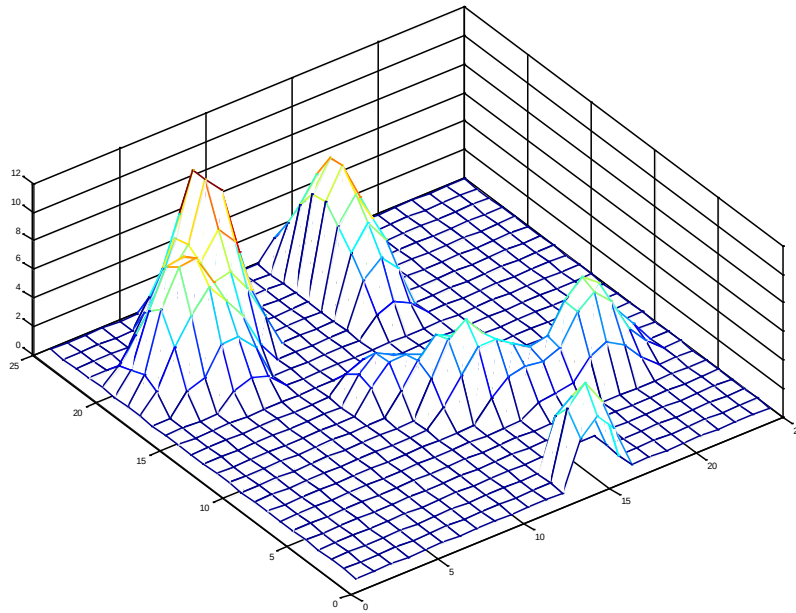


Figure 5.7 An example of a density space

In the original CLIQUE algorithm, the next step is to select a threshold for the density measure, i.e. any cell above that threshold remains; otherwise it is discarded. However, the question is how to decide on an appropriate value. For example, the following two figures show the contour maps of applying different thresholds on the upper density space. Figure 5.8 uses a threshold of 1, which results in four clusters on the map. And Figure 5.9 uses a threshold of 4, which results in five clusters. Area 3 in Figure 5.8 separated into two areas, area 3 and area 4. Generally speaking, a lower density threshold tends to retain more information but also allows more data points to fall into

the same cluster. On the other hand, higher thresholds will lose some information and will also separate data points from each other.

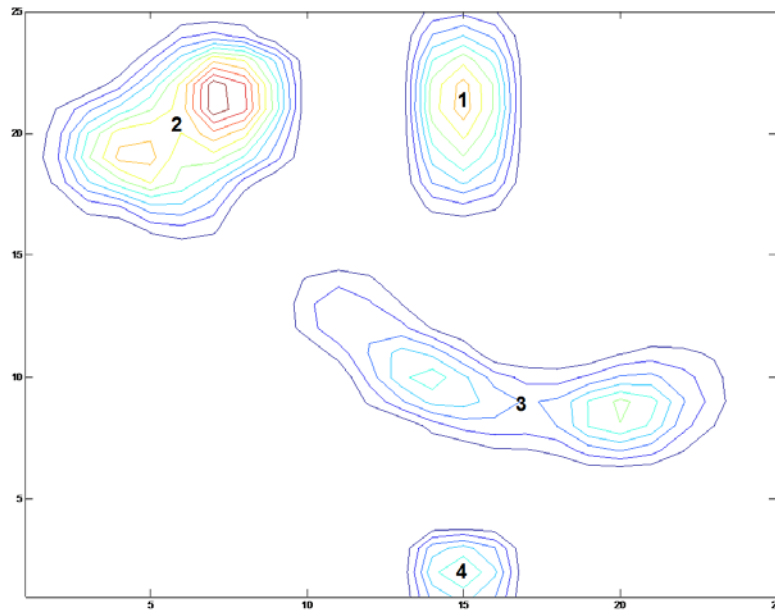


Figure 5.8 Contour map on threshold 1

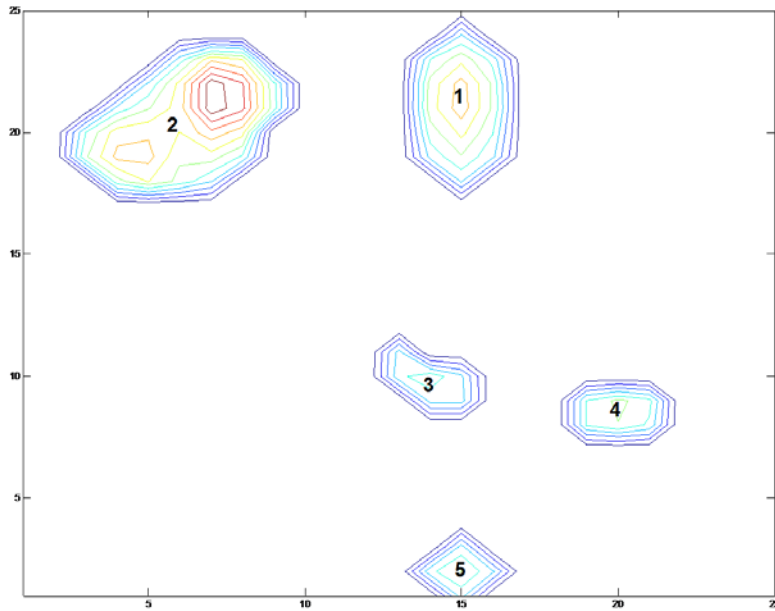


Figure 5.9 Contour map on threshold 4

One final modification to the original CLIQUE algorithm is the inclusion of an algorithm that is less affected by the density threshold. In fact, the algorithm we implemented takes a minimal threshold to have less impact on information loss (as in Figure 5.8), but it also applies its analysis

recursively, so that sub-clusters may be generated (as in Figure 5.9). As a result, the algorithm generates a hierarchical structure on the clusters, in which clusters are combined to generate larger clusters. Figure 5.10 shows the general idea, i.e., 8 clusters are generated. Among them, area 4 is the combination of area 2 and area 3; area 7 is the combination of area 5 and area 6. All clusters and their hierarchy relations are recorded.

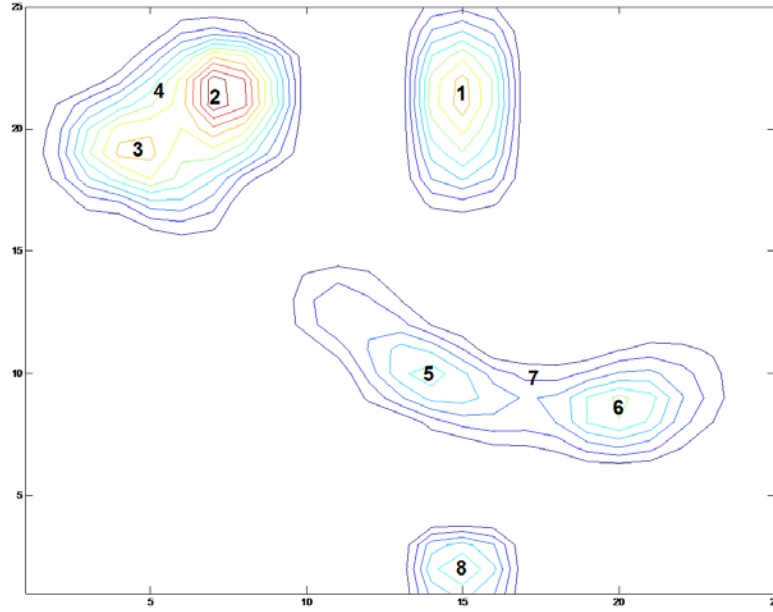


Figure 5.10 Contour map in modified algorithm

The algorithm starts by looking for local maxima in the density space. These local maxima represent possible positions for clusters. They are the seeds of the clusters, because as the threshold value increases, these remain until the very end. In Figure 5.10, these seed positions are labeled 1, 2, 3, 5, 6, 8. The algorithm then descends along the contour lines. As the algorithm scans an area, if it's only adjacent to one cluster, then the area will be added into the cluster it is adjacent to. Therefore, those seeds become larger as more area is treated as being part of those clusters. If a cluster contains more than one sub-cluster, then there must be at least one point that connects more than one of the sub-clusters at the same density. At this time, those sub-clusters are combined into a bigger cluster, and all the sub-clusters will stop growing. The algorithm also examines whether the sub-clusters are too small to be considered "real" clusters. If so, the sub-cluster will be keep in the record. For example, in Figure 5.10 area 7 is the combination of area 5 and 6; area 4 is the combination of area 2 and 3. The algorithm runs until all the area has been scanned and all the clusters are generated.

The detailed algorithm is described in Table 5.7. Notice that scanning by decreasing density can be achieved by sorting the grid cells from high density to low density.

Table 5.7 Algorithm of cluster growth

```

Algorithm ClusterGrow(density area)

cell_queue = Sort all density areas in decreasing order by the density measure;

for cell_queue is not empty
    cell = cell_queue.pop();
    if cell is connect with no cluster
        create a new cluster based on cell;
    elseif cell is connect with only one cluster
        expend the cluster area by including cell;
    else
        create a parent cluster based on all the clusters connected with cell;
        stop growing for all the children clusters;
        for all the children clusters
            if it does not satisfied the cluster criterion
                remove it from the cluster list;
end;

```

After applying the modifications to the original CLIQUE algorithm, we need to estimate the computational costs comparing to CLIQUE. There are two parts to the CLIQUE algorithm: building the density space and searching for high-density clusters. To build the space, CLIQUE partitions the space into k cells, and applies all the data samples (size n) one by one into the cells. Together, it takes $O(k+n)$ time in the original CLIQUE algorithm. In the modified algorithm, the partition processing is the same. By applying the distribution function of each data sample, it changes the cost from 1 (for a simple cell) to a constant number of neighbor cells. Therefore, the modified algorithm still takes $O(k+n)$ in building the density space. To search for high density clusters, the modified algorithm uses the same algorithm to find the consecutive areas which are above the density threshold. The difference is that the modified algorithm will apply the cluster growth algorithm in Table 5.7. So the computational cost of cluster growth algorithm becomes the difference in applying the original CLIQUE algorithm and the modified algorithm. There are two parts to operations in the cluster growth algorithm. The first is to sort the consecutive areas found by CLIQUE, which takes $O(m \log m)$ (if the size of the areas is m). The second part is to add those areas by decreasing order into a cluster list, which is $O(m)$ in computational costs. Together, the cluster growth algorithm will take an extra $O(m \log m)$ time, where m represents the consecutive areas that

are found by CLIQUE algorithm. Therefore, the modified algorithm will take the same $O(m \log m)$ time in computational cost, where m represents the areas that are denser than the preset density threshold.

To summarize, we improve the CLIQUE algorithm in the following respects:

1. We adopt the idea of kernel density estimation, which changes the contribution of every sample data from just one point to a series of points under the same distribution function (a Gaussian function).
2. We normalize the computation of density and Gaussian distribution so that they are irrelevant to the data size and the partitions of the space.
3. We generate a hierarchical structure of clusters that is robust with respect to the density threshold.

5.2.4 Prototype Definition and Description

Although the clustering algorithm is the focus of prototype construction, the goal of this stage is to have a symbolic description of sensorimotor interactions to provide a foundation for later analysis.

After clusters have been identified, they are labeled. A typical clustering algorithm only needs to label the data points that form the cluster space. It's not always necessary to explicitly label the clusters themselves. However, our work not only requires having appropriate labels on already-processed data; it also relies on these clusters to become classifiers for data that has not yet been seen. Further, after being used to label the sensorimotor data, the labeled clusters will be used in a symbolic representation; they become the bridge that connects the numerical layer and the symbolic layer. In other words, they provide sensorimotor interpretations for symbolic descriptions. Combining these two functionalities – classification and interpretation – the role of the cluster become the same as the role of a prototype in the concept theory of human cognition. This is the reason we characterize this stage as the description of prototypes.

As discussed above, the first task in describing prototypes is to label a prototype. We call this a *prototype definition*. Since the prototype connects sensorimotor data and a symbolic description, there are two parts to the information contained in the prototype definition (see Table 5.8),

Table 5.8 Prototype Definition

Symbolic Information of a Prototype
{
Prototype name; // an unique name representing the prototype
Prototype signal name and index;
// Two parts of information are used to generate the unique prototype name – signal name, the signal the prototype is generated from; and index, the generating sequence of the prototype in the same signal.
Prototype parent;
// this feature of having a hierarchical structure of prototypes gives the system more flexibility in generating the prototypes and more precision in finding the association between prototypes.
}
Sensorimotor Information of a Prototype
{
// sensorimotor information is organized by the parameters (features). There are two parameters per item.
Prototype centroid;
// represents the most typical data belonging to the same prototype. This information is also useful when we want to translate back to the sensory layer from symbol layer.
Prototype scope;
// records the variance for each parameter in this prototype. More specifically, we use a min-max pair to represent each parameter in the log file. This information provides the main method for classification.
}

In the system implementation, the sensorimotor part of a prototype is mainly used for classification. The symbolic part of a prototype is used in the symbolic description. For example, the earlier figure shows the readings from a distance sensor when a simulated robot is trying to approach a target in front of it (more details can be found in Scenario 1 in the system evaluation section). Table 5.9 is a part of the prototype data generated from the areas in Figure 5.10,

Table 5.9 An example of prototype data

distance1	//the first prototype in this signal represents the area 8
3.40318 2.48078 4.31338	//center is at (3.4, -0.02), similar to a “NEAR” concept.
-0.0211393 -0.0413371 0.00049438 1	
distance2	//the second prototype in this signal, represents the area 7
10.377 7.97857 14.3927 1	//its center is at (10.3, 0.015) but it has a large variance.
0.0151105 -0.125 0.146905 1	//similar to the concept “PASSING” the target.
distance3	//the third prototype, which represents area 4
20.1396 16.2253 23.5557 1	//similar to the concept of “APPROACHING”
-0.206498 -0.292326 -0.145916 1	
distance4	//the fourth prototype, which is the area 1
20.8499 17.1416 24.4719 1	//represents the concept of “FAR”
-0.0209492 -0.0413371 0.00049438 1	

After these prototypes are defined, the original sensorimotor data which is recorded in piecewise linear form now can be transformed into a symbolic description. We call this a *prototype description*. This transformation can be achieved by classifying the piecewise linear representation using the sensorimotor part of each prototype (Table 5.9). As a result, the prototype description describes the sensorimotor interactions between the robot and the environment.

In the system implementation, the prototype description follows the following format:

<Timestamp>, <controller symbols1>, <controller symbols2> ..., <sensor symbols1>, <sensor symbols2> ...

The order of the controllers and sensors is defined in the scenario script file. If there are tools involved in the interaction, “<tool symbol>” is inserted between the controller symbols and sensor symbols.

The following is an example of a prototype description when a simulated robot is trying to approach a target in front of it (Scenario 1). There are three controller signals that represent forward speed, rotation speed, and stretch speed (a controller that allows the robot to move its head forward and backward). There are also three sensor signals, which are distance from the target, direction between the target and the robot, and contact (whether or not the robot has made contact with the target).

Since the prototype description only uses prototype names to represent the data, it needs to combine with the prototype definition to capture the “meaning”. In Table 5.10, we interpret the

prototype with an English-language, similar concept in blue font. Text in bold font shows the changed prototype compared to the last timestamp.

Table 5.10 An example of symbolic representation

0 forward1 rotation1 stretch1 distance4 direction2 contact1
//Time 0: “STOP”, “No Rotation”, “No Stretch”, “FAR”, “CENTER FRONT”, “NO CONTACT”
20 forward2 rotation1 stretch1 distance4 direction2 contact1
//Time 20: “Moving with speed 5”, “No Rotation”, “No Stretch”, “FAR”, “CENTER FRONT”, “NO CONTACT”
25 forward2 rotation1 stretch1 distance3 direction2 contact1
//Time 25: “Moving with speed 5”, “No rotation”, “No Stretch”, “APPROACHING”, “CENTER FRONT”, “NO CONTACT”
72 forward2 rotation1 stretch1 distance7 direction2 contact1
//Time 72: “Moving with speed 5”, “No rotation”, “No Stretch”, “APPROACHING”, “CENTER FRONT”, “NO CONTACT”
92 forward2 rotation1 stretch1 distance1 direction2 contact2
//Time 92: “Moving with speed 5”, “No rotation”, “No Stretch”, “NEAR”, “CENTER FRONT”, “CONTACT”
111 forward2 rotation1 stretch1 distance1 direction2 contact2
//Time 111: “Moving with speed 5”, “No rotation”, “No Stretch”, “NEAR”, “CENTER FRONT”, “CONTACT”

5.3 Action Schema Discovery

Up to this stage, the sensorimotor data streams have been analyzed and transformed into prototype definitions and descriptions. The information records the activities of a robot system. The prototype description is a live data flow, which means that it supports a hypothetical higher-level system with online decision-making capabilities. It also contains every aspect of information gathering from the sensorimotor data, so that it provides a full description of the system and the environment.

However, there’s more that can be learned from the prototype description, and the embodied meaning of a prototype is also more than just a sensorimotor definition. This section discusses how connections between prototypes can be identified. As the discussion above about human concepts suggests, having connections to the external environment is also critical to forming meaningful concepts.

5.3.1 Knowledge for Learning

In this system design, time series segmentation and prototype construction provide well-organized data as a foundation for knowledge discovery. Time series segmentation transforms the time series data into measurable vectors, a process that is critical for algorithms in data mining and learning. Prototype construction finds repeatedly appearing primitives and categorizes them under the same concepts. This gives the system a basis for anchoring its embodied experiences in the same categorization. An important thing left is to link these concepts in a knowledge base and to study how they transition from one to another.

One of the patterns we see in Table 5.10 is that some of the prototypes remain unchanged during the trial, while others represent a typical sequence in an action we can describe as “approaching”. The key to our work in this area is to find the prototypes that have a high chance of co-occurrence. This relationship can exist along two different dimensions. First, the prototypes may happen at the same time but apply to different signals; for example, when the robot is moving forward, the distance reading to the target will decrease. Second, they may happen in the same sensor signal as a sequence; for example, a decreasing distance sensor reading may either lead to a “stop” (where the robot is blocked) or a “passing” (where the distance starts increasing again). Both kinds of relationships serve important roles for a higher-level system in making reasoning or planning decisions.

5.3.2 Action Schemas

Before discussing an algorithm that can find the rules, we need to limit what kinds of rules we want to discover, and what kind of representation we use to capture the rules. Although different kinds of relations might exist in a prototype description, it’s the rules about actions that are most important for a robot agent. For a planning agent in particular, the rules about the actions the agent can take are the core of its knowledge base.

For these rules, there are three fundamental elements – the conditions that allow the action to take place, the effects of executing the action, and the action itself. To include these three elements, the most widely used action schema in the literature of robotic planning systems is as in Table 5.11.

Table 5.11 Structure of the action schema

Action schema
{
preconditions;
action;
effects;
}

In this schema, the *preconditions* slot describes the world status before the action is executed. In our research, they are captured by the robot’s sensor signals. The *action* slot is captured by the controller signals. The *effects* slot is the difference in the robot’s sensor signal after the action is taken. Therefore, in our system, we use the combination of {sensor signals, controller signals, sensor signals} in time sequence to capture the meaning of an action. For example, Table 5.12 is one action schema captured in the last scenario.

