A FRAMEWORK FOR LEARNING FROM DEMONSTRATION, GENERALIZATION AND PRACTICE IN HUMAN-ROBOT DOMAINS

by

Monica Nicolette Nicolescu

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2003

Dedication

To Mircea, my husband,

with all my heart.

Acknowledgments

While writing my Ph.D. dissertation I have been extremely fortunate to be surrounded by wonderful people, whose very special contribution to this dissertation I would like to acknowledge.

First, I would like to express my deepest gratitude to my thesis advisor, Maja Matarić, for her extraordinary support and guidance during my graduate years at USC. I cannot thank her enough for the many things I have learned and for everything that she helped me with during this time. I would like to thank her for introducing me to robotics, for opening my eyes to this wonderful world, and for her energy, enthusiasm and dedication which were a constant source of inspiration. For these, and for many other reasons I will be always indebted to her.

I would also like to express my heartfelt gratitude to the other members of my Qualifying and Dissertation Defense Committee: Gaurav Sukhatme, Margaret McLaughlin, Lewis Johnson, Paul Rosenbloom, and Jonathan Gratch, for their thorough and valuable feedback that helped shape the final version of my dissertation. In particular, I would like to thank Gaurav Sukhatme for his support and much valuable advice on school, job and life-related issues throughout my Ph.D. years. Heartfelt thanks to George Bekey for continuous encouragement and support from my first days as a student at USC.

I would like to thank my labmates in the Interaction Lab for the fun, friendly and stimulating atmosphere that made the past five years in the lab such a wonderful experience. My warmest thanks go to Brian Gerkey and Chad Jenkins, with whom I have shared my entire experience of a robotics graduate student, from the first years of searching for a research topic to the challenges of graduation and searching for a job. Many thanks to Brian for great spirit, delicious food, late-night deadlines and demos preparations, and for

always jumping to help with fixing a robot or an unruly computer. Many thanks to Chad for being a wonderful student at learning Romanian, for great conversations, humor, midnight trips to Kinkos, home-made cookies for my qualifying exam, hoop-warrior games, and for always lending a hand and having a word to cheer up. Thanks to Roger as well, for his kind support during busy times, especially before my defense. Many thanks to Dani Goldberg and Barry Werger, the first students in the Interaction Lab, for creating the spirit of the lab that continues now with new generations. Thanks to Dani for teaching me everything about the lab and for always having an answer to my questions. Thanks for the laughs, the HampsterDance and for giving such a great model to follow. Thanks to Barry for great music, driving across the country to robot competitions, and for initiation in the art of minimalist robotics. I will always have "Barry's Pioneer School" diploma with me. Many thanks to Helen Yan for being so eager to help with my robot experiments, and for being the only other woman student in the lab. Thanks to Amit Ramesh for great Indian food, and for being such a great colleague. Thanks to Evan Drumwright for the discussions and interest in Romanian life, and for the support and confidence he had always shown. Thanks to Chris Jones, for the great stories, the fun games of hoop-warrior, and for always lending a hand during robot demos. Thanks to Dylan Shell for bringing fashion back in the lab, for the cheerful spirit and for the numerous weekends and nights when we were the only people in the lab. Thanks to Carol Wald, for being so kind and helping so much with my work, from reading chapters of my thesis to making experiments with the robots for my dissertation.

Many thanks to Andrew Howard, Ash Tews, Paolo Pirjanian, Richard Vaughan, and Torbjorn Dahl for great discussions, encouragement and feedback on my work throughout my graduate years. I would also like to thank Stefan Weber, Jakob Fredslund, Esben H. Ostergaard, Kasper Stoy, Jens Wawerla, who visited our lab during the past five years, for the great spirit of camaraderie they brought to the lab.

Many thanks to Kusum Shori, Angela Megert, Aimee Barnard, Julieta De La Paz, and Bende Lagua for their amazing help with every administrative problem. The friendliness and the smile on their face made it always a pleasure to walk in their office.