Table 5.12 Approaching action

Action “Approaching”
{
preconditions: distance4 direction2 contact1
// means “ FAR ”, “ CENTER FRONT ”, “ NO CONTACT ”
action: forward2 rotation1 stretch1
//means “ Moving with speed 5 ”, “ No Rotation ”, “ No Stretch ”
effects: distance3 direction2 contact1
//means “ APPROACHING ”, “ CENTER FRONT ”, “ NO CONTACT ”
}

5.3.3 Schema Generation

In the most action schemas, many symbols remain the same from preconditions to effects. Worse, if a robot or the environment gets complicated, more and more sensorimotor signals will be included, and most of them will be irrelevant. In fact, a complete description of the entire robot world (itself and environment) has already been included in the prototype description during the prototype construction stage. There’s no need for an action schema to have the same complete description as well.

If we allow the action schema to focus only on the changing part of the sensorimotor data, however, it not only simplifies the computation and expression of the action schema, also allows the system to focus on the most interesting part of the data.

For those reasons, a simple differential analysis becomes a useful procedure in generating action schemas. This analysis keeps only the changing components between adjacent description sentences and removes whatever remains unchanged. For example, the last scenario will become Table 5.13 after the unchanged symbols in the adjacent sentence are removed.

Table 5.13 Symbol representation after differential analysis

0	forward1	rotation1	stretch1	distance4	direction2	contact1
//Time 0: “STOP”, “FAR”, “CENTER FRONT”, “NO CONTACT”- the initial state						
20	forward2					
//Time 20: “Moving with speed 5” - this is the action						
25			distance3			
//Time 25: “APPROACHING” - the effect						
72						
//Time 72: - empty sentence, everything is same						
92			distance1,	contact2		
//Time 92: “NEAR”, “CONTACT” - final effect						

With a few rules for differential analysis, the key points in a prototype description become clearer. This analysis reduces dependence on the number of sensor streams, and it allows the system to focus on the relevant effects of an action.

Focusing on fewer symbols enables time-consuming methods like association analysis to be applied. Association analysis is a methodology for discovering interesting relationships hidden in large data sets. These relationships are expressed in association rules of the form $X \rightarrow Y$ (Tan et al, 2006). To evaluate the utility of such rules, there are two measures used in association analysis, support and confidence. *Support* expresses how often a rule appears compared to the size of the data set. *Confidence* determines how frequently the rule $X \rightarrow Y$ holds comparing to the premise of the rule, X . In other words, support is the probability of the rule $X \rightarrow Y$, and confidence is the conditional probability of the rule $X \rightarrow Y$, given its premise X . The formal expression can be defined thus:

$$\text{Support, } s(X \rightarrow Y) = \frac{|X \cap Y|}{N}$$

$$\text{Confidence, } c(X \rightarrow Y) = \frac{|X \cap Y|}{|X|}$$

In association analysis, support and confidence are mainly used to separate useful rules from seldom-appearing rules. In this research, we evaluate the action rules in the form of

preconditions & action $\rightarrow$ effects

That is, given the preconditions and the action, such a rule represents the effects an action could produce. The support and confidence provide important measures. We want to find the action rules that have high confidence and adequate support, measures of the efficiency of an action. An efficiency factor of the action schema should represent how well this action will take effect.

Table 5.14 explains the algorithm for generating the action rules.

Table 5.14 Algorithm of building action scehma

Algorithm buildAction <pre> for all the trials preconditions = get the first sentence of the trial; for time = second timestamp to the last timestamp if controller signals are changed action = changing symbols in controller signals; if sensor signals are changed effects = changing symbols in sensor signals; if action & effects are updated schema = build(preconditions, action, effects); if schema is new add schema into schema_list; else update schema measure in schema_list; endif; preconditions = current sentence of the trial; end; end; for all the schemas in schema_list sort by the confidence and support measure; </pre>
--

After applying such algorithm, two schemas can be generated from the early scenario in Table 5.15.

Table 5.15 Example of two action schemas

Action "Approaching"
{
preconditions: distance4 direction2 contact1
<i>// means "FAR", "CENTER FRONT", "NO CONTACT"</i>
action: forward2 <i>//means "Moving with speed 5"</i>
effects: distance3 <i>//means "APPROACHING"</i>
}
Action "Approaching"
{
preconditions: distance3 direction2 contact1
<i>// means "APPROACHING", "CENTER FRONT", "NO CONTACT"</i>
action: forward2 <i>//means "Moving with speed 5"</i>
effects: distance1 contact2 <i>//means "NEAR", "CENTER FRONT", "CONTACT"</i>
}

Chapter 6

Evaluation

The general goal of this research is to acquire semantic knowledge for higher-level reasoning behaviors, in particular knowledge relevant to the activities associated with tool use, from the experience of a cognitive agent embodied in either a simulated environment or the real world. The research has involved examining cognitive theories, designing a general component (an Action Schema Generator) for a computational cognitive architecture, and implementing techniques from different areas of artificial intelligence. Evaluating such work is complicated; in general there are no standard procedures for evaluation, either with respect to tool-using agents or even more generally with respect to cognitive robotic agents. We also find no experiments with human or animal behavior that provide data at a sufficient level of detail to validate the performance of the system described here.

This is not an unusual situation in AI research, in particular in the evaluation of complex agent architectures, in part or in whole. A pragmatic approach, adopted here, involves two stages of evaluation, at the system level and the component level. At the system level, an agent is built to embody the architecture or architecture components. In our case, this component is the action schema generator, and the agent is the Canis agent in a Breve simulation (with the physical Aibo robot substituting for Canis in some situations). The agent is evaluated with respect to sufficiency in accomplishing a range of tasks for which its architecture has been targeted: tool-using tasks.

At the component level, individual algorithms and representations are analyzed to identify their strengths and weaknesses. In this research, there are three such components, corresponding to

the three stages of processing: time series segmentation, prototype identification, and action schema discovery. Each is evaluated with respect to different criteria; the focus is not on hypothesis testing but rather on understanding how well these components meet their requirements, most importantly in how they contribute to higher-level processing.

In this section, we will first explain the simulation scenarios built in Breve and how the training data are collected and organized. Then to address component level evaluation, we will explain the evaluation method and discuss the results. To evaluate this research at the system level, we have designed and implemented a forward-planning algorithm that can be used to apply the knowledge learned in each scenario. The goal of these evaluation activities is to examine the functionalities of the current system as well as the limitation the system architecture, and to assess support for higher-level robotic applications.

To summarize the results, we find that the Action Schema Generator achieves the primary goals as an intermediate layer in a robotic system:

1. Automatically translate sensorimotor information from numerical raw signals to a symbolic representation without prior knowledge about the scenarios or the action patterns.
2. Capture key concepts in different situations of training scenarios. Even though the sensorimotor signals vary in number, scope, and behavior, the prototype description and action schemas generated by the system records enough information for a planning system to support effective reasoning in all test scenarios.
3. Learn knowledge from observing the training scenarios. The training scenarios could include multiple objects, multiple actions altogether, and can have multiple outcomes. Notice that there's no explicit prior knowledge about the scenarios in the system.
4. Make certain levels of generalization even with the simplest algorithm, like closest match. By default, the prototypes learned from the scenario represent the experiences in the training trials. How generally the system can apply the knowledge to new trials depends on how broadly the prototypes can be matched.
5. Accumulate knowledge by combining multiple scenarios. This is very important since it can help a high-level system to make reasoning based on accumulated memory or global knowledge.

6. Support tool-use behaviors, and allow higher-level system to discover the difference between the non-tool-use and tool-use behaviors, therefore to discover the effects of using such a tool.

Of course, we have also learned about the limitations associated with the current system. They can be summarized as follows:

1. The system lacks a mechanism to decompose actions in time, once they have been learned. For example, a long-running action might involve approaching a distant object, with a given precondition and effect; the system cannot make inferences about shorter-term actions that might be derived if the precondition and effect were to be modified.
2. The system lacks a mechanism to capture the inter-relationships between multiple objects. In other words, in the current system, prototypes learned for one object have no relationship to those of other objects, and all relationships are with respect to the agent itself.
3. There remains a lot of redundant information in both the prototype definitions and action descriptions, which increases the complexity of planning and of combining knowledge.

6.1 Simulation Scenarios

In order to evaluate the system performance on different aspects, we designed six scenarios in the Breve simulation. These six scenarios cover a few standard behaviors of a robot, including basic navigation and combinations of actions; they also include two fundamental tool-using behaviors. The first is reaching an object that is inaccessible without the use of a tool for reaching (Beck, 1981); the second is the use of a tool to test for the presence of an object (e.g., Breuer et al., 2005).

In all the scenarios, an agent is designed as a simplified version of Sony Aibo, as described briefly in Chapter 2. In each scenario, there are several designed situations. Each situation is run for at least 20 trials, depending on the number of relevant parameters. All data are generated by the built-in Breve physics engine. In order to simulate more realistic environments, we also include random parameter noise on every trial setting. Therefore, even though two trials may be from the same situation, they won't be the same due to the random variation in the initial state. In a trial file, the first column is always the time stamp measuring the system time on seconds. The second column is a valid tag which is always 1 in the Breve simulation (the reason to keep this column is to be compatible with the data of Sony Aibo). The next three columns are always the parameters of all

three Canis controller – forward, rotation and stretch. And the rest columns are the sensory signals, which vary by scenario. A data example can be seen in Figure 6.1,

```
time valid forward rotation stretch distance direction contact
22.288000 1 0.000000 0.000000 5.000000 19.928074 0.017564 0.000000
22.388000 1 0.000000 0.000000 5.000000 19.929309 -0.017457 0.000000
22.488000 1 0.000000 0.000000 5.000000 19.929176 -0.017467 0.000000
22.588000 1 0.000000 0.000000 5.000000 19.940519 -0.018256 0.000000
22.688000 1 0.000000 0.000000 5.000000 19.957108 -0.019361 0.000000
22.788000 1 0.000000 0.000000 5.000000 19.933322 -0.017746 0.000000
22.888000 1 0.000000 0.000000 5.000000 19.923066 -0.017489 0.000000
22.988000 1 0.000000 0.000000 5.000000 19.922811 -0.017479 0.000000
23.088000 1 0.000000 0.000000 5.000000 19.922809 -0.017482 0.000000
23.188000 1 0.000000 0.000000 5.000000 19.922807 -0.017484 0.000000
23.288000 1 0.000000 0.000000 5.000000 19.922807 -0.017484 0.000000
23.388000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
23.488000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
23.588000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
23.688000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
23.788000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
23.888000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
23.988000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
24.088000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
24.188000 1 0.000000 0.000000 5.000000 19.922806 -0.017485 0.000000
24.288000 1 4.925000 0.000000 5.000000 19.922806 -0.017485 0.000000
24.388000 1 4.925000 0.000000 5.000000 19.910101 -0.017551 0.000000
24.488000 1 4.925000 0.000000 5.000000 19.815244 0.017605 0.000000
24.588000 1 4.925000 0.000000 5.000000 19.672574 0.017781 0.000000
24.688000 1 4.925000 0.000000 5.000000 19.588975 0.017916 0.000000
24.788000 1 4.925000 0.000000 5.000000 19.401964 0.018109 0.000000
24.888000 1 4.925000 0.000000 5.000000 19.191337 -0.018346 0.000000
24.988000 1 4.925000 0.000000 5.000000 18.964164 -0.018504 0.000000
25.088000 1 4.925000 0.000000 5.000000 18.742382 -0.018875 0.000000
25.188000 1 4.925000 0.000000 5.000000 18.639343 -0.018850 0.000000
25.288000 1 4.925000 0.000000 5.000000 18.418897 -0.019028 0.000000
25.388000 1 4.925000 0.000000 5.000000 18.184325 -0.019341 0.000000
25.488000 1 4.925000 0.000000 5.000000 17.996584 -0.019602 0.000000
25.588000 1 4.925000 0.000000 5.000000 17.795341 -0.019710 0.000000
25.688000 1 4.925000 0.000000 5.000000 17.690865 -0.019979 0.000000
25.788000 1 4.925000 0.000000 5.000000 17.473979 -0.020034 0.000000
```

Figure 6.1 Example of raw data

A summary of the data trials used in the whole evaluation in Table 6.1,

Table 6.1 Summary of data trials

Scenario	Situation	Trials in Total
Scenario 1	3	200
Scenario 2	2	160
Scenario 3	3	100
Scenario 4	2	60
Scenario 5	2	40
Scenario 6	4	100
Total	16	660

In the following sections, we briefly explain the design for every scenario.

1. Scenario 1 – Approaching a target

In this scenario, a target (yellow ball) has been placed to the front, front-left, or front-right with respect to Canis (Figure 6.2). Canis moves forward in every trial. There are three sensory

signals in this scenario. They are distance – the relative distance from Canis to the target; direction – the relative direction of the target to Canis in radius; and a contact indicator – which indicates if Canis has made contact with the target yet. The trial ends when Canis either makes contact with the target or passes the target.

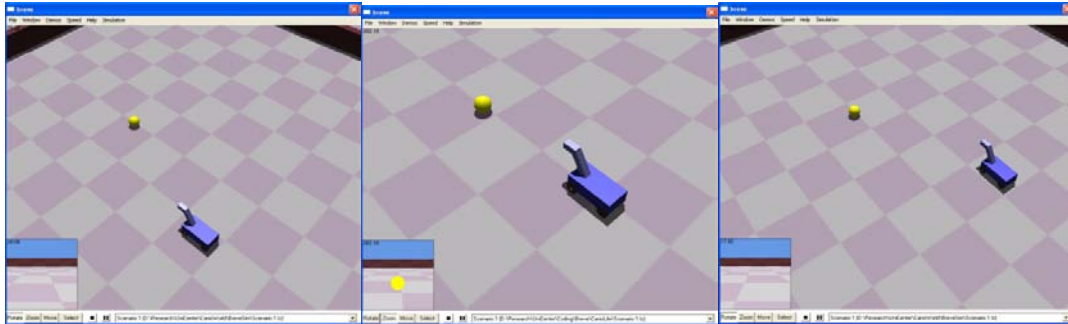


Figure 6.2 Simulation pictures of scenario 1

2. Scenario 2 – Turning towards a target

In this scenario, a target (yellow ball) has been placed to the front-left or front-right with respect to Canis (Figure 6.3). Canis turns toward the target in every trial. There are the same three sensory signals in this scenario as in Scenario 1. They are distance – the relative distance from Canis to the target; direction – the relative direction of the target to Canis in radius; and a contact indicator – which indicates if Canis has made contact with the target yet. The trial ends when the target is in the front of Canis.

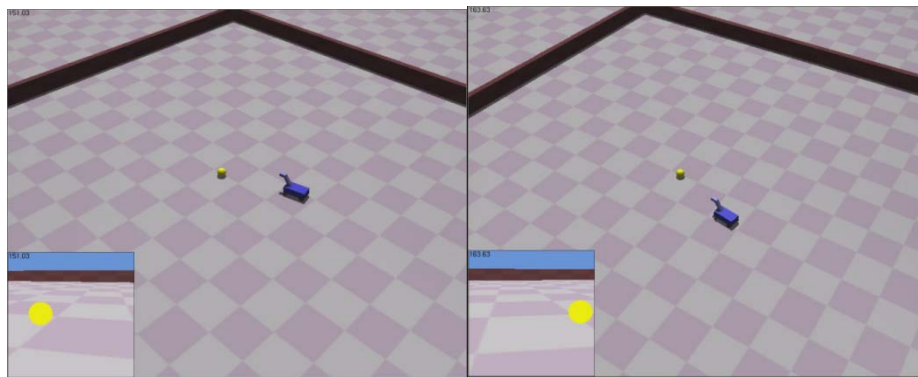


Figure 6.3 Simulation pictures of scenario 2

3. Scenario 3 – Pushing a target off a box

In this scenario, a target (yellow ball) has been placed on a box (brown color) in front of Canis (Figure 6.4). Canis moves directly toward the target in every trial. The size of the box varies in three different situations, so in some cases Canis can push the target off the box by using its head. In the third situation, Canis is augmented with a stick in its mouth; this increases the distance for

making the contact with the target. There are two objects other than Canis in this scenario (the ball and the box), and each has the same three sensory signals as before. They are distance – the relative distance from Canis to the target; direction – the relative direction of the target to Canis in radius; and a contact indicator – which indicates if Canis has made contact with an object yet. There is an extra signal to represent whether Canis has the stick in its mouth or not. The trial ends when the target is pushed off the box or Canis is blocked by the box.

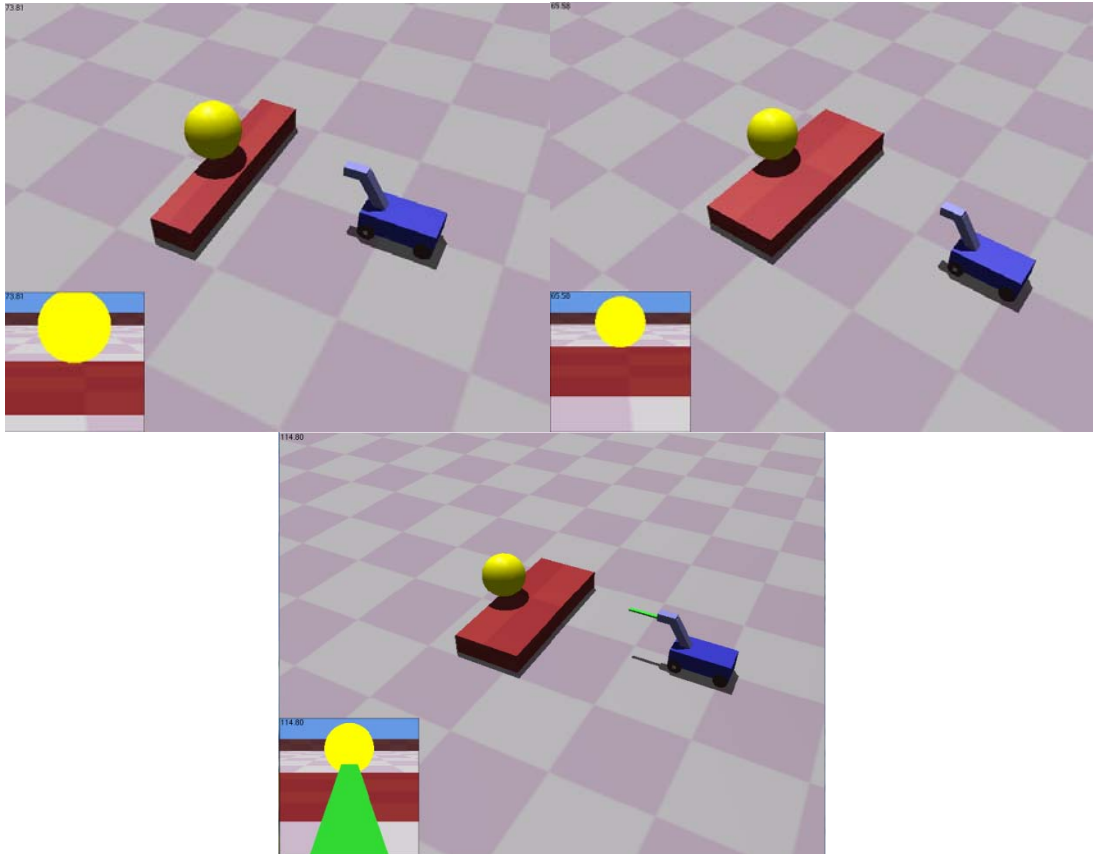


Figure 6.4 Simulation pictures of scenario 3

4. Scenario 4 – Stretching to push a target

In this scenario, a target (yellow ball) has been placed on a box (brown color) in front of Canis (Figure 6.5). Instead of moving forward, Canis will lean its head toward the target in this scenario. The size of the box varies between three situations, as in Scenario 3. In the second situation, Canis takes a stick in its mouth, which will increase the distance for making the contact with the target. The sensory signals are the same as in Scenario 3. There are two objects other than Canis in this scenario (the ball and the box), and each has the same three sensory signals as before.

They are distance – the relative distance from Canis to the target; direction – the relative direction of the target to Canis in radius; and a contact indicator – which indicate if Canis has made contact with the object yet. There is an extra signal to represent whether Canis has the stick in its mouth or not. The trial ends when the target is pushed off the box or Canis is blocked by the box.

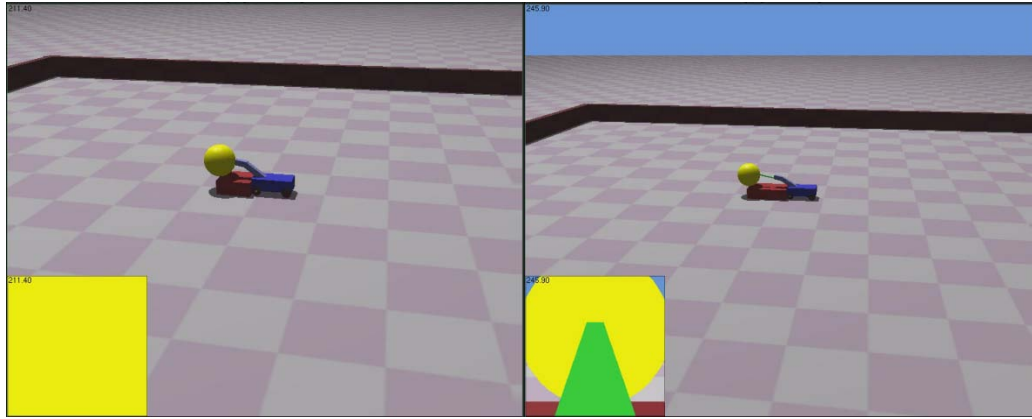


Figure 6.5 Simulation pictures of scenario 4

5. Scenario 5 – Detecting a target

In this scenario, Canis is placed on a box (brown color), and there might be a target (yellow ball) in front of the box (Figure 6.6). Canis will try using a stick while leaning downward to detect if there's something in front of the box. The size of the target varies. There is no direct signal for target's appearance; instead there are only two signals monitoring the angles of Canis's head joint and neck joint. The intention here is to model situations in which the use of a tool changes the type, quality, or even the presence of sensory information. The trial will end when Canis has leaned to its limit.

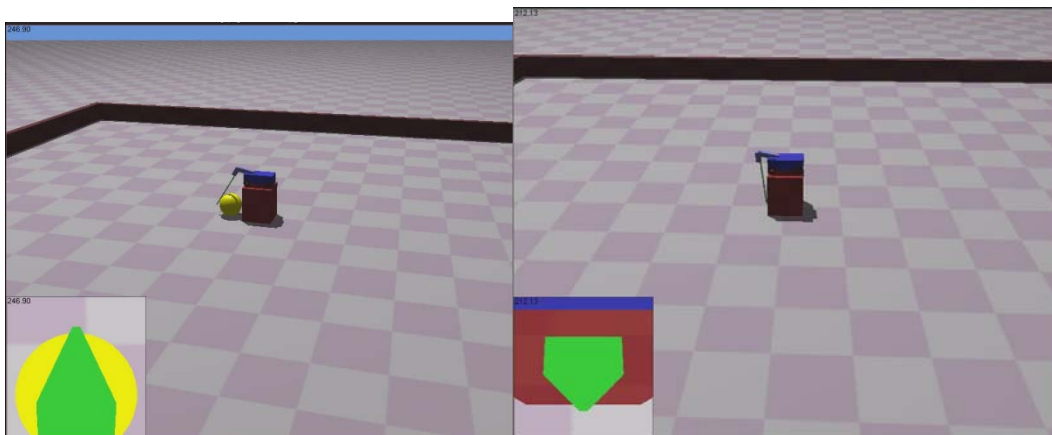


Figure 6.6 Simulation pictures of scenario 5

6. Scenario 6 – Sweeping a target

In this scenario, a target (yellow ball) is placed in a corner (brown color) in front of Canis (Figure 6.7). Canis will try using a stick to sweep the ball from the corner. In this scenario, Canis has to carry out two actions, forward and rotation. There are four variations of situations we generate in this scenario, with Canis starting from either left or right of the target, and Canis sweeping leftward or rightward. Canis may also start its sweeping at different distances from the target. There are four sensory signals in this scenario. They are distance – the relative distance from Canis to the target; direction – the relative direction of the target to Canis in radius; a contact indicator – which indicates if Canis has made contact with the target yet; and corner distance – the relative distance between the target and the point of the corner. The intention here is to model a behavior that is more sophisticated than simply poking or reaching with a stick; this behavior is comparable to using a lever. The trial will end when Canis has swept the target from the corner or been blocked by the corner.

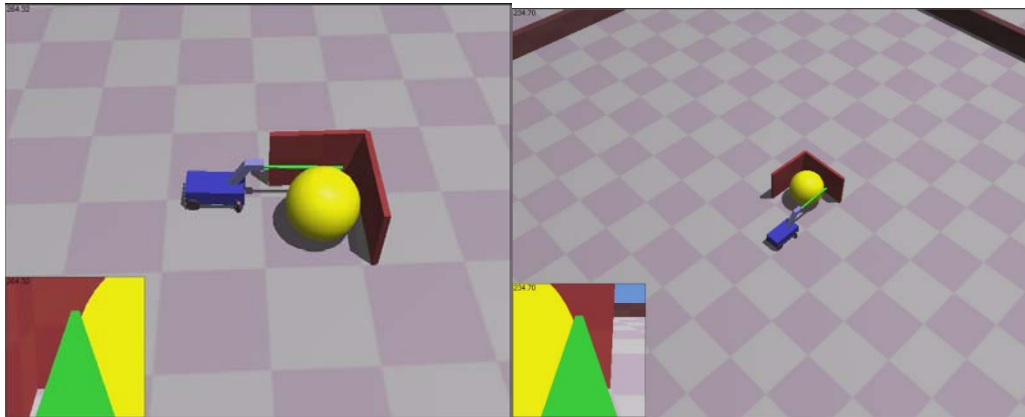


Figure 6.7 Simulation pictures of scenario 6

6.2 Evaluation of Time Series Segmentation

The first stage of processing transforms the data from continuous sensorimotor signals into a piecewise linear representation, without losing the capability of regenerating quantitative data. Both of these directions are important for later stages of processing. For the transformation from raw data to piecewise linear vectors, the system needs the primitives to be measured and segmented from a continuous time series format. The reverse transformation provides a simple and direct way to retrieve numerical data, such as control commands, when needed. Also, when the system needs a

self-monitoring mechanism, like the “emulator” in Clark’s theory of cognitive robotics (Clark and Grush, 1999), having numerical prototype data is an effective way to achieve this functionality.

Since we have established a reversibility criterion for this stage of processing, one means of evaluation is to compare the original signal to the regenerated signal from piecewise linear representation. In this case, we need to have a tolerance for loss of a particular level of details. In general, error is measured by the vertical differences between the original signals and the regenerated signals (Keogh et al., 2003). In the following sections, we will use figures from Scenario 1 to explain this process and discuss the evaluation results from all six scenarios.

Scenario 1 simulates the situations in which Canis is trying to approach a target. It might either pass it on the left or right side, or it may come in contact with the target. There are three controller signals and three sensor signals (signal details can be found in last section). Figure 6.8 illustrates a complete trial of the situation in which Canis makes contact with the target in the center front. In this figure, the horizontal axis represents the time stamp, and the vertical axis represents the signal readings.

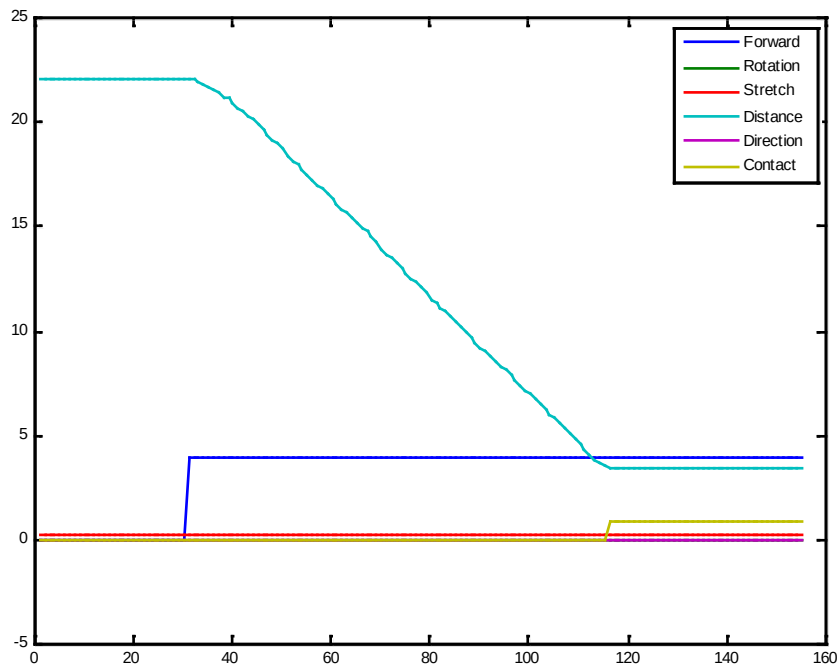


Figure 6.8 Approaching the target in center front

Since some of these readings can be easily discretized without losing detail, like rotation or contact, we will only focus on distance and direction signals in this example. Figure 6.9 illustrates the distance (in blue) and direction (in green) signals in three different situations – when the target is in the front, to the left-front, and to the right-front. We can see that in the first situation, the distance measure keeps decreasing in most of the time, and the direction measure stays roughly constant. In the second and third situations, the distance measure goes to a local minimum while Canis is passing the target, then starts increasing. In the meantime, the direction measure goes all the way to one direction (either left or right).

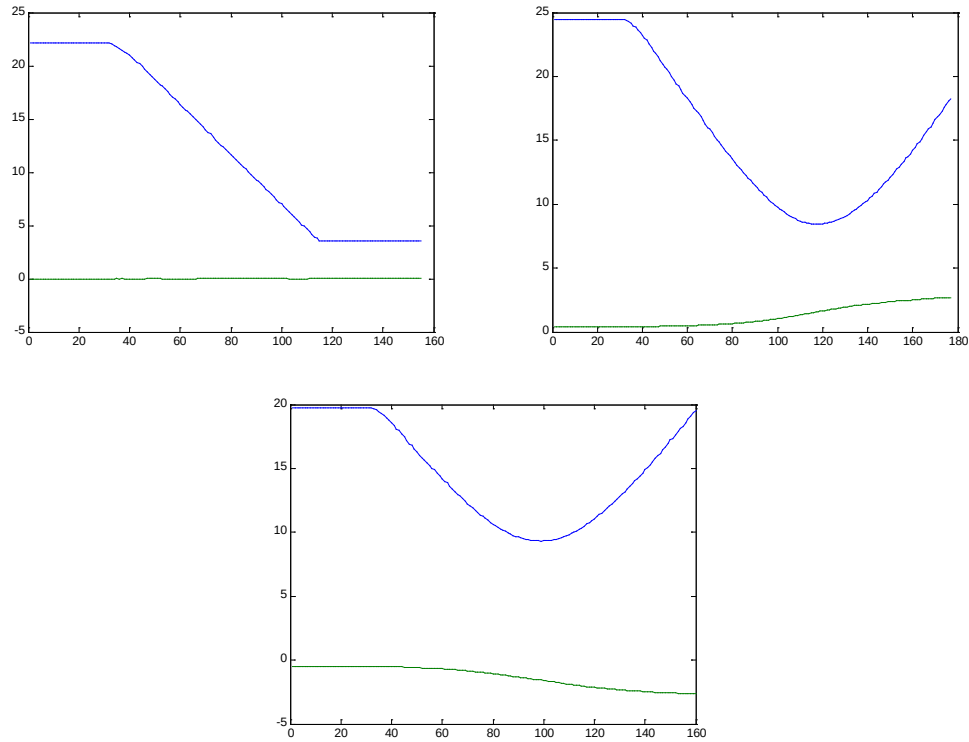


Figure 6.9 Distance and direction data from scenario 1

Transformed into a piecewise linear representation, the output (linear regression) is very close to the original data. Figure 6.10 shows the original data in the left side (the same as above figure) and the recovered data in the right side. In recovered data, the piecewise linear representation data is actually displayed by a set of connected line segments. The more often the signal changes, the more line segments in the recovered data.

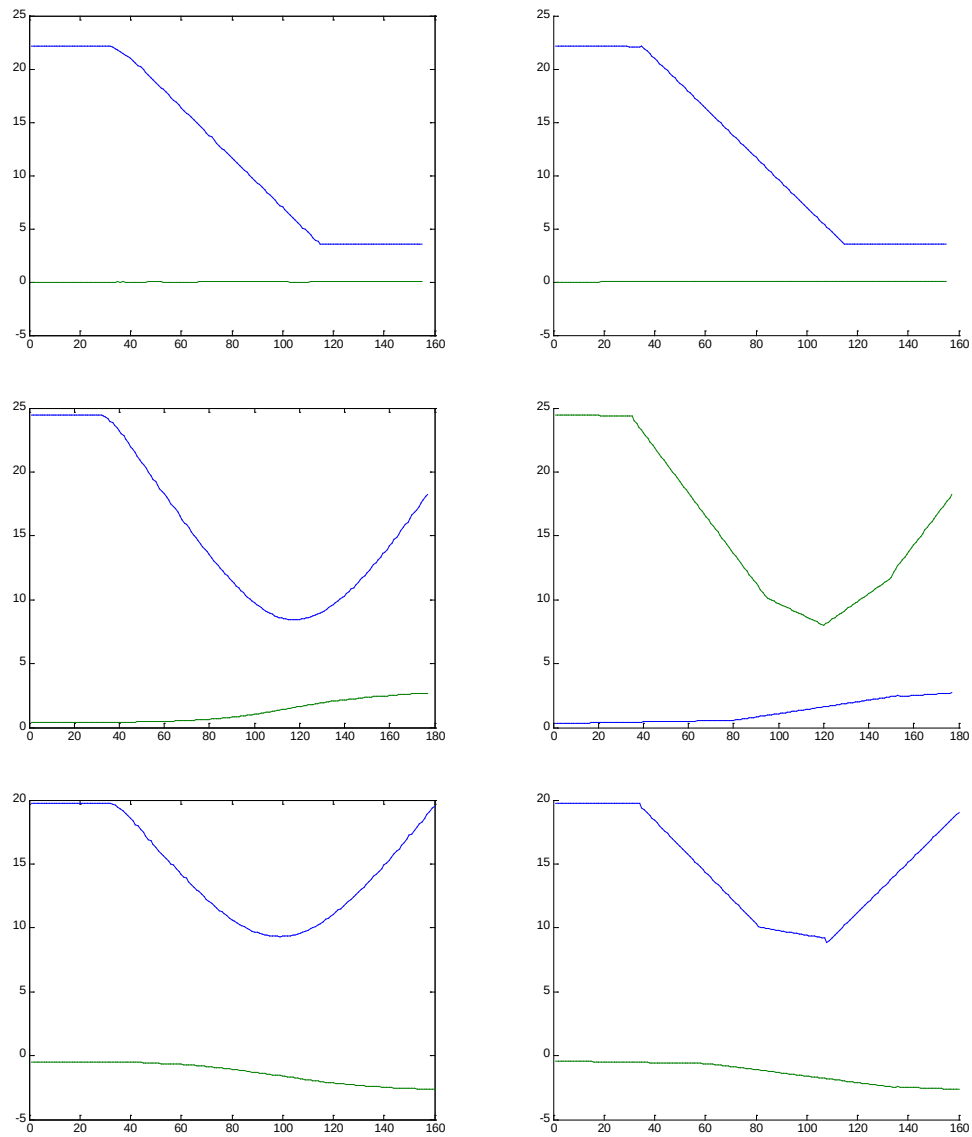


Figure 6.10 Comparison between raw data and recovered data

Pictures on the left side are the raw signal data, on the right are the data recovered from piecewise linear representation data.

Figure 6.11, which combines the data from both sides, may give a better idea about how close they are.

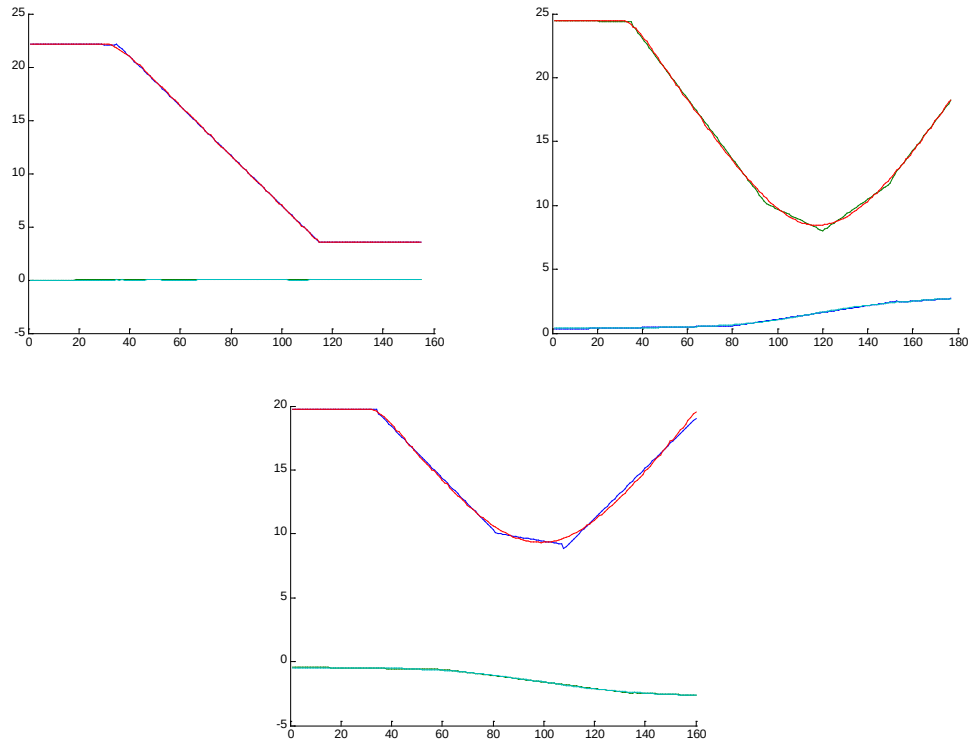


Figure 6.11 Combination of the raw data and recovered data

There are 200 trials in this scenario. As discussed earlier, error is measured by the vertical differences between the original signals and the regenerated signals. We have chosen two types of error measures. The first is the average error per time sample per trial. The second one is the maximum error per trial. We computed the errors over all the signals for all the trial, and also compared them with the signal variations to get an idea about the percentage of each error. Table 6.2 shows the result for scenario 1.