I would like to give my very special thanks to the people closest to my heart: my family. My deepest love and gratitude to my parents, Terezia and Marin, for being the most wonderful parents in the world. I can never thank them enough for the love, support and for the many sacrifices they made so that I can achieve the best in my life. My warmest love and thanks to my parents and grandparents-in-law, Veronica and Gabriel, Ica and Ticu. Their love, care and encouragements found no obstacles in the thousands of miles between us, and they were always with me in my heart. My infinite love to Mircea, my husband, for making each day the happiest day of my life. The past five years are the most wonderful and dear to my heart, and are particularly special for representing the beginning of our lives together. I will cherish this time for all the special things that filled our hearts with joy and made us happy every day: from getting used to microwavable dinners during deadlines, to enjoying my culinary experiments when time had allowed me to cook, from sleepless nights in the lab, to long trips across the country, with early sunrises in the desert or sunsets at the beach. His love and understanding gave me the inspiration and energy that made this dissertation possible - I dedicate it to him, with all my heart.

The research presented in this dissertation has been performed at the Robotics Research Lab, within the Computer Science Department of the University of Southern California. The work was supported in part by DARPA Grant DABT63-99-1-0015 under the Mobile Autonomous Robot Software (MARS) program and by the ONR Defense University Research Instrumentation Program grant.

Contents

List Of Tables

List Of Figures

Abstract

This dissertation presents a framework that enables robots to learn complex tasks from one or several demonstrations by a teacher, based on a set of available underlying robot capabilities. The framework, inspired from the approach people use when teaching each other through demonstration, uses an *action-embedded* approach for task representation, and uses learning by having the robot perform the task along with the teacher during the demonstration.

Among humans, teaching skills or tasks is a complex process that relies on multiple means of interaction and learning, both on the part of the teacher and of the learner. In robotics, however, skill teaching has largely been addressed by using only one or very few of these modalities. This dissertation presents a novel approach that uses multiple modalities for instruction and learning, allowing a robot to learn representations of high-level, sequential tasks from *instructive demonstrations, generalization over multiple but sparse demonstrations* and by *practicing under a teacher's supervision*.

To enable this type of learning, the underlying robot control architecture should exhibit the following key properties: *modularity*, *reusability of existing modules*, *robustness and real-time response*, *support for learning* and *the ability to encode complex task representations*. Behavior-based control (BBC) is an effective methodology for robot control, which provides modularity and robust real-time properties, but is limited with respect to the other capabilities listed above. This dissertation presents a *hierarchical abstract behavior architecture* that extends the standard BBC in that: i) it allows for the representation and execution of complex, hierarchically structured tasks within a behavior-based framework; ii) it enables reusablity of behaviors through the use of *abstract behaviors*; and iii) it provides support for automatic generation of a behavior-based system.

The experimental validation and evaluation presented in the dissertation demonstrate that the proposed task representation, in conjunction with multiple instructional modalities, provides an effective approach for robot learning of high-level sequential tasks, based on a set of underlying capabilities (behaviors).

Chapter 1

Introduction

This chapter provides an overview of the dissertation and its contributions to increasing the learning and interactive capabilities of robots through *teaching by demonstrations*. It discusses the main challenges involved in this process, and presents the assumptions on which the proposed solution is based. This chapter also outlines the approach to task learning and introduces the *Hierarchical Abstract Behavior Architecture*, which employs *action-embedded representations* to facilitate the transfer of task knowledge from demonstrations.

One of the main goals of Robotics is that robots ultimately be used in real-world domains, helping people in their jobs, and even replacing them completely in harmful or dangerous environments. Advances in designing autonomous robots take us closer to this possibility and create an increasing interest in the area of human-robot interaction. As a result, robots will co-exist with people, and their desired role in the resulting "society" is essential in designing their capabilities and in determining the nature of their interaction with humans.

This dissertation proposes a framework that allows robots to interact with humans naturally in human-robot domains, and enables them to learn to perform new tasks from this interaction. This provides robots with new learning, autonomous control and interaction capabilities that increase their ability to perform in dynamic, unpredictable environments.