Table 6.2 Time series segmentation errors of scenario 1

	Ave Error	Ave %	Max Error	Max %
Forward	1.13E-14	0.00%	1.91E-14	0.00%
Rotation	0	0.00%	0	0.00%
Stretch	1.89E-15	0.00%	1.91E-15	0.00%
Distance	0.051522	0.21%	0.359324	1.44%
Direction	0.022989	0.73%	0.089847	2.86%
Contact	0	0.00%	0	0.00%

From the table, the first observation is that for signals like “Forward”, “Rotation”, “Stretch” and “Contact”, which are discrete signals, the piecewise linear transformation produces almost no error (the non-zero values shown for “Forward” and “Stretch” are actually introduced by floating number system). For signals like “Distance” and “Direction”, the transformation produces less than 1% error on average and less than 3% at maximum.

We applied the same measure to all six scenarios. For each scenario, we compute the average error percentages for all none-zero-error signals; e.g., for scenario 1, we compute the average on “Distance” and “Direction” only. Figure 6.12 shows the results for average error percentage and maximum error percentage.

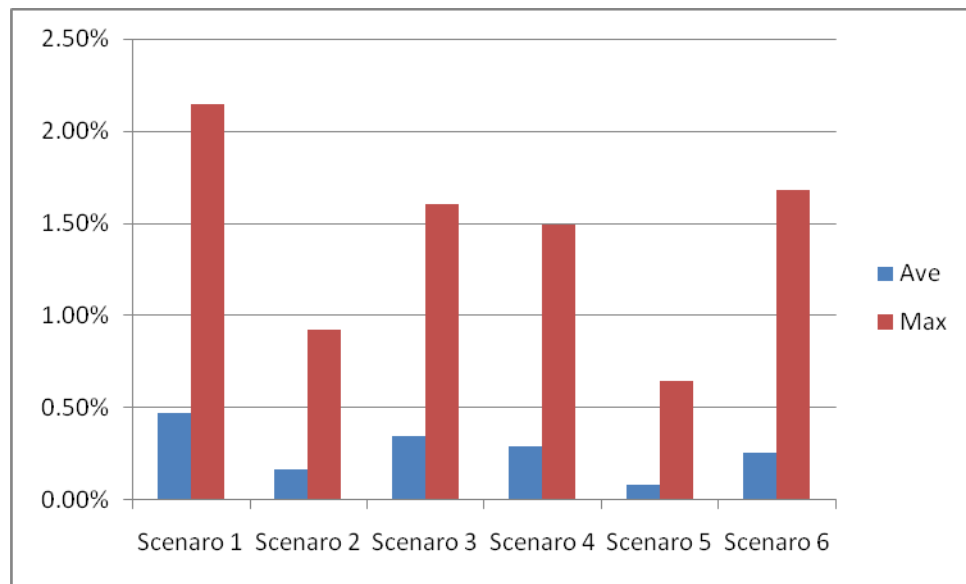


Figure 6.12 Time series segmentation errors

From the results, we can say that the algorithm we use to transform from sensorimotor raw data to a piecewise linear representation is faithful; it keeps most of the details of the original data, and the differences between them can be neglected for the purposes of this research.

We also apply the same method to the sensorimotor data generated by the Sony Aibo robot. We collected 14 trials recoding the signals of Aibo moving in different directions with respect to a target, similar to Scenario 1 in the Breve simulation. Figure 6.13 shows the result for average error percentage and maximum error percentage in those 14 trials. The results only show a little bit higher error than for the Breve simulation. The time series segmentation algorithm still preserves most of the detail from the real robot situation.

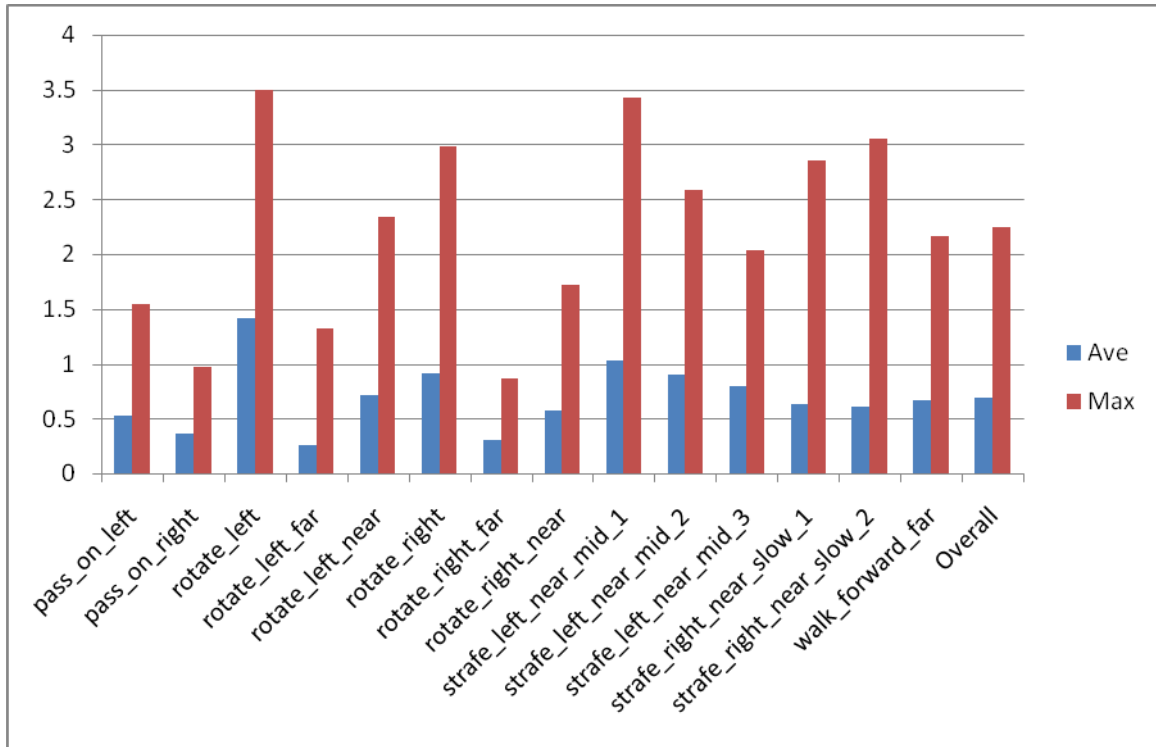


Figure 6.13 Time series segmentation errors in Sony Aibo robot

6.3 Evaluation of Prototype Construction

As discussed earlier, the central part of building prototypes is clustering. The difficulty for evaluation is that there's no "ground truth" for an unsupervised clustering algorithm in this domain: collecting human judgments about how to decompose the scenarios we have developed into concepts is beyond the scope of our work. The ultimate goal of the work, however, is to let the data serve action schema discovery or any higher-level reasoning system better, which means that pragmatic performance will be one reasonable substitute: do the results of prototype generation support successful reasoning and acting in a complete agent? However, there's still a commonly used measure to evaluate the general performance of an unsupervised clustering algorithm, which are the measures of cohesion and separation (Figure 6.14).

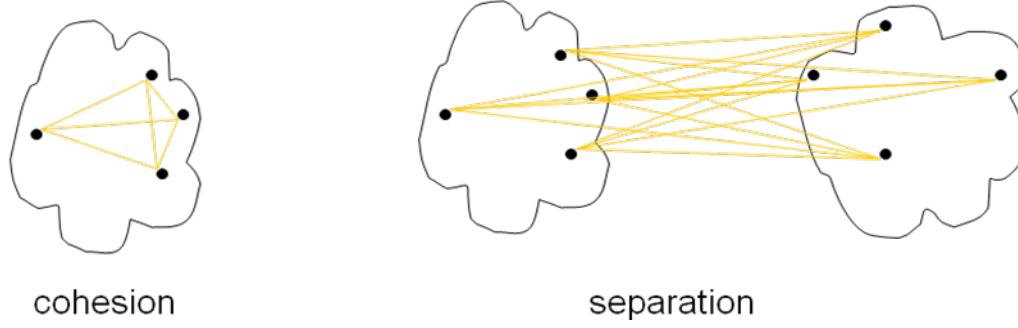


Figure 6.14 Cohesion and separation

Cluster cohesion measures how closely related the objects in one cluster are, as shown in Figure 6.14. Cluster separation, on the other hand, measures how distinct or well-separated the objects in one cluster are from the objects of other clusters (Tan et al., 2006). The general equations for cluster cohesion and separation are as follows:

$$\text{cohesion}(C_i) = \sum_{\substack{x \in C_i \\ y \in C_i}} \text{proximity}(x, y)$$

$$\text{separation}(C_i, C_j) = \sum_{\substack{x \in C_i \\ y \in C_j}} \text{proximity}(x, y)$$

In this research, the density space is built from the parameters corresponding to the piecewise linear representations. We use Euclidean distance as the proximity function. Notice that for the cohesion measure, the smaller the value returned, the tighter the cluster is. For separation function, the larger the value returned, the more distinct the objects are from different clusters. Therefore, a better clustering algorithm should give a low cohesion measure and a high separation measure. Based on cohesion and separation, we can evaluate the overall algorithm validity by this form:

$$\text{overall validity} = \sum_{i=1}^K w_i \text{validity}(C_i)$$

Ech cluster's validity is the result of combining the cohesion and separation measure (Tan et al., 2006), thus:

$$\text{validity}(C_i) = \sum_{j=1, j \neq i}^K \sum_{\substack{x \in C_i \\ y \in C_j}} \text{proximity}(x, y)$$

$$w_i = \frac{1}{\sum_{\substack{x \in C_i \\ y \in C_i}} \text{proximity}(x, y)}$$

The value of the first equation represents the separation, and the coefficient in the second equation represents the cohesion. A better clustering gets a higher result (larger separation and smaller cohesion).

One problem with this validity measure is that there's no universal standard for what is good and what is bad. In other words, a result from one data set can't be compared with results from other data sets. Because of this, we need to compare different algorithms on the same data set. In the following, we evaluate both the original CLIQUE algorithm (cluster by given threshold), and our modified algorithm (identifying more clusters inside bigger clusters). We also compare the evaluation results by the same algorithm with different partition and threshold settings.

Table 6.3 shows the result from scenario 1. Again, we only focus on non-discrete signals like target distance and direction. In the table, we also vary the number of the partition in the density space. Recall that the partition number means how many grid cells the density space has been divided into.

Table 6.3 Validity measures between CLIQUE and modified algorithm

	Partition Number			
	10	20	40	60
Target-distance, Modified Algorithm	177.721	7348.98	2811.1	4964.59
Target-distance, CLIQUE	77.4201	216.954	68.4648	62.4383
Target-direction, Modified Algorithm	49.19	598.677	1191.36	2097.25
Target-direction, CLIQUE	0.320504	121.013	468.906	401.954

It's clear that our modified algorithm scores much higher than the classic CLIQUE algorithm. The main reason for this is that our algorithm generates many more clusters than the original CLIQUE, and those extra clusters, which is called sub-clusters in that chapter's discussion, are located on highly dense areas, which dramatically reduces the cohesion measure by tagging them as a single cluster.

Another observation from the table is that the partition number has influence on the result as well. In theory, the partition number determines how many grid cells are generated for every axis, which is a major factor in the discretizing of the continuous density space. In other words, this number determines how much detail the clustering algorithm can distinguish in the density space. Figure 6.15 explains this idea by using the target direction data from Scenario 1. The left picture is the contour map of the density space when the partition number is 10 per axis, by which our intuitions tell us that there are only three different clusters. However, in the right picture, when the

partition number is increased to 60 per axis, it's pretty clear that there are about five different dense areas located in the center area.

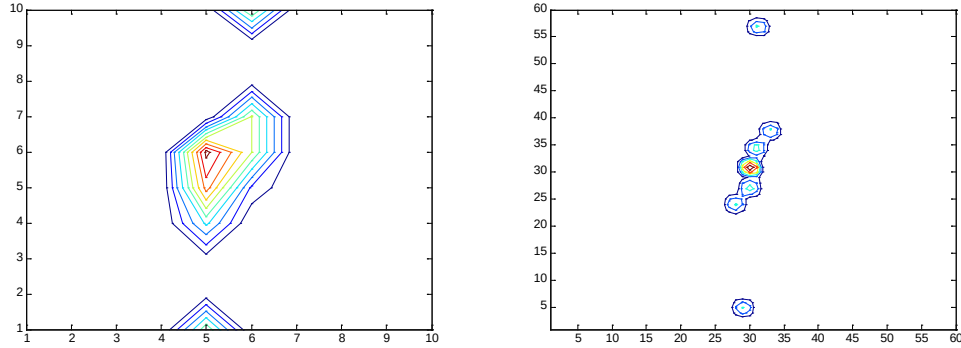


Figure 6.15 Density spaces with different partition size

Making too many partitions has the danger of lowering the validity measure, because there may not have enough training data to cover the area. For example, the highest measure for target distance in Scenario 1 is for 20 partitions, both for the modified algorithm and CLIQUE algorithm. For the scenarios in this research, we choose 40 as the partition number.

The same measure has been applied to all six scenarios. Table 6.4 shows the average measure ratios between the modified CLIQUE algorithm and the original CLIQUE algorithm for all the sensor signals.

Table 6.4 Average validity ratios between modified algorithm and CLIQUE

	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
Average Measure Ratio, Improved/CLIQUE	21.79989	9.832179	1.89829	35.49867	1	45.27956

6.4 Planning Program and System Evaluation

At this point we turn to evaluation at the system level. The primary goal of this research is to study how the generated knowledge from physical robot interactions can benefit a high-level system, like a planning system, and to understand the limitation of the process of generating such knowledge as is done in the system we have developed. For such evaluation, an independent planning program that can acquire and interpret the output of the current system is necessary. Given sensor readings as input, this program will use a forward planning algorithm try to generate a plan

for the goal. In the following sections, we will first explain the implementation of such a program, and then explain the evaluation results using this program on all six scenarios.

Before we explain the evaluation program, we need to emphasize something about the output data from our system. There are three types of the output data (knowledge) generated by the system – prototype definitions, which generalize over numerical time series data; prototype descriptions – a set of prototype labels that correspond to the prototypes and can be used to interpret the sensorimotor signal readings, time step by time step; and action schemas – summaries of the typical preconditions and effects of the actions in a scenario. For the purposes of a planning program, we focus on two types of system outputs – prototype definitions and action schemas.

The planning program takes two types of input. First are the prototypes and actions from the Action Schema Generator for a given scenario; the planning program stores and interprets these. With this process the planning program is given the knowledge base that the ASG learned from scenario trials. The second type of input is a description of the initial and goal states for a given situation. Here the planning program takes the same type of signal readings as was simulated in the scenario.

Next, the planning program interprets the signal readings of the initial and goal states and maps them to the prototypes defined in the system. There are two ways of doing this. One is a restricted match, which means that for a given set of readings, a prototype will only match if they fall strictly within its boundaries. Another is a closest match, which means that if a set of readings does not fall within the scope of a prototype, the nearest prototype will be returned as a match.

The planning program executes a forward, breadth-first search algorithm from the starting prototypes to look for the goal state. The search can also examine all the branches of the search space from the initial state and output a full planning tree which indicates all the possible outcomes from the starting point. Table 6.5 explains the detail of the algorithm.

Table 6.5 Algorithm of forward planning

```
Algorithm runPlanning(init_state, goal_state)

open = create a state list with only init_state;
closed = create an empty state list;

while open is not empty
    state = pop out the first state from open;
    push state into closed;

    if state is a goal state
        generate the action plan from state;
        break;
    endif;

    action_list = find actions whose precondition is compatible with state;
    for all action in action_list
        new_state = apply action in state;
        if new_state is not appeared in open and closed
            push new_state into open;
        end;
    endwhile;

    if action plan is generated
        output the action plan;
    else
        output "no plan is found.";
```

In order to do more evaluations, we can also define a scope of variance in the initial input data, and the planner will report whether a goal state is reached. Thus we can use this function to examine the robustness of the process.

The last step is more complicated. It involves combining the knowledge from different scenarios. For example, by combining the knowledge from Scenario 1 (approaching a target straight ahead) and Scenario 2 (turning to face a target), we would expect a capable system to be able to solve problems that (while very simple) have not been presented in training to the system. The only difficulty here is the translation between differently labeled prototypes for the “same” concept. This turns out to be straightforward from a pragmatic standpoint, though it should be considered an ad hoc solution in general.

6.4.1 Evaluation on Scenario 1 – Approaching to a target

The training data in Scenario 1 are three ways in which Canis approaches a target (yellow ball) – in the center in front of Canis, or to the left or right front of Canis. The outcome is simple when the target is in the center front: Canis will finally contact with the target. Otherwise, Canis will just pass the target (Figure 6.2).

In evaluation, the first test, when a target is in the center, is whether a contact state can be discovered by the planning system. Converting this to the sensorimotor signals in the system, the initial readings are distance<12.5>, direction<0>, and contact<0>. The plan generated by the program is shown in Table 6.6,

Table 6.6 Action plan of approaching to a target

It took 2 steps from distance<12.44, 0> direction<0.0, 0.0> contact<0.0, 0.0> to achieve contact<1.0, 0.0> .
Step 1
from distance<12.44, 0.0> direction<0.0, 0.0> contact<0.0, 0.0>
act forward<4.975, 0.0> to distance<11.7, -0.2>
end with distance<11.7,-0.21696> direction<0.0, 0.0> contact<0.0, 0.0>
Step 2
from distance<11.7,-0.2> direction<0.0, 0.0> contact<0.0, 0.0>
act forward<4.975,0.0> to contact<1.0, 0.0>
end with distance<11.7, -0.2> direction<0.0, 0.0> contact<1.0,0.0>

The generated plan in this table begins with the starting state and the goal state. The symbols given in the plan description are translations from automatically generated prototype names, which we have done to make the plan easier to understand. The format of each prototype description is “signal name<parameter one – signal start, parameter two – signal change>”. For each step, the plan gives the condition in the step, which action schemas it uses, and then the effect of the step. The bold text is simply our effort to document the information we consider relevant at each step. In words, this plan could be described so:

1. Starting from a distance of 12.44 units from the target object, with direction 0 and a zero contact value (contact with the ball), execute the action of moving forward with speed 4.975; the effect will be that distance changes by -0.2 at each time step.
2. In this situation, with the distance changing by -0.2, direction at 0, and so forth, continue to execute forward with speed 4.975; the result will be (non-zero) contact with the ball.

Note that the knowledge acquired in Scenario 1 includes more than just making contact with the target. The program also generates the whole planning space for this scenario, starting with a stationary target in front of Canis. A diagram of this space is shown in Figure 6.16.