In existing and future human-robot environments, robots may be workers or members of a team, co-operating with other robots and people to solve and perform tasks. They may also be entertainers, such as museum tour-guides and robot pets, trying to capture humans' attention with their behavior. Finally, they may be emotional companions, looking to establish social relations and to make friends. Designing controllers for these types of tasks is usually done by people specialized in programming robots. Even for them, most often this is a complicated process, as it essentially requires creating by hand a new and different controller for each particular task. The number of situations which the robot may face and the wide spectrum of tasks it may have to perform make the job of robot programming difficult. Rather than *pre-programming* a robot for all the tasks its users might want it to perform, which is infeasible in most cases, it would be more useful if the robot could learn such tasks from the user, through flexible and natural interaction. The wide variety of domains mentioned above indicates the large spectrum of possible robot users, ranging from factory workers to hospital nurses, and from elderly people to young children. Since these users do not have the necessary computer programming skills to re-task and program the robots, it becomes of great interest to develop an approach for robot controller design that is accessible to all users, from the lay to the skilled.

The work presented in this dissertation aims to provide robots with task learning capabilities that address the above problems, and thus reduce the amount of time and expertise required for the development of an autonomous, intelligent robot system. A natural approach to this problem is to have the robots learn a particular task from a teacher's demonstration, thus increasing the ability of robots to interact with people, and relieving the user from writing controllers by hand.

The following section presents in more detail the challenging problem of teaching robots by demonstration and the goals of this dissertation.

1.1 Motivation and Goals

1.1.1 Teaching Robots by Demonstration

The *teaching by demonstration* paradigm allows the transfer of task knowledge from an expert teacher to a learner through the use of demonstrations (Figure 1.1).

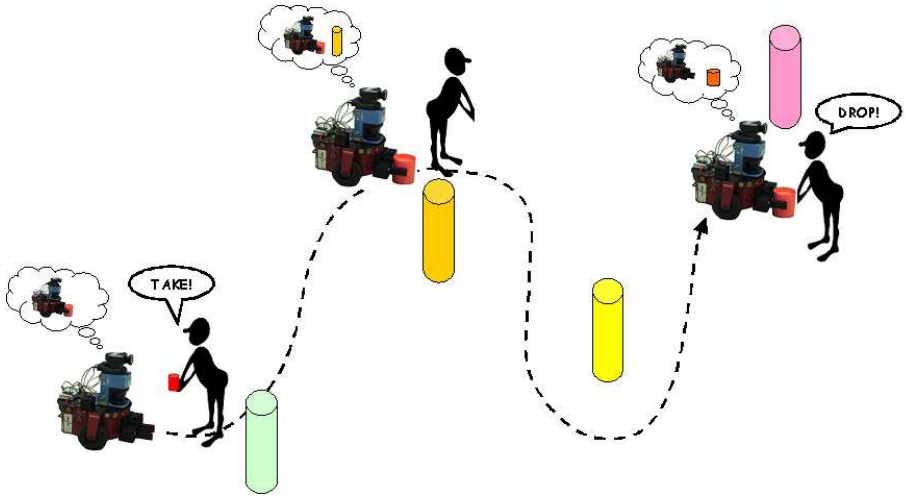


Figure 1.1: Transfer of task knowledge through demonstration

Although at the first look the problem seems simple (what could be hard in repeating what someone already showed?), human-robot teaching by demonstration poses numerous challenges:

- the robot's sensing capabilities are limited and different from human perception. What is the best way to give demonstrations to robots so as to maximize knowledge transfer?
- a robot's body is different than a human's. What *matching* mechanism is needed to create the mapping between a teacher's actions and the robot's own sensory-motor capabilities?
- learning is incremental, meaning that certain knowledge and skills could only be learned if there is already an existing appropriate substrate. What could a given robot learn and what are the required capabilities for learning it?

The following sections elaborate on these topics and present the approach taken in addressing these issues in the dissertation.

1.1.2 What Should the Robot Perceive?

An essential part of any robotic system employing teaching by demonstration is the ability of the robot to interpret the demonstration. Numerous methods have been employed for teaching robots through demonstrations, relying on different levels of complexity of the robot's sensory capabilities. Although each of these approaches has its own particularities, they can be classified into two main categories:

Learning by observation techniques, which rely on the learner passively observing the teacher's performance, and attempting to reproduce the observed behavior (Kuniyoshi, Inaba & Inoue 1994). These only allow a robot to gather heteroceptive (i.e., external) sensory information, in most cases from a camera. This requires the use of complex computer vision techniques to interpret the teacher's actions.

Learning from experience approaches demand that the robot take active part in the demonstration, performing the task along with the teacher and experiencing it through its own sensors. In addition to the visual information, the robot is also able to record proprioceptive information such as joint angles or positions relative to its own body. "Putting robots through" the task is performed through means appropriate for their own body: in humanoid robots this is mostly achieved through *teleoperation* (Pook & Ballard 1993), or by using virtual reality environments (Ogata & Takahashi 1994), while in the mobile robot domain *teacher-following* methods are typically used (Hayes & Demiris 1994, Nicolescu & Matarić 2001*b*).

In learning by observation a robot is faced with the major challenge of accurately perceiving the teacher's demonstration, which is subject to partial observability and noise in the majority of real-world domains. In addition, the learner must also be able to interpret the observations and map them to its own capabilities, while also accounting for differences in body structure with respect to the teacher. When learning by experience, a robot has the advantage that, in addition to using external sensory information, by executing the task to be learned it perceives the task through its own sensors: this avoids the above problems, as the robot now has an example of the task in terms of its own sensory-motor capabilities. In this dissertation:

We rely on experienced demonstration as a modality of the transfer of task knowledge from teachers to robots.

1.1.3 What Should the Robot Learn?

Irrespective of their particular method of demonstration, all approaches have to address the problem of what the robot should learn.

If the goal is to have the robot learn a particular sequence of movements (e.g., *waving-a-hand*) then learning to reproduce the demonstrated trajectory of the teacher is sufficient. This relies on the assumption that the environment does not change during the robot's performance and therefore cannot influence the behavior of the robot.

However, in most cases the tasks that the robot should learn depend on and are influenced by the state of the environment. In the case of learning a *grasping* capability, if the robot only records the exact trajectory of a particular instance of a demonstration (e.g., grasping an object from the coffee table), it would not be able to get to the object in a dynamic environment. This problem occurs due the fact that the robot has no representation of the high-level goals of the task: what is important in this case is to get the object irrespective of where it may be, not to follow a particular trajectory. In this dissertation:

We focus on learning high-level task representations, rather then precise trajectories of the teacher.

The level of complexity of the tasks the robot is to learn is also a factor in designing a suitable learning approach. Two important factors in this complexity are the *encoding of sequences* and the *granularity* of the components that represent the task.

With respect to sequencing, at one end of the spectrum are *reactive policies* (e.g., maze navigation (Hayes & Demiris 1994)) which constitute direct mappings from situations to robot actions, and have no explicit sequencing capabilities. At the other end are the *sequential skills/tasks* (e.g., assembly tasks (Kuniyoshi et al. 1994)) which encode explicit sequences of their steps. In the middle of this continuous

spectrum of approaches for encoding and executing of sequences are those that allow for implicit sequence representation, in which the sequences arise from the robot's interaction with the environment (Brooks, Connell & Ning 1988).

The components from which each of these classes of tasks can be built range from low-level actions (e.g., *turn-right-10-degrees*), to higher-level, more complex capabilities (e.g., *walking, grasping-a-cup*). The terminology referring to these categories of tasks is not always consistent in the literature. In Chapter 2 we discuss in more detail how these terms are generally used, and show how they are consistent with our classification.

From the discussion above, it is intuitive that the complexity of the tasks increases with sequential constraints and the use of higher-level components. In this dissertation:

We aim at giving robots the ability to learn complex tasks, involving temporal sequences and history.

1.1.4 What Should the Robot Know?

As mentioned in the previous section, complex tasks may be learned through elaborate combinations of existing controller components. Intuitively, more complicated tasks may be more easily learned from higher-level skills than from low-level commands.

The majority of approaches to teaching by demonstration attempt to teach the robot the task that it has to perform without the robot's having any pre-existing capabilities. While learning a new robot capability is not necessarily dependent on having other capabilities, learning complex tasks directly is more difficult, as the robot needs to learn both the necessary component modules and how to compose them to represent more elaborate structures. This limits the ability of such methods to acquire complex tasks.

In the domain of assistant/helper robots for the consumer, it would not helpful if the user had to teach the robot all the basic skills that are needed. It is, however, reasonable to assume that a robot would come equipped with a set of basic capabilities that cover the spectrum of tasks the robot is able to perform. The

user would then only have to teach the robot how to use and combine those skills to perform any particular task that requires them. For this dissertation:

We assume that the robots are equipped with a basic set of capabilities.