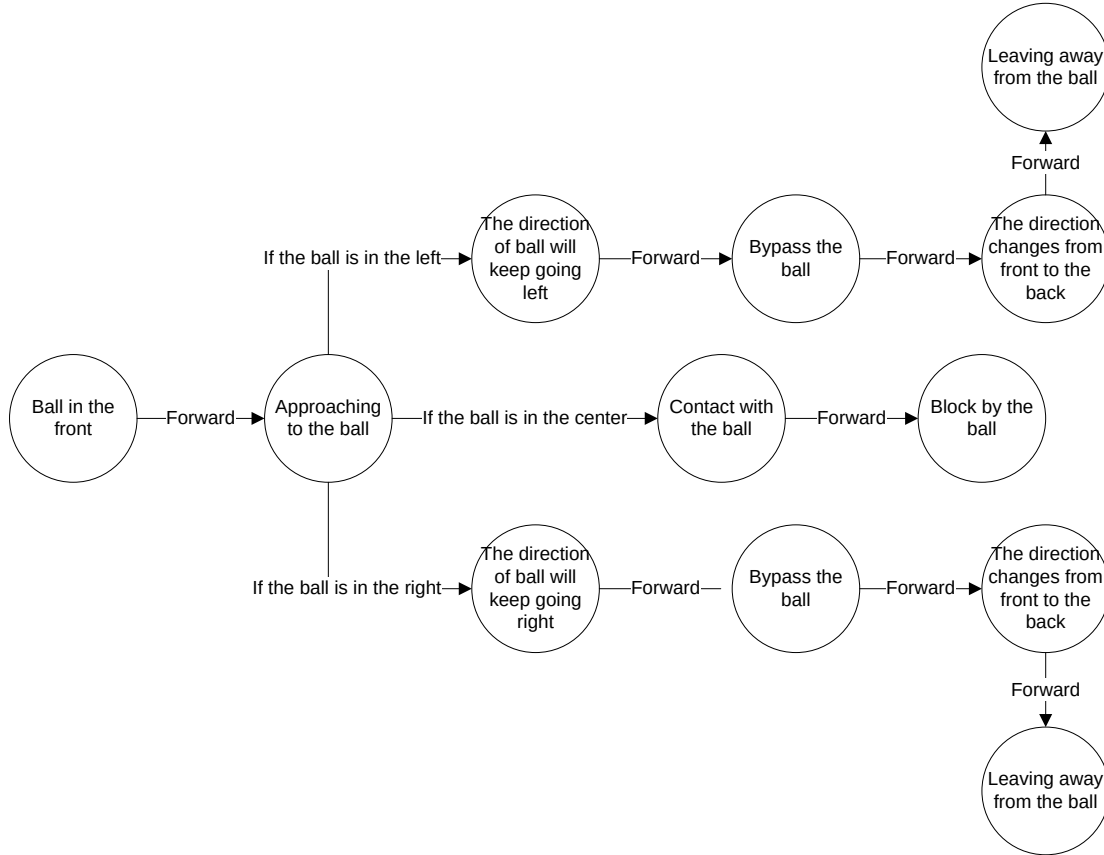


Figure 6.16 Action space of Scenario 1

We also examine the scope of the area which can lead to a successful contact plan, based on variations in the space of initial parameter settings, as shown in Figure 6.17. Here the x-axis represents the direction of the ball and the y-axis represents the distance from the ball; 1 represents a plan found, 0 otherwise.

This result shows the area that, in the training data, generates a successful contact. It is not surprising that when the evaluation program uses a strict match in applying its prototypes, this is the scope of successful plans: essentially, the agent succeeds in situations it has encountered before.

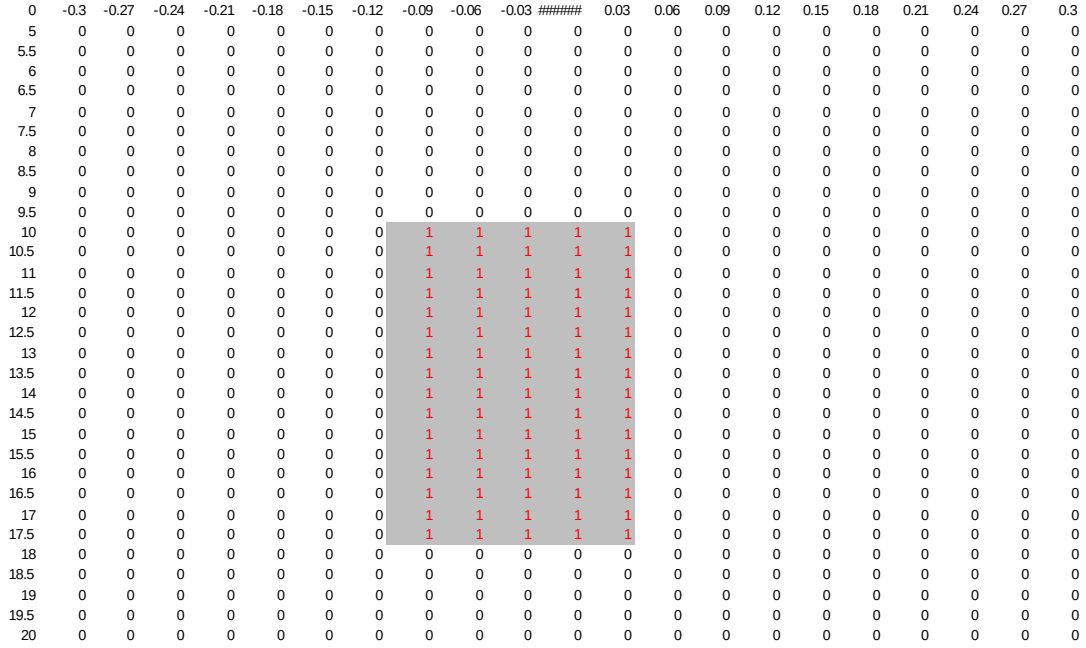


Figure 6.17 Successful areas of making contact

However, if we allow the planning program to use a closest match in applying the prototypes it has learned in training, it will produce appropriate plans in a wider variety of initial parameter settings. Figure 6.18 shows the result, with the blue area representing the additional scope.

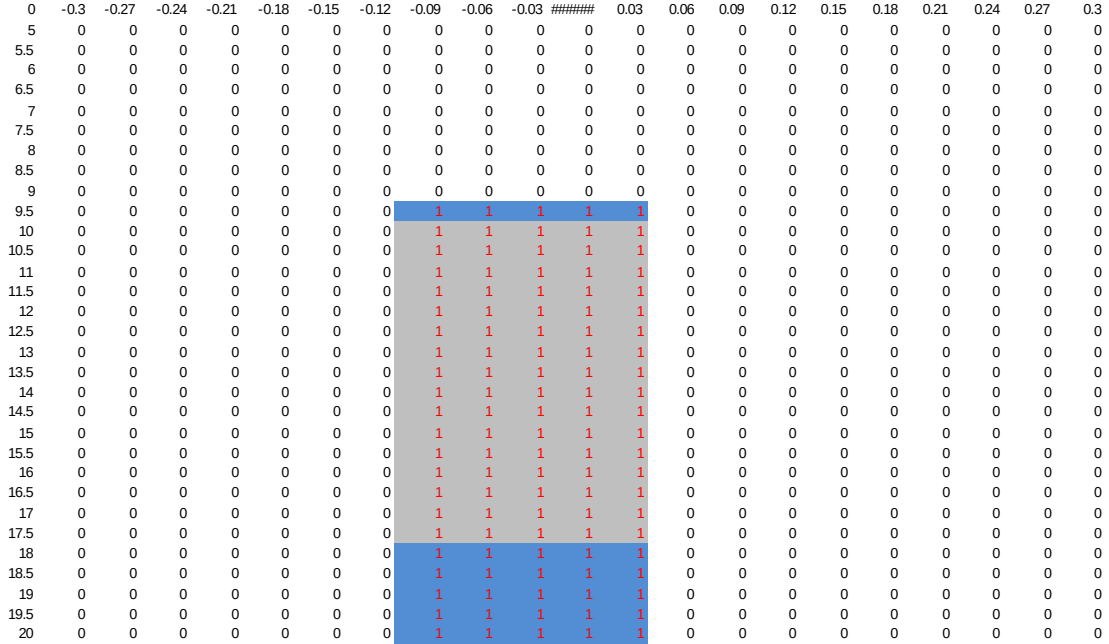


Figure 6.18 Successful areas of making contact with closest match

One more thing we should notice is that the reason that the left and right area can't lead to successful contact is because starting from there, Canis will bypass the ball when moving forward.

6.4.2 Evaluation on Scenario 2 – Turning towards a target

In the second scenario, Canis will turn towards the target from different directions, either left-turn or right-turn, and stop turning when the ball is directly ahead. A successful execution plan can be generated as in Table 6.7,

Table 6.7 Action plan of turning towards a target

It took 4 steps from distance<11,0.0> direction<-0.6,0.0> contact<0.0, 0.0> to achieve distance<10, 0.0> direction<0.0, 0.0> contact<0.0,0.0>.	
Step 1	from distance<11, 0.0> direction<-0.6,0.0> contact<0.0,0.0> act rotation<5.46,0.0> to direction<-0.6,0.01> end with distance<11,0.0> direction<-0.6,0.01> contact<0.0,0.0>
Step 2	from distance<11,0.0> direction<-0.6,0.01> contact<0.0,0.0> act rotation<5.46,0.0> to distance<11,-0.02> end with distance<11,-0.02> direction<-0.6,0.01> contact<0.0,0.0>
Step 3	from distance<11,-0.02> direction<-0.6,0.01> contact<0.0,0.0> act rotation<5.46,0.0> to direction<0.0, 0.0> end with distance<11,-0.02> direction<0.0, 0.0> contact<0.0, 0.0>
Step 4	from distance<11,-0.02> direction<0.0, 0.0>contact<0.0, 0.0> act rotation<0.0,0.0> to distance<10, 0.0> end with distance<10, 0.0> direction<0.0, 0.0> contact<0.0, 0.0>

The plan describes starting with the ball at distance 11, direction -0.6, and contact 0. When Canis executes the action of rotation, with speed 5.46, the direction to the ball changes in the direction of 0; in the meantime, distance is also decreasing slightly (due to the design of Canis's body and the physics of movement in its environment). The direction reaches 0, and when the rotation stops, the distance stops decreasing as well.

In examining the area in which a successful rotation is found, we find that the planning system can generate the plan from almost all directions (Figure 6.19). However, it can't make plans

from a very close distance, because it's so far from the training examples that it maps to other prototypes.

0	-1.5	-1.3	-1.1	-0.9	-0.7	-0.5	-0.3	-0.1	0.1	0.3	0.5	0.7	0.9	1.1	1.3	1.5
10	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
10.5	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
11.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
19.5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 6.19 Successful areas of turning towards the target

6.4.3 Evaluation on Combining the Scenario 1 and 2

It's not surprising that by combining the knowledge from Scenario 1 and Scenario 2, the system can successfully approach the target from a wider area by generating plans that first turn to center on the target and then approach. Table 6.8 demonstrates this.

Table 6.8 Action plan of combining approaching and turning

It took 5 steps from distance<12,0.0> direction<-0.6, 0.0> contact<0,0> to achieve contact<1,0 >.	
Step 1	from distance<12,0> direction<-0.6,0> contact<0,0> act rotation<5.46,0.0> to direction<-0.6,0.01> end with distance<12, 0> direction<-0.6,0.01> contact<0,0>
Step 2	from distance<12, 0> direction<-0.6, 0.01> contact<0,0> act rotation<5.46,0.0> to distance<11,-0.02> end with distance<11,-0.02> direction<-0.6,0.01> contact<0,0>
Step 3	from distance<11,-0.02> direction<-0.6,0.01> contact<0,0> act rotation<5.46,0.0> to direction<0,0> end with distance<11,-0.02> direction<0,0> contact<0,0>
Step 4	from distance<11,-0.02> direction<0,0> contact<0,0> act forward<5,0.0> to distance<11,-0.2> end with distance<11,-0.2> direction<0,0> contact<0,0>
Step 5	from distance<11,-0.2> direction<0,0>> contact<0,0> act forward<5,0.0> to contact<1, 0> end with distance<11,-0.2> direction<0,0> contact<1,0>

The plan starts with distance 12, direction -0.6. In order to make contact with the ball, a rotation with speed 5.46 is carried out, until the direction becomes 0. Then, in the second part of the plan, a forward movement is carried out with speed 5. The plan is generated from the combined knowledge of Scenarios 1 and 2. Therefore, every search step needs to access both knowledge bases. The most challenging part of this planning task is symbol translation—the automatically generated prototype labels have no relationships to each other. (This is a problem that would presumably be handled by a higher-level system that could identify and catalog similarities and identities between the time series variables that generate the prototypes.) Even given this limitation, what is interesting is that the scope of Canis’s performance is not as restrictive as that of its training process.

As a result of combining the two scenarios, when we examine the successful area of plans for reaching a target, there’s a significant extension (compare to the early figures) especially in the

direction signal (Figure 6.20). In this table, the gray area represents the successes only achieved in Scenario 1, and the blue areas are those learned by combining the knowledge from Scenarios 1 and 2. The failed areas are the failed area in Scenario 2.

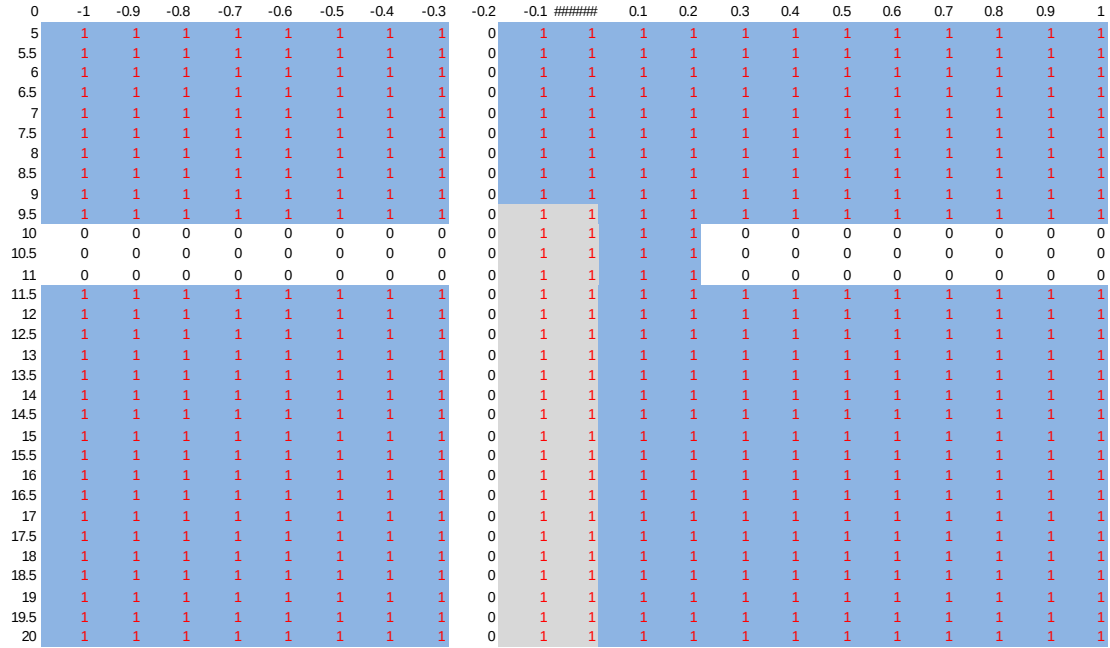


Figure 6.20 Successful areas of combining approaching and turning

To summarize, by examining the capabilities of generating plans from Scenarios 1 and 2, we have shown the system is capable of

1. Learning effective knowledge via the training scenarios. Notice that even though approaching and turning are just simple motions, there's no hand-coded knowledge about them given to the system.
2. Allowing for a simple form of generalization, as in using a closest match for prototypes. The generality with which the system can apply knowledge to new trials depends on how broadly the prototypes can be matched.
3. Accumulating knowledge by combination over different experiences. This is important in that it can help a high-level system pull together disparate experiences dynamically to solve new problems.

The main limitations associated with this system are these:

1. The knowledge of the system gets quite complex and interrelated when every symbol is not only defined by itself (as a prototype definition) but also connects to the other symbols through the rules or actions it is associated with.
2. The capability of combining knowledge from different scenarios is not robust, mainly because the system lacks a high-level mechanism to manage the accumulating memory and concepts.
3. There is no mechanism to allow the system to learn knowledge in the middle of an action. In other words, the current system only attends to the preconditions or effects when an action is started or stopped, rather than intermediate points.

6.4.4 Evaluation on Scenario 3 - Pushing a target off a box

We continue evaluation of Scenarios 3 and 4 in the following sections based on the same methods. In Scenario 3, Canis tries to push a ball (the target) off a box in three different situations (Figure 6.4). One major difference in these three situations is that the size of the box is increasing. Further, Canis has a stick in mouth in the last situation—the use of a tool. The following three figures illustrate the sensory signals in those three setup trials. In order to compare to the data recovered from regression, we also attach recovered data for box distance and ball distance in the same figure to illustrate how close they are.

Among these signals, there are two focus points, the ball and the box. For each object, three signals are computed, relative distance, relative direction and contact indicator. For the ball, the signal is computed from the head of the Canis. For the box, the signals are computed from the body of Canis.

The first figure (Figure 6.21) shows the trial when the box has a small size – the distance to the ball is actually smaller than the distance to the box (because Canis’s head is far in front of its body). The ball is contacted by Canis’s head first, and then the box is contacted by Canis’s body. The ball will eventually move away, since it’s pushed off the box. Canis is blocked by the box in the end.

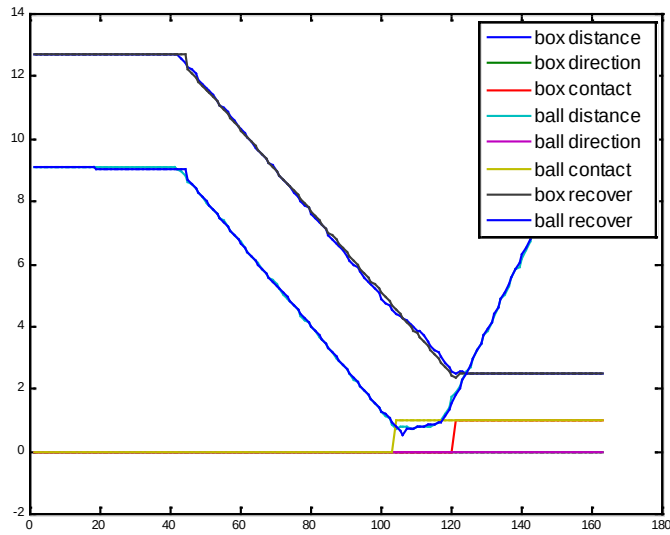


Figure 6.21 Pushing the target on a small box

In the next figure (Figure 6.22), the box is larger. So the distance to the ball becomes larger, since the ball is always placed at the further end of the box. Only the box is contacted by Canis at the end. The ball stays on the top of the box all the time.

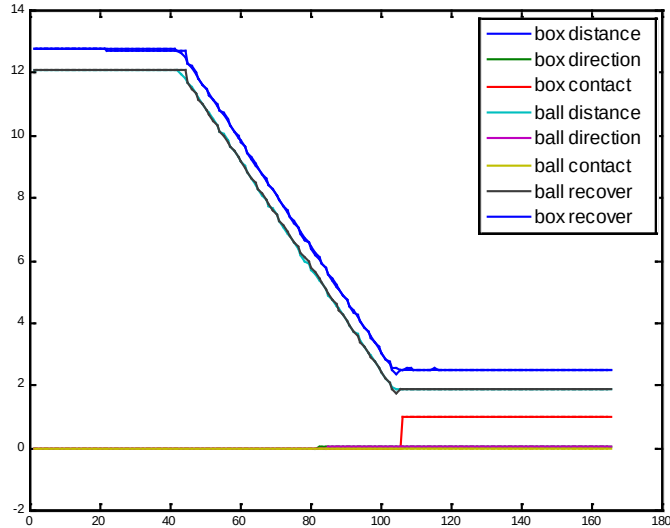


Figure 6.22 Pushing the target on a larger box

In the final figure (Figure 6.23), the size of the box is even larger. So the distance to the ball is also larger (larger than the distance to the box). However, since Canis has a stick in its mouth, it

can reach farther, and it eventually pushes the ball off the box and then contacts the box as in the first setup.

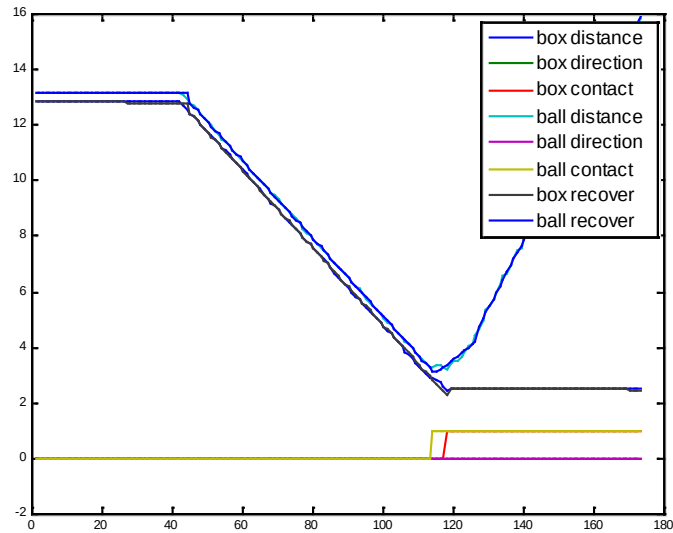


Figure 6.23 Pushing the target with a stick

The evaluation results for time series segmentation and prototype construction can be found in an earlier section. We focus on the system evaluation in this section. In the plan generation evaluation, we are concerned with whether Canis can generate a successful plan to push the ball off the box, with or without a stick. An action plan example is shown in Table 6.9,

Table 6.9 Action plan of pushing a target

It took 2 steps from stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<0,0> ball-distance<9.5,0> ball-direction<0,0> ball-contact<0,0> to stick<0,0> box-distance<12,-0.15> box-direction<0,0> box-contact<0,0> ball-distance<1,0> ball-direction<0,0> ball-contact<1,0>.

Step 1

from stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<0,0> ball-distance<9.5,0> ball-direction<0,0> ball-contact<0,0>

act forward<2.5,0> to box-distance<12,-0.15> ball-distance<9.1,-0.15>

end with stick<0,0> box-distance<12,-0.15> box-direction<0,0> box-contact<0,0> ball-distance<9.1,-0.15> ball-direction<0,0> ball-contact<0,0>

Step 2

from stick<0,0> box-distance<12,-0.15> box-direction<0,0> box-contact<0,0> ball-distance<9.1,-0.15> ball-direction<0,0> ball-contact<0,0>

act forward<2.5,0> to ball-distance<1,0> ball-contact<1,0>

end with stick<0,0> box-distance<12,-0.15> box-direction<0,0> box-contact<0,0> ball-distance<1,0> ball-direction<0,0> ball-contact<1,0>

We examine the entire planning space as well; it can be described as in the following diagram (Figure 6.24),

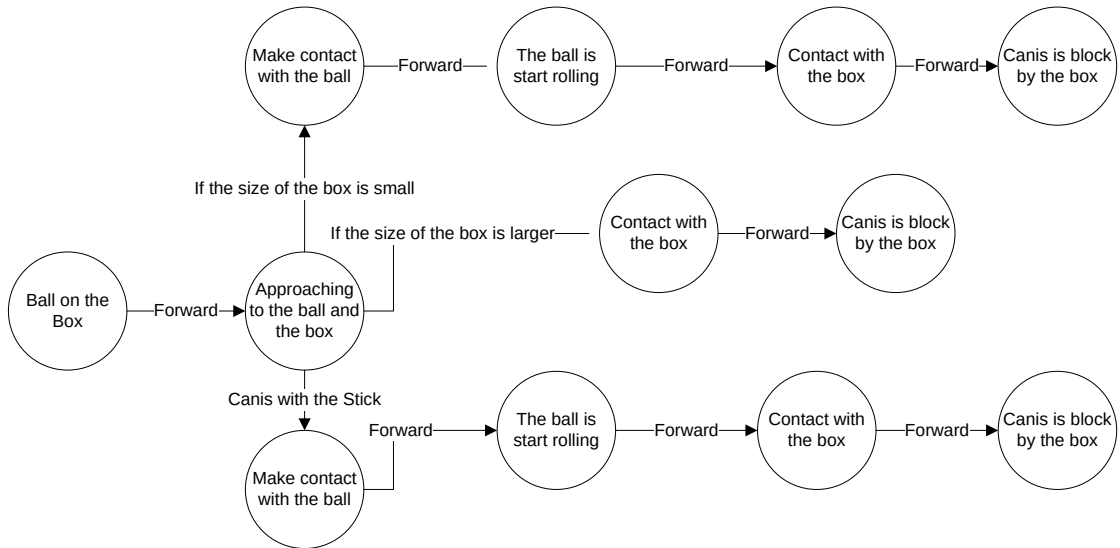


Figure 6.24 Action space of scenario 3

This diagram captures all three situations in this scenario. Notice that the first branch, in which the box is small and Canis has no stick to push with, the states and transitions are very similar to the last branch, with Canis carrying a stick and working with a large box.

We also generated a comparison matrix for successful plans to evaluate the situations the ball can be pushed off the box. In this scenario, we are interested in the relationship between maximal distances to the ball and box in the initial state, as shown in Table 6.10. This is because one implicit precondition of pushing the ball off the box is to have a small relative difference between the distance to the ball and the distance to the box, and the effect of using a stick is to reduce such a distance. Although discovering this implicit condition is beyond the scope of our Action Schema Generator, we still want to understand how the ASG might make the problem solvable. From Table 6.10, one observation is that ASG did capture the maximal distance to the ball for a successful trial (pushing the ball off the box), and the effect of having a stick is also clear – it increases the reaching ability of Canis by 3.5, which is the size of the stick. However, the maximal starting distance to the ball didn't change regard to the distance to the box. In other words, the ASG didn't capture the relative relationships between the distance to the box and the distance to the ball. This is mainly because the prototypes are constructed independent analysis of its sensor signals; identifying structural relationships *between* such signals and thus between prototypes remains for future work.

Table 6.10 Maximal distances in successful plan

Starting Distance	Box	Maximal Starting Distance	Ball	Max Ball Distance with Stick	Starting Distance
12		10		13.5	
12.5		10		13.5	
13		10		13.5	
13.5		10		13.5	

6.4.5 Evaluation on Scenario 4 - Stretch to push a target

In Scenario 4, Canis is trying to push the target (a yellow ball) off the box by stretching its neck, with or without stick. The data files include the same sensorimotor signals as Scenario 3. The signal plots are also very similar to Scenario 3. One major difference is that in this scenario, Canis will first make contact with the box then with the ball. And there's a longer time from the contact of the ball to force the ball off the box (Figure 6.25 and Figure 6.26).

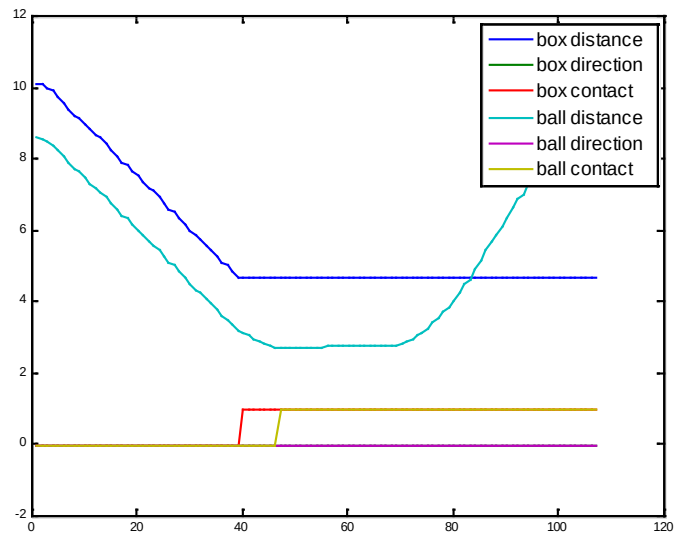


Figure 6.25 Stretch to push the target

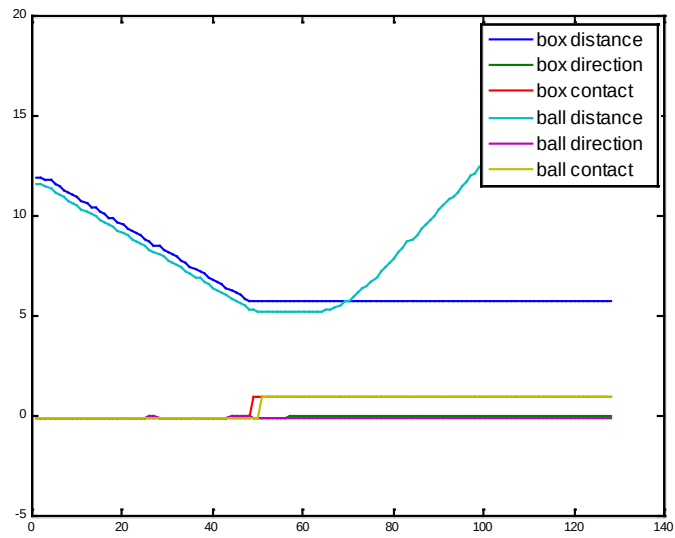


Figure 6.26 Stretch to push the target with stick

The most challenging task is to combine the actions from both Scenarios 3 and 4 to reach the ball a little bit further away (Table 6.11),

Table 6.11 Action plan of combining pushing and stretching

It took 5 steps from stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<0,0> ball-distance<10.5,0> ball-direction<0,0> ball-contact<0,0> to achieve stick<0,0> box-distance<2,0> box-direction<0,0> box-contact<1,0> ball-distance<1,0> ball-direction<0,0> ball-contact<1,0> Step 1 from stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<0,0> ball-distance<10.5,0> ball-direction<0,0> ball-contact<0,0> act forward<2.5,0> to box-distance<12,-0.15> ball-distance<10,-0.15> end with stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<0,0> ball-distance<10,-0.15> ball-direction<0,0> ball-contact<0,0> Step 2 from stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<0,0> ball-distance<10,-0.15> ball-direction<0,0> ball-contact<0,0> act forward<2.5,0> to box-contact<1,0> end with stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<1,0> ball-distance<10,-0.15> ball-direction<0,0> ball-contact<0,0> Step 3 from stick<0,0> box-distance<12,0> box-direction<0,0> box-contact<1,0> ball-distance<10,-0.15> ball-direction<0,0> ball-contact<0,0> act forward<2.5,0> to box-distance<2,0> ball-distance<1.8,0> end with stick<0,0> box-distance<2,0> box-direction<0,0> box-contact<1,0> ball-distance<1.8,0> ball-direction<0,0> ball-contact<0,0> Step 4 from stick<0,0> box-distance<2,0> box-direction<0,0> box-contact<1,0> ball-distance<1.8,0> ball-direction<0,0> ball-contact<0,0> act stretch<0.3,0> to ball-distance<1.8,-0.14> end with stick<0,0> box-distance<2,0> box-direction<0,0> box-contact<1,0> ball-distance<1.8,-0.14> ball-direction<0,0> ball-contact<0,0> Step 5 from stick<0,0> box-distance<2,0> box-direction<0,0> box-contact<1,0> ball-distance<1.8,-0.14> ball-direction<0,0> ball-contact<0,0> act stretch<0.3,0> to ball-distance<1,0> ball-contact<1,0> end with stick<0,0> box-distance<2,0> box-direction<0,0> box-contact<1,0> ball-distance<1,0> ball-direction<0,0> ball-contact<1,0>
--

The first difference in the combination of Scenarios 3 and 4, compared to Scenarios 1 and 2, is that multiple objects are being monitored. Therefore, the system has to process multiple effects on all the objects. The evaluation must be extended in the same way.

The second difference is that tool-using behavior is involved starting in Scenarios 3 and 4. Although the Action Schema Generator is not capable of generating rules that explicitly identify tool use, the actual effect of using a tool is clear in the generated plans. The ASG provides the necessary information, we believe, for a higher level system to learn a tool-using action schema.

One limitation demonstrated in these two scenarios is that the system doesn't capture the inter-relations between multiple objects, in this case that the reaching limit to the ball depends on the size of the box.

6.4.6 Evaluation on Scenario 5 - Detecting a target

In Scenario 5, Canis swings a stick, while leaning downward, to detect whether there's an object (a yellow ball) placed in front of the box on which it is located. There's no direct signal for the target's presence; instead there are only two signals monitoring the angles of Canis's head joint and neck joint. The main goal of this scenario is to test whether the system can distinguish the different states at the end of the simulation instead of the beginning. Implicitly, this scenario is also intended to provide insight into the issue of how a tool-using agent can deal with the transformation or even elimination of specific sensory inputs when a tool is used. For example, if a person touches an object with a finger, the contact registers on the nerve endings of the fingertip; if the person instead touches the object with a stick, the contact (leaving aside visual and auditory indications) must be detected by dynamic constraints on movement of the arm holding the stick.

The sensorimotor signals are relatively simple (Figure 6.27 and Figure 6.28). The upper picture shows the readings when there's a ball in front of the box. The lower one shows no object in front of the box. The only difference is that the reading of Canis's head joint will change at the end of the simulation trial.

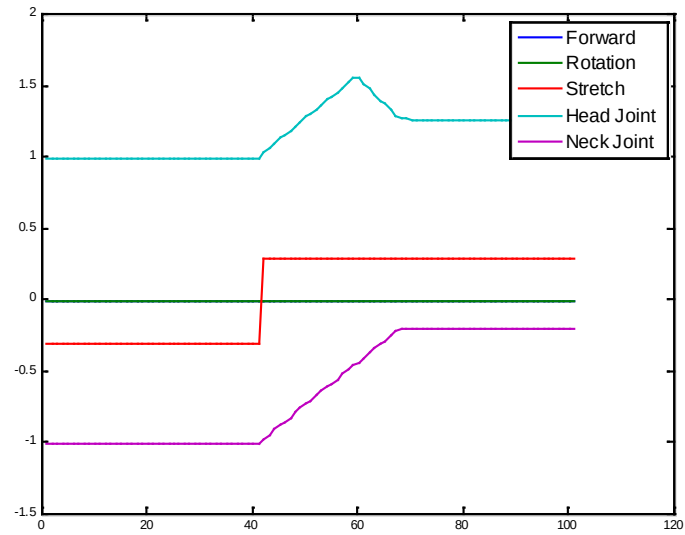


Figure 6.27 Detecting the object in front of the box

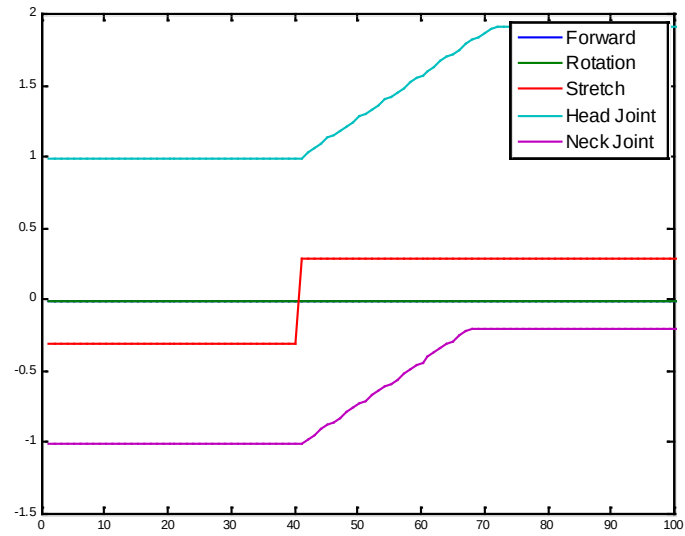


Figure 6.28 Detecting no object in front of the box

The system identified all 20 trials with a ball in front of the box. The state map of the action descriptions is illustrated as in Figure 6.29.

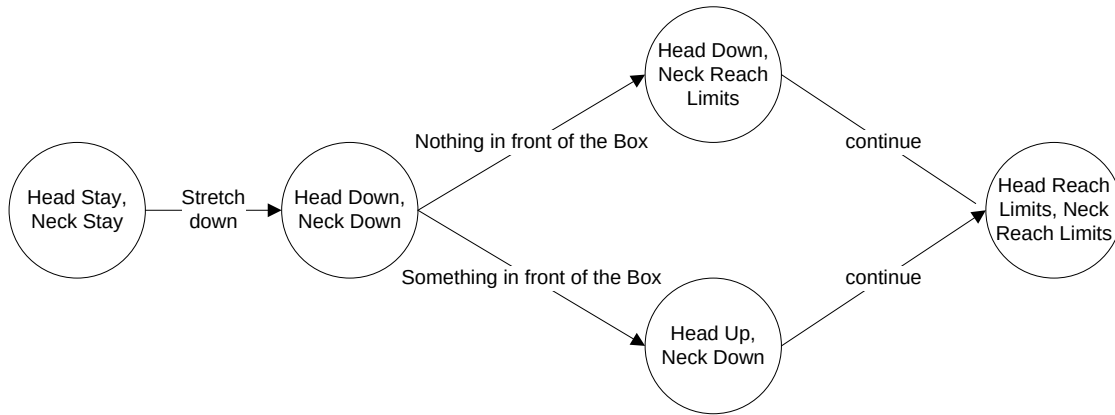


Figure 6.29 Action plan of scenario 5

6.4.7 Evaluation on Scenario 6 – Sweep a target

In scenario 6, Canis is trying to use a stick to move a target (yellow ball) that is placed in a corner. A direct approach, without a tool, will simply force the ball further into the corner, but a sweeping action with a stick tool will move the ball out with some effectiveness. This is a more complicated scenario than earlier scenarios. First, in this scenario, Canis has to carry out two actions, moving forward and rotating, for success. This is the first scenario in which Canis learns more than one action at a time. Second, we vary the scenario in more ways than in other scenarios. Canis may start from either left or right of the target; it may sweep leftward or rightward; and in one situation, Canis can execute only one action instead of two. The main purpose of this scenario is to examine whether our system can learn complicated action executions from one training set.

There are three controller signals and four sensor signals in this scenario. Besides three controllers – “forward”, “rotation”, and “stretch”; the sensor signals are “distance” – the relative distance from Canis to the target, “direction” – the relative direction of the target to Canis in radius, “contact” – which indicates if Canis has made contact with the target yet, and “corner” – the relative distance between the target and the end of the corner. The following figures (Figure 6.30 and Figure 6.31) show the signal plots for two situations. The upper picture shows when Canis starts from the left side of target and then sweep the target off the corner. The lower picture shows when Canis starts from the right side of the target and then sweeps the target towards the corner.

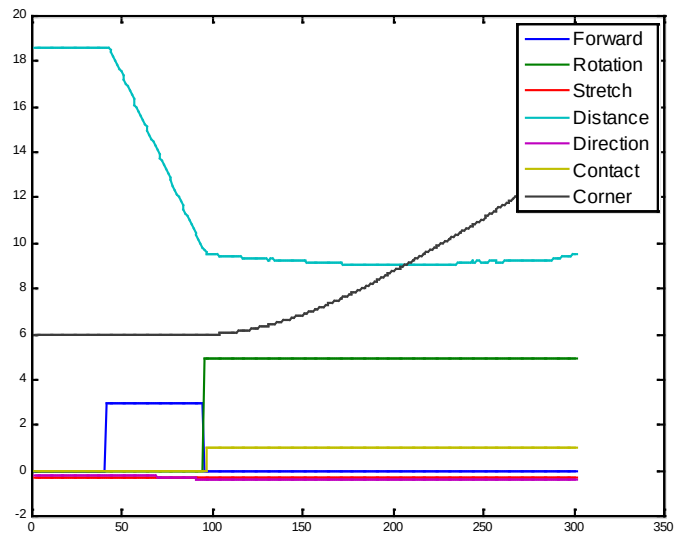


Figure 6.30 Sweeping the target off the corner

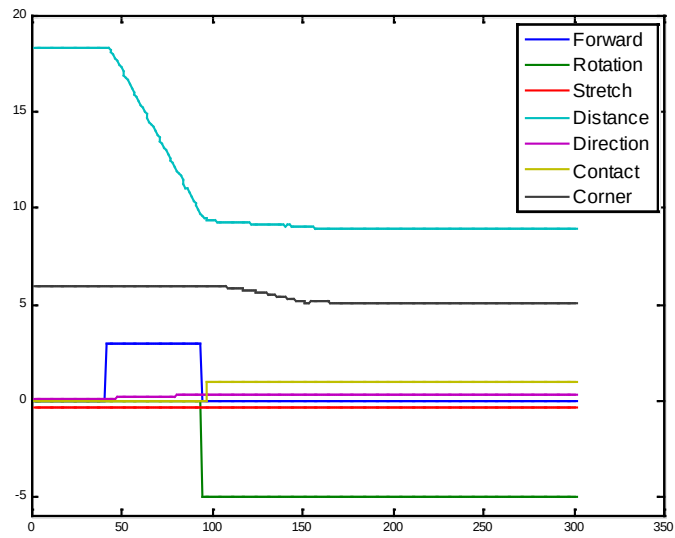


Figure 6.31 Sweeping the target towards the corner

In the planning evaluation, the first task is to find a plan to sweep the target out of the corner. Table 6.12 shows the result.

Table 6.12 Action plan of sweeping a target off the corner

It took 6 steps from distance<18,0> direction<-0.17,0> contact<0,0> corner<6,0> to achieve distance<9,0> direction<-0.3,0> contact<1,0> corner<7,0.04>.	
Step 1	from distance<18,0> direction<-0.17,0> contact<0,0> corner<6,0> act forward<3,0> to distance<18,-0.17> end with distance<18,-0.17> direction<-0.17,0> contact<0,0> corner<6,0>
Step 2	From distance<18,-0.17> direction<-0.17,0> contact<0,0> corner<6,0> act forward<3,0> to direction<-0.25,-0.005> end with distance<18,-0.17> direction<-0.25,-0.005> contact<0,0> corner<6,0>
Step 3	from distance<18,-0.17> direction<-0.25,-0.005> contact<0,0> corner<6,0> act forward<0,0> rotation<5,0> to distance<9,0> end with distance<9,0> direction<-0.25,-0.005> contact<0,0> corner<6,0>
Step 4	from distance<9,0> direction<-0.25,-0.005> contact<0,0> corner<6,0> act forward<0,0> rotation<5,0> to direction<-0.3,0> contact<1,0> end with distance<9,0> direction<-0.3,0> contact<1,0> corner<6,0>
Step 5	from distance<9,0> direction<-0.3,0> contact<1,0> corner<6,0> act forward<0,0> rotation<5,0> to corner<6,0.025> end with distance<9,0> direction<-0.3,0> contact<1,0> corner<6,0.025>
Step 6	from distance<9,0> direction<-0.3,0> contact<1,0> corner<6,0.025> act forward<0,0> rotation<5,0> to corner<7,0.04> end with distance<9,0> direction<-0.3,0> contact<1,0> corner<7,0.04>

In the plan, step 1 and step 2 execute a forward action with speed 3. The distance reading starts dropping as well as the direction reading. These are the results of approaching the target from left side. From step 3, the action plan changes to stop the forward action and start rotation to the right (rotation<5,0>). It results in the distance reading remaining static (step 3 – distance<9,0>), the direction reading stopping at -0.3, and a contact with the target (step 4). From step 5, the target starts moving away from the corner (corner<6,0.025>), in an accelerating speed (corner<7,0.04> in step 6).

The second task is to find the action results if Canis starts from the right side of the target and sweeps the target leftward (see Table 6.13),

Table 6.13 Action plan of sweeping a target towards corner

It took 5 steps from distance<18,0> direction<0.17,0> contact<0,0> corner<6,0> to achieve distance<9,0> direction<0.3,0> contact<1,0> corner<5,0>.	
Step 1	from distance<18,0> direction<0.17,0> contact<0,0> corner<6,0> act forward<3,0> to distance<18,-0.17> end with distance<18,-0.17> direction<0.17,0> contact<0,0> corner<6,0>
Step 2	from distance<18,-0.17> direction<0.17,0> contact<0,0> corner<6,0> act forward<3,0> to direction<0.3,0> end with distance<18,-0.17> direction<0.3,0> contact<0,0> corner<6,0>
Step 3	from distance<18,-0.17> direction<0.3,0.0003> contact<0,0> corner<6,0> act forward<0,0> rotation<-5,0> to distance<9,0> contact<1,0> end with distance<9,0> direction<0.3,0> contact<1,0> corner<6,0>
Step 4	from distance<9,0> direction<0.3,0> contact<1,0> corner<6,0> act forward<0,0> rotation<-5,0> to corner<6,-0.018> end with distance<9,0> direction<0.3,0> contact<1,0> corner<6,-0.018>
Step 5	from distance<9,0> direction<0.3,0> contact<1,0> corner<6,-0.018> act forward<0,0> rotation<-5,0> to corner<5,0> end with distance<9,0> direction<0.3,0> contact<1,0> corner<5,0>

This plan is basically a mirror of the last one. Step 1 and step 2 execute the moving forward action from the right side of the target. Step 3 changes to stop the forward action and start rotation to the left side (rotation<-5,0>). As a result, the target is moving towards the corner in step 4 (corner<6,-0.018>). Finally, the target is blocked at step 5 (corner<5,0>).

We also examine if the system can capture knowledge about the initial states, more specifically, the starting distance and direction of Canis. The following figures (Figure 6.32, Figure 6.33) show the information. Figure 6.32 (distance is on the vertical axis, direction on the horizontal axis) shows where Canis can sweep the target out of the corner (1 represents cases where a success

plan is generated, 0 otherwise). Figure 6.33 shows where Canis sweeps the target towards the corner. Notice this is the opposite plan to the above.

0	-0.25	-0.21	-0.17	-0.13	-0.09	-0.05	-0.01	0.03	0.07	0.11	0.15	0.19	0.23
17.5	0	0	0	0	0	0	0	0	0	0	0	0	0
17.6	0	0	0	0	0	0	0	0	0	0	0	0	0
17.7	0	0	0	0	0	0	0	0	0	0	0	0	0
17.8	0	0	0	0	0	0	0	0	0	0	0	0	0
17.9	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0
18.1	0	0	0	0	0	0	0	0	0	0	0	0	0
18.2	0	1	1	0	0	0	0	0	0	0	0	0	0
18.3	0	1	1	0	0	0	0	0	0	0	0	0	0
18.4	0	1	1	0	0	0	0	0	0	0	0	0	0
18.5	0	1	1	0	0	0	0	0	0	0	0	0	0
18.6	0	1	1	0	0	0	0	0	0	0	0	0	0
18.7	0	1	1	0	0	0	0	0	0	0	0	0	0
18.8	0	1	1	0	0	0	0	0	0	0	0	0	0
18.9	0	1	1	0	0	0	0	0	0	0	0	0	0
19	0	1	1	0	0	0	0	0	0	0	0	0	0
19.1	0	1	1	0	0	0	0	0	0	0	0	0	0
19.2	0	1	1	0	0	0	0	0	0	0	0	0	0
19.3	0	1	1	0	0	0	0	0	0	0	0	0	0
19.4	0	1	1	0	0	0	0	0	0	0	0	0	0
19.5	0	1	1	0	0	0	0	0	0	0	0	0	0
19.6	0	1	1	0	0	0	0	0	0	0	0	0	0
19.7	0	1	1	0	0	0	0	0	0	0	0	0	0
19.8	0	0	0	0	0	0	0	0	0	0	0	0	0
19.9	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0
20.1	0	0	0	0	0	0	0	0	0	0	0	0	0
20.2	0	0	0	0	0	0	0	0	0	0	0	0	0
20.3	0	0	0	0	0	0	0	0	0	0	0	0	0
20.4	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6.32 Successful area of sweeping the ball off the corner

0	-0.25	-0.21	-0.17	-0.13	-0.09	-0.05	-0.01	0.03	0.07	0.11	0.15	0.19	0.23
17.5	0	0	0	0	0	0	0	0	0	0	0	0	0
17.6	0	0	0	0	0	0	0	0	0	0	0	0	0
17.7	0	0	0	0	0	0	0	0	0	0	0	0	0
17.8	0	0	0	0	0	0	0	0	0	0	0	0	0
17.9	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0
18.1	0	0	0	0	0	0	0	0	0	0	0	0	0
18.2	0	0	0	0	0	0	0	0	0	0	1	1	0
18.3	0	0	0	0	0	0	0	0	0	0	1	1	0
18.4	0	0	0	0	0	0	0	0	0	0	1	1	0
18.5	0	0	0	0	0	0	0	0	0	0	1	1	0
18.6	0	0	0	0	0	0	0	0	0	0	1	1	0
18.7	0	0	0	0	0	0	0	0	0	0	1	1	0
18.8	0	0	0	0	0	0	0	0	0	0	1	1	0
18.9	0	0	0	0	0	0	0	0	0	0	1	1	0
19	0	0	0	0	0	0	0	0	0	0	1	1	0
19.1	0	0	0	0	0	0	0	0	0	0	1	1	0
19.2	0	0	0	0	0	0	0	0	0	0	1	1	0
19.3	0	0	0	0	0	0	0	0	0	0	1	1	0
19.4	0	0	0	0	0	0	0	0	0	0	1	1	0
19.5	0	0	0	0	0	0	0	0	0	0	1	1	0
19.6	0	0	0	0	0	0	0	0	0	0	1	1	0
19.7	0	0	0	0	0	0	0	0	0	0	1	1	0
19.8	0	0	0	0	0	0	0	0	0	0	0	0	0
19.9	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6.33 Successful area of sweeping the ball towards the corner

6.5 Summary

In this chapter, we have examined the performance of our system by first designing a series of simulation scenarios, then evaluating the components of the system, and finally evaluating how the whole system can be used in conjunction with a simple planning system. In each test, we have identified the main capabilities and limitations of the system.

As a result of the component-level evaluation, we claim that, first, the time series representation preserves most of the relevant details from the sensorimotor signals in the transformation to a piecewise linear representation; second, the schematic construction process captures more information and forms its prototypes in a plausible way in comparison with a more traditional algorithm.

The final planning evaluation is the measure for the entire system. The system has to capture key concepts, distinguish them from each other, discover relationships among them with respect to execution ordering, and represent them in a form that could be used for those planning tasks. We can claim that our system can do the following:

1. Automatically transfer sensorimotor information from raw numerical signals to a symbolic representation without prior knowledge about the scenarios or the action patterns.
2. Capture key concepts in different situations of training scenarios. Even though the sensorimotor signals vary in number, scope, and type, the symbolic representation generated by the system records enough information for a planning system to make successful reasoning for all test scenarios possible.
3. Learn knowledge from observing the training scenarios. The training scenarios could include multiple objects, multiple actions altogether, and can have multiple outcomes. Again, there's no explicit prior knowledge about the scenarios given to the system.
4. Accumulate knowledge by combining information from multiple scenarios. This is important since it can help a high-level system to reason based on accumulated memory or global knowledge.
5. Support tool-using behaviors, and potentially allow a higher-level system to discover the difference between non-tool-using and tool-using behaviors, thus (again potentially) to discover the effects of using tools.

Of course we also learned about limitations associated with the current system. They are these:

1. The system lacks a mechanism to learn knowledge from the middle of an action. In other words, the current system only deals with the preconditions or effects when an action is started or stopped.
2. The system lacks a mechanism to capture the inter-relations between multiple objects. In other words, in the current system, the form of one object's prototype has no relationships to other objects.
3. There is a lot of redundant information in both the prototype definitions and action descriptions, which increases the complexity of planning and combining knowledge.

Chapter 7

Integration and Comparison

At this point it becomes possible to discuss what the Action Schema Generator enables a system to do, in comparison with existing architectural approaches. Three somewhat overlapping areas for comparison are relevant: work on behavior-based robotics, work in neural network and their evolution, and work on cognitive modeling architectures. In this section we describe similarities and differences to our approach. It is not possible in general to claim that our work enables some capability that is in principle not available in other approaches; rather, the Action Schema Generator attacks a problem that other systems, in general, do not consider.

7.1 Comparison with Behavior-based Robotics

The most distinctive aspect of the behavior-based approach is the direct connection between sensors and effectors in a robot agent. This differs from the more conventional sense-model-plan-act framework, which explicitly pre-stores descriptions of interactions with the world; in the behavior-based approach, an internal symbolic representation of the world is not maintained. In fact, Brooks holds that all autonomous agents need to be situated and embodied, and he argues that embodied robots should “use the world as (their) own best representation”. Ronald Arkin identifies the following as the common features shared by many behavior-based approaches (Arkin, 1998):

- Emphasis on the importance of a tight coupling between sensing and action.
- Avoidance of representational symbolic knowledge.
- Decomposition into contextually meaningful units (behaviors or situation-action pairs).

The main reason that the behavior-based approach avoids building an internal representation is because the uncertainty of the real-world domains is difficult if not impossible to model effectively, and a robot based on such a representation will fail to perform if the environment is changed. However, by avoiding symbolic representations, behavior-based systems generally lack the ability to explain their actions. Some researchers believe that a robot cannot assign meaning to its actions or environment without representing them, even if indirectly (Bruemmer, 2006).

Compared to the approach taken in this research, we also believe that a pre-determined model of the world is too expensive for semantically deep tasks. However, translating sensorimotor experience into a symbolic representation, even via a simple mechanism, provides the possibility for both learning about the world and being able to explain actions. Our approach does recognize the embodiment perspective on a robotic agent's own sensorimotor experience. We believe by learning from those experiences, the agent can acquire its own unique model of its interactions with the world. This knowledge is not pre-programmed, which avoids expensive programming. Further, there is a sense in which the knowledge acquired by the agent is "meaningful" to itself: it can reason about it (in its planning process) and in principle can examine it (by analyzing the components of action schemas). By generating such a symbolic representation, our approach also provides support for symbolic reasoning techniques widely used in AI.

The well-known subsumption architecture, another traditional feature of behavior-based systems, has been mentioned in the introduction to this dissertation. High-level functionalities like "explore" or "go to goal" are built from lower-level behaviors and may function only when lower-level requirements are satisfied. Different layers function simultaneously and independently of each other to reduce complexity. The challenges for such an architecture are how to decompose the problem into behavior layers and how to combine behaviors such that appropriate actions are selected. In our approach, we decompose system functionalities following simple guidelines from work on human cognition: for example, a particular approach to concept representation based on the characteristics of prototypes. We agree that behaviors should be accumulated from low levels to higher levels, as our system supports. However, in our approach, knowledge about layered behaviors is structured in high-level memory management modules, rather than hard coded in the architecture of the system. The benefits of our approach are mainly in support for a central control unit in a system, which can deliberate over the behaviors of lower modules and control the general behaviors of the entire system through some decision-making process.

Similar to other approaches, behavior-based robotics uses biology as the best model for understanding intelligence. However, most roboticists in this area focus on modeling the adaptive behaviors of lower-level animals like insects, rather than modeling the internal cognition process of the higher-level animals like primates or even human beings. As a difference, our approach is based, informally, on a theory of human cognition and studies of higher-level animal tool-using behaviors. Human-level cognitive processing remains a distant goal, of course.

To sum up, we find the following commonality between our approach and the behavior-based approach,

- Pre-determined models or knowledge are too expensive for semantic deep tasks.
- Behaviors develop from lower levels to higher levels.
- Biology is the best model for understanding intelligence.

However, differences between our approach and behavior-based approach are present as well.

- We take the bodily experience to form the symbolic representation without pre-determined patterns, while behavior-based approaches generally avoid symbolic representation and use interactions with the world as a means of evaluation.
- We carefully select training tasks to allow our agent to learn the knowledge about its behaviors from simple to complicated; this allows a central system to deliberate and plan over behaviors. The behavior-based approach encodes behaviors as layers in the system architecture and avoids a central deliberative control system.
- We study theories of higher-level animals and try to model some of the internal cognition processes, while the behavior-based approach focuses on the behaviors of lower-level animals.

7.2 Comparison with Neural Networks and Neuroevolution

Inspired by biological neural networks, artificial neural networks are abstract models of neurons that are connected to each other and function in a massively parallel fashion. Artificial neural networks have a long history in various applications such as system control, robotics, pattern recognition and vision. However, complicated tasks with many inputs and outputs dramatically increase the complexity of a neural network. The so-called “curse of dimensionality” has historically been a significant obstacle in machine learning problems. Traditional methods tend to exhibit the following problems (Siebel et al., 2007):

- Designing the network structure can be difficult or even infeasible for complicated tasks.
- Determining network parameters by optimization is impractical for large problems since algorithms tend to get stuck in local minima.

Neuroevolution focuses on methods for evolving artificial neural networks with genetic algorithms. In contrast to traditional methods, neuroevolution learns to optimize both the topology and the parameters of artificial neural networks without being given information about the nature of the problem. According to Stanley and Miikkulainen, this is made possible by the method of historical marking, which supports comparison of neural networks with differing topologies and allows evolution of networks through genetic algorithms (Stanley and Miikkulainen 2002).

Two popular and recent evolutionary methods include Neuroevolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) and Evolutionary Acquisition of Neural Topologies (EANT/EANT2) (Siebel et al., 2007).

Starting from a simple feed-forward network of only input and output neurons, NEAT begins the evolution process by either adding a neuron along an existing connection, or by adding a new connection between previously unconnected neurons. Unlike earlier evolutionary approaches, NEAT uses not only mutation but also crossover to produce valid offspring from a given neural network by first aligning similar or equal sub-networks, and then exchanging differing parts. The NEAT approach tracks genes with historical markings, protects innovation via speciation, and builds topologies incrementally from an initial structure.

EANT2 also starts with minimal structures and gain complexity along the evolution path. EANT2 uses a compact genetic encoding that handles both direct and indirect encoding of neural networks. It also applies CMA-ES (“Covariance Matrix Adaptation Evolution Strategy”) (Hansen and Ostermeier, 2001) for parameter optimization, to avoid problems like premature convergence. According to Siebel et al., one main difference between EANT2 and other methods is “the clear separation of structural exploration and structural exploitation, which means EANT2 will try to make sure a new structural element is tested as much as possible before a decision is made to discard it or keep it” (Siebel et al., 2007).

Although the methods of Artificial Neural Networks or Neuroevolution have many advantages, such as robustness against noisy and incomplete input data, they take a very different route to designing intelligent agents. The most notable difference is that methods in Artificial Neural Networks focus on connecting input signals with output controls without minimal

interpretation or extra information. They are typically “black box” systems, which only have one way in and one way out. Compared with our approach, a generic neural network approach will have two major limitations because of this.

First, connecting inputs and outputs without interpretation limits the explanatory or justification capability of the system. Furthermore, it limits the capability of applying further processes to the outputs of the neural networks. For example, methods in neural networks are usually trained for specific tasks. Many of those problems may have overlap in either inputs or outputs. When a complicated job requires the knowledge from multiple tasks, a system based on neural networks may find it difficult to combine the knowledge learned in different tasks, because of this black box characteristic. In other words, divide and conquer will be problematic for a system based on neural networks unless it’s coded by the designer. Our approach focuses on providing a symbolic representation for a higher-level system, which allows the outcomes to be explained and justified. We also demonstrate that a central system could accumulate the knowledge learned in our approach to solve more complicated tasks. This is not a hard limitation on neural network systems, but their complexity has proved difficult to manage in practice.

Pfeifer and Bongard argue that a system based on neural networks will exhibit the following problems comparing with an embodied system (Pfeifer and Bongard, 2007); we take these as distinctions with our approach as well.

- Artificial neural networks, like their biological counterparts, are parallel systems. Even for today’s powerful microprocessors, however, parallelism is still a simulated process. Computational speed is still a bottleneck especially for robots that have to behave in the real world.
- Solving computational problems of neural networks would not solve the issue of designing neural networks for robots, because the networks must always be developed together with robot’s morphology.
- In the development process for a neural network, the network cannot be tested for its fitness in the abstract; it must be embedded in a robot. But in almost all evolutionary algorithms, there is no interaction with the environment during development: the weights as encoded in the genome are directly used on the robot.

7.3 Comparison with ACT-R and Soar

A different approach to modeling the capabilities of the brain starts at a much higher level of abstraction; rather than focusing on brain mechanisms, research on cognitive architectures takes a top-down perspective on processing. Cognitive architectures form a subset of general agent architectures. Their goal is to design computational processes that act like a human being or act intelligently under some definition. Cognitive architectures can take different forms, but the architectures in widest use in cognitive science, Soar and ACT-R, are based on symbolic rule processing.

Soar, created by Laird, Newell, & Rosenbloom (Laird et al, 1987), attempts to unify a range of cognitive phenomena with a single set of mechanisms, and addresses a number of significant methodological and theoretical issues common to all computational cognitive theories (Lewis, 2001). Soar represents the world as a large problem space with states and goals. It attempts to use its knowledge about its current state to apply operators to change the state in an effort to reach its goal. Soar has a working memory, which contains the information about its current state. It also has long-term knowledge base, which is encoded as production rules (also called operators in some of the literature). Each production rule is a condition-action pair. All the production rules in a Soar agent are continuously matched against the declarative working memory. When matching productions are selected, their contents (actions) are eventually applied to working memory. Soar is a goal-directed approach, which means that the agent will choose rules that lead it closer to reaching its current goal.

Similar in some ways to Soar, ACT-R is also a cognitive architecture inspired by Allen Newell and mainly developed by John Anderson. The architecture of ACT-R consist of several modules: a visual module for identifying objects in the visual field, a manual module for controlling the hands and fingers, a declarative module for retrieving information from memory, and a goal module for keeping track of current goals and intentions; each of these modules processes a different kind of information (Anderson et al., 2004). The theory of ACT-R doesn't restrict the number of modules in the architecture; information is exchanged via productions that make changes to buffers associated with modules.

There are a variety of differences between the ACT-R and Soar architectures related to their representation of knowledge and the means by which productions are selected and executed (Muller et al., 2008; Jones et al., 2007). From a very high-level perspective, however, the architectures can be treated as being largely similar. The clearest difference with respect to our approach is how

production rules are generated in Soar or ACT-R. Although Soar and ACT-R design the modules to store, select and execute production rules based on what is known about human cognition, production rules are generally hand-coded by designers. While productions can be learned, the architectures focus mainly on symbolic-level learning rather than the conversion of numerical sensory information to symbols. Our approach focuses on learning production rules automatically from such data.

That is, compared with the theories of Soar and ACT-R, our approach focuses on a lower level of human cognition, the level at which symbolic information and rules are generated and learned. There's no hard conflict between applying our approach and the approach based on Soar or ACT-R. In fact, there's a strong possibility that further research can combine the two approaches.

Chapter 8

Conclusion

We have presented a computational system called an Action Schema Generator, which can automatically discover useful knowledge from a (simulated or physical) agent's sensorimotor experience of interacting with the world, in order to support higher-level reasoning. The design of the system relies on a perspective associated with embodiment theory; in other words, the fundamental assumption of this work is that knowledge about the world should be acquired from the robotic agent's experience, not provided by the designer. We argue that the results of the agent's processing are then meaningful to the agent itself, and that this meaning is derived from the agent's embodiment and its interactions with the world. Compared with classical approaches, the contribution of this research is shifted from directly engineering the knowledge about the world into the system to the design of a general architecture that can discover knowledge through the agent's embodied experience.

In order to achieve the above goal in our research, the system solves three major challenges. First, it breaks down its continuous sensorimotor data into a discrete format, to make distinctions about when the inputs are quantitatively different but qualitatively similar. Second, it identifies frequently appearing pieces of information and organizes them appropriately in order to make learning or other high-level reasoning possible. Third, the system addresses the issue of capturing the meaning of its embodied experiences by learning connections between these generated structures for the purpose of building actions, which can be selected by a higher-level reasoning system.

We have also focused on how to apply such a system to tool-using tasks for an agent. Tool-using behaviors have been widely studied not only with humans but also with other animal species. We tested our system on both non-tool-using tasks and tool-using tasks. We find no hard boundaries between in the representations that the agent derives for the tool use and non-tool use cases, which is consistent with conventional views of tool use in human cognitive research. We also found two different types of effects in tool-using tasks, which may help in designing a higher-level reasoning system for tool-using behaviors.

Evaluating our research is complicated, especially since there are few standard procedures for evaluation, either with respect to tool-using agents or even more generally with respect to cognitive robotic agents. We developed six different simulation scenarios to evaluate different aspects of our system’s performance, as well as the performance of the system components. Based on the evaluation, we believe that the main contributions of the work are in these capabilities of the system:

1. To automatically translate sensorimotor information from numerical raw signals to a symbolic representation without prior knowledge about the scenarios or the action patterns and to encapsulate this information in the form of simple concept prototypes.
2. To generate action schemas based on these prototypes that support successful behavior across a range of tasks that include dealing with multiple objects, executing multiple actions, pursuing multiple outcomes, and using tools.

A third contribution is in the mapping from cognitive considerations to an architecture component that we have demonstrated has some generality; one insight we have gained into how tool use can be conceptualized involves a distinction between tools that amplify behaviors and tools that provide for novel behaviors.

We end with a brief discussion of future work. When we evaluated our system, we designed most of the scenarios in Breve, a 3D simulation environment. We evaluated some aspects of the system’s performance with a real physical robot, the Sony Aibo ERS-7, but this testing was far from complete. Part of the reason is the well-recognized complexity of dealing with the interactions between real physical robots and real physical environments. On obvious future direction for this research is deployment and testing with a physical robot, and evaluation of performance in real environments that mimic the simulations we have constructed.

During the stage of building action schemas, we noticed that if there are too many sensorimotor signals monitored by the agent, differential analysis alone was not able to distinguish

irrelevant signals from conditional signals. This is a standard problem in statistics, in particular in the area of causal modeling. Another direction for future research is to test the extent of action schemas as representations of an environment. Can an agent develop strategies for building and refining action schemas that are more than sufficient? What are the boundaries on such a representation?

Finally, in the comparison with other cognitive architectures, we argued that the main focus of our system is to bridge the gap between the perception layer and the higher-level reasoning layer of a robotic agent. Cognitive architectures like Soar or ACT-R could in principle make use of the knowledge generated by our system. There's a potential opportunity to combine the advantages of both systems. A high-level cognitive architecture like Soar or ACT-R could plausibly provide high-level cognitive processing, based on a set of knowledge structures automatically generated by the Action Schema Generator. This is possibly the most promising direction for our future work.

Bibliography

Acosta-Calderon, C. A. and Hu, H. (2005). *Robot imitation: Body schema and body percept*. Applied Bionics and Biomechanics, 2 (3 & 4), 131-148.

Anderson, J. R.; Bothell, D. and Byrne, M. D. (2004). *An Integrated Theory of the Mind*. Psychological Review, 111 (4), 1036-1060.

Arkin, R. C. (1998). *Behavior-Based Robotics*. The MIT Press.

Beck, B. B. (1981). *Animal Tool Behavior: the Use and Manufacture of Tools*. The Quarterly Review of Biology, 56 (2), 231.

Bertel, S.; Barkowsky, T.; Engel, D. and Freksa, C. (2006). *Computational modeling of reasoning with mental images: basic requirements*. Proceedings of the 7th International Conference on Cognitive Modeling, 50-55. Trieste.

Bicici, E. and St.Amant, Robert. (2003). *Reasoning About the Functionality of Tools and Physical Artifacts*. North Carolina State University.

Boucard, P. (2005). *Body Schema and Robotics*. Department of Information Technology, University of Zurich.

Breuer, T.; Ndoundou-Hockemba, M. and Fishlock, V. (2005). *First Observation of Tool Use in Wild Gorillas*. PLoS Biology, 3 (11), 2041-2043.

Brooks, R. A. (1999). *Cambrian Intelligence: The Early History of the New AI*. Cambridge, MA: MIT Press.

Bruemmer, D.(2006). *Behavior-Based Robotics*. <http://www.inl.gov/adaptiverobotics/behaviorbasedrobotics/index.shtml>.

Chapman, D. (1991). *Vision, Instruction, and Action*. Cambridge, Mass.: MIT Press.

Clark, A. and Grush, R. (1999). *Towards a Cognitive Robotics*. *Adaptive Behavior*, 7 (1), 5-16.

Cohen, P. R. and Beal, C. R.(2000). *Natural Semantics for a Mobile Robot*. University of Massachusetts, Department of Computer Science.

Cowart, M.(2006). *Embodied Cognition*. <http://www.iep.utm.edu/e/embodcog.htm>.

Everitt, B. S. (1995). *Cluster Analysis*. London: Arnold.

Fodor, J. A. (1998). *Concepts: Where Cognitive Science Went Wrong*. Oxford: Clarendon Press.

Gallese, V. and Lakoff, G. (2005). *The Brain's Concepts: The Role of the Sensory-Motor System in Reason and Language*. *Cognitive Neuropsychology*, 22, 455-479.

Hansen, N. and Ostermeier, A. (2001). *Completely derandomized self-adaptation in evolution strategies*. *Evolutionary Computation*, 9 (2), 159-195.

Hegarty, M. (2004). *Mechanical reasoning by mental simulation*. *Trends in Cognitive Sciences*, 8 (6), 280-285.

Johnson, M. (1987). *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. Chicago: University of Chicago Press.

Johnson, M. and Rohrer, T. (2007). *We Are Live Creatures: Embodiment, American Pragmatism, and the Cognitive Organism*. In *Body, Language and Mind Volume 1: Embodiment* (Vol.1, pp. 17-54). Berlin: Mouton de Gruyter.

Jones, R. M.; Lebiere, C. and Crossman, J. A.(2007). *Comparing Modeling Idioms in ACT-R and Soar*. In proceedings of the 8th International Conference on Cognitive Modeling, Ann Arbor, Michigan.

Keogh, E.; Chu, S.; Hart, D. and Pazzani, M.(2003). *Segmenting Time Series: A Survey and Novel Approach*. In an Edited Volume Data mining in Time Series Databases, 1-22.

Klein, J.(2002). *BREVE: a 3D Simulation Environment for the Simulation of Decentralized Systems and Artificial Life*. Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems.

Kosslyn, S. M. (1999). *Image and Brain: The Resolution of the Imagery Debate*. Cambridge, MA: MIT Press.

Kripke, S. A. (1980). *Naming and Necessity*. Cambridge, Mass: Harvard Univ. Press.

Kruetzen, M.; Mann, J.; Heithaus, M. R.; Connor, R. C.; Bejder, L. and Sherwin, W. B. (2005). *Cultural transmission of tool use in bottlenose dolphins*. Proceedings of the National Academy of Sciences (102), 8939-8943.

Laird, J. E.; Newell, A. and Rosenbloom, P. S. (1987). *SOAR: An Architecture for General Intelligence*. Artificial Intelligence, 33 (1), 1-64.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. Chicago: University of Chicago Press.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. Chicago: University of Chicago Press.

Lakoff, G. and Johnson, M. (1999). *Philosophy in the Flesh: The Embodied Mind and Its Challenge to Western Thought*. New York: Basic Books.

Laurence, S. and Margolis, E. (2003). *Concepts*. In The Blackwell guide to philosophy of mind (Vol.10, pp. 190-213). Malden, Mass: Blackwell.

Lewis, J. A. and Luger, G. F.(2000). *A Constructivist Model of Robot Perception and Performance*. Proceedings of the 22nd Annual Conference of the Cognitive Science Society, 788-793. Philadelphia, PA.

Lewis, R. L. (2001). *Cognitive Theory, Soar*. In International Encyclopedia of the Social & Behavioral Sciences (pp.). Amsterdam: Elsevier Science.

Minnen, D.; Starner, T.; Essa, I. and Isbell, C.(2007). *Improving Activity Discovery with Automatic Neighborhood Estimation*. International Joint Conference on Artificial Intelligence, Hyderabad, India.

Mitchell, M. (1993). *Analogy-Making as Perception: A Computer Model*. Cambridge, Mass: MIT Press.

Moreno, R. A. and de Miguel, A. S.(2006). *A Machine Consciousness Approach to Autonomous Mobile Robotics*. In 5th International Cognitive Robotics Workshop. AAAI-06., 111-118. Boston, MA.

Muller, T. J.; Heuvelink, A. and Both, F.(2008). *Implementing a Cognitive Model in Soar and ACT-R: A Comparison*. Sixth International Workshop "From Agent Theory to Agent Implementation", Estoril, Portugal.

Nabeshima, C.; Lungarella, M. and Kuniyoshi, Y.(2005). *Timing-Based Model of Body Schema Adaptation and its Role in Perception and Tool Use: A Robot Case Study*. Proceedings of 2005 4th IEEE International Conference on Development and Learning, 7-12.

Nilsson, N. J. (1995). *Eye on the Prize*. AI magazine, 16 (2), 9-17.

Oakley, T. (2007). *Image Schemas*. In D. Geeraerts and H. Cuyckens (Ed.), The Oxford Handbook of Cognitive Linguistics (pp. 214-235). New York, NY: Oxford University Press.

Oates, T.; Schmill, M. D. and Cohen, P. R.(2000). *A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgments*. Proceedings of the Seventeenth National

Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, 846-851.

Oswalt, W. H. (1973). *Habitat and Technology: the Evolution of Hunting*. New York [u.a.]: Holt, Rinehart and Winston.

Pfeifer, R. and Bongard, J. (2007). *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, MA: MIT Press.

Pinker, S. (1999). *How the Mind Works*. New York: W.W. Norton & Company.

Pylyshyn, Z. (2002). *Mental imagery: In search of a theory*. Behavioral and Brain Sciences, 25, 157-238.

Pylyshyn, Z. (2003). *Return of the mental image: are there really pictures in the brain?* Trends in Cognitive Sciences, 7 (3), 113-118.

Rakova, M. (2002). *The philosophy of embodied realism: A high price to pay?* Cognitive Linguistics, 13 (3), 215-244.

Regier, T. (1996). *The Human Semantic Potential: Spatial Language and Constrained Connectionism*. Cambridge, MA: MIT Press.

Rohrer, T. (2008). *The Body in Space: Dimensions of embodiment*. In R. M. Frank (Ed.), *Body, Language, and Mind: Volume 2: Sociocultural Situatedness* (Vol.2, Berlin: Mouton de Gruyter.

Rosch, E. and Lloyd, B. L. (1978). *Cognition and Categorization*. Hillsdale, NJ: Lawrence Erlbaum.

Rosch, E. and Mervis, C. B. (1975). *Family Resemblances: Studies in the Internal Structure of Categories*. Cognitive Psychology, 7 (4), 573-605.

Rosenstein, M. T. and Cohen, P. R.(1999). *Continuous categories for a mobile robot*. Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence, 634-640. Orlando, Florida.

Sankoff, D. and Kruskal, J. (2001). *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Stanford, Calif: CSLI.

Siebel, N. T.; Krause, J. and Sommer, G.(2007). *Efficient Learning of Neural Networks with Evolutionary Algorithms*. 29th DAGM Symposium for Pattern Recognition, 466-475. Heidelberg, Germany.

Smith, E. E. and Medin, D. L. (1981). *Categories and Concepts*. Cambridge, Mass: Harvard University Press.

St. Amant, R. and Wood, A. B. (2005). *Tool Use for Autonomous Agents*. Proceedings of the Twentieth National Conference on Artificial Intelligence, 184 - 189. Pittsburgh, PA.

Stanley, K. O. and Miikkulainen, R. (2002). *Evolving Neural Networks through Augmenting Topologies*. Evolutionary Computation, 10 (2), 99-127.

Stoytchev, A. (2003). *Computational Model for an Extendable Robot Body Schema*. Georgia Institute of Technology.

Tan, P.; Steinbach, M. and Kumar, V. (2006). *Introduction to Data Mining*. Boston, Mass: Addison-Wesley.

Thagard, P. (2005). *Mind: Introduction to Cognitive Science*. Cambridge, MA: MIT Press.

van Schaik, C. P.; Ancrenaz, M.; Borgen, G.; Galdikas, B.; Knott, C. D.; Singleton, I.; Suzuki, A.; Utami, S. S. and Merrill, M. (2003). *Orangutan Cultures and the Evolution of Material Culture*. Science, 299, 102-105.

Whiten, A.; Goodall, J.; McGrew, W. C.; Nishida, T.; Reynolds, V.; Suigiyama, Y.; Tutin, C. E. G.; Wrangham, R. and Boesch, C. (1999). *Cultures in chimpanzees*. Letters to Nature, Nature (399), 682-685.

Appendix

In Action Schema Generator, we use a text-based script file to record all the configuration information for each scenario. For example, the following is an example file of scenario 3's script.

```
# This is the script file for scenario3
# ver 1.0

# scenario settings
scenario_name      scenario3

# raw file settings
rawfile_dir       Raw
rawfile_base      rawfile
rawfile_ext       .txt
rawfile_cnt       100

# prim file settings
primfile_dir      Prim
primfile_base     primitve
primfile_ext      .txt
primfile_cnt      100

# schematic file setting
scheffile_dir     Sche
scheffile_prot    proto
scheffile_base    schfile
scheffile_ext     .txt
scheffile_prcnt   10
scheffile_sccnt   100

# rule file setting
rulefile_dir      Rule
rulefile_base     result
```

```

rulefile_ext      .txt
rulefile_cnt      1

# data setting
leastsample       7
minimalvar        0.02

# controller setting
controller_count  3

controller        forward 0 0
controller        rotation 0 0
controller        stretch 0 0

# sensor setting
sensor_count      7

sensor            stick    0 0
sensor            box-distance 0.5 0
sensor            box-direction 0.1 0
sensor            box-contact 0 0

sensor            ball-distance 0.5 0
sensor            ball-direction 0.1 0
sensor            ball-contact 0 0

```

The first part of the script file focuses on the file setting of the scenario data, like the directory of the data file, the filename, extend name, and how many files. The second part of the script file records the minimal requirements for time series segmentation, like the minimal size of a segment, and the minimal standard deviation of the signals.

The third part of the script file provides information about all controller and sensor signals of the simulation agent. For example, in scenario 3, there are three controllers and seven sensors (the signal about stick tool is also count as sensor signal). For each signal, the file records the sensor name following by the estimated standard deviation.