Summarizing our discussion, we state the goal of this dissertation as follows:

> **Goal:** Develop a flexible mechanism for robot learning of high level representations of robotic tasks, through demonstrations experienced through its own sensors, based on a set of underlying capabilities (behaviors) already available to the robot.

1.2 Task Representations for Learning from Demonstration

1.2.1 Challenges in Selecting the Robot Control Architecture

An important aspect of designing any robot system is to decide on what type of control architecture to use. This choice is determined in large part by the specifics of the domain in which the robot performs and by the particulars of the its tasks and required capabilities. For the conditions and goals outlined above, we consider that a suitable control architecture should be endowed with the following key features:

- **Modularity**. Our goal is to enable flexible and automatic development of robot controllers from existing robot capabilities, and therefore it is highly desirable that such skills be encapsulated in modules that could be easily combined into more complex structures.

- **Reusability of existing modules**. As different robot skills are employed in various different tasks, it is desirable that the component modules not require customization and redesign across tasks; they should be designed so as to maximize module reusability.

- **Support for learning**. Since our goal is to develop a method for learning task representations from teacher demonstrations, the robot's control architecture should provide the ability to construct such representations automatically from observations gathered during the training experience.

- **Robustness and real-time response**. Especially in dynamic environments, such as human domains, it is essential that a robot be able to respond quickly to sudden changes around it, while at the same time continuing to perform its task.

- **Ability to encode complex task representations**. As we aim at designing methods for learning of complex robot tasks, the underlying control architecture should have the ability to encode the necessary representations.

Behavior-based control (BBC) is an effective approach to robot control, which provides modularity and robust real-time properties. While it constitutes an excellent basis for this work, it has limitations regarding the other required capabilities. This dissertation investigates methods for extending BBC by proposing a *Hierarchical Behavior-Based Architecture* which provides capabilities for behavior reusability, learning, and complex representations. The architecture employs an *action-embedded representation* for task execution and learning. This means that the robot learns representations and tasks through performing of actions (in this case, behaviors), and not by observation alone. The process of learning such representations is described in more detail in Chapter 4.

1.2.2 Behavior-Based Control

Behavior-based control (BBC) (Matarić 1997*a*, Arkin 1998) has become one of the most popular approaches to embedded system control both in research and in practical applications. Behavior-based systems (BBS) employ a collection of concurrently executing processes, which take information from the sensors or other behaviors, and send commands to the actuators. These processes, called *behaviors*, represent time-extended actions that aim to achieve or maintain certain goals, and are the key building blocks for intelligent, more complex behavior.

An important principle to follow in BBS design is building behaviors that operate on a compatible execution time scale. Having a behavior that performs reasoning on a centralized world model, as in deliberative or hybrid systems, is not consistent with the behavior-based philosophy. Using both slow and fast behaviors in a BBS would make the system hybrid in terms of time-scale, and thus would not maintain BBC properties: fast, real-time responses, and similar representations and execution time.

An important property of BBS is their ability to contain state, and thus also construct and use distributed representations. However, this ability has been underused, so BBS are yet to be explored and extended to their full potential. The reason for this limitation is the fact that behaviors lack the abstract (symbolic, logic-like) representation that would allow them to be employed at a high level, like operators in a plan. Behaviors are typically invoked by built-in reactive conditions, and as a consequence, BBS are typically unnatural for, and thus rarely applied to complex problems that contain temporal sequences. Since we seek a method that allows learning representations of general tasks that would require the sequential activation of the robot's behaviors, in this dissertation we develop a mechanism that would allow first the representation and then the execution of such sequences.

A second limitation is that the vast majority of behavior-based systems are still designed by hand for a single task: the lack of abstract representation prevents automatic generation of BBS. Also, behaviors themselves, once refined, are usually reused by designers, enabling the gradual accumulation of *behavior libraries*. Unfortunately, the remainder of the system that utilizes such libraries is usually constructed by hand and involves customized behavior redesign in accordance with the specifics of any new task. The aim of this work is to conserve the robustness and real-time properties of behaviors and to develop a behavior representation that would support automatic generation of BBS and behavior reuse for multiple tasks (at least within a class of related tasks) while avoiding behavior redesign and even recompilation when switching to a different task.

In the next section we introduce the Hierarchical Abstract Behavior Architecture we developed to extend the capabilities of BBC and address the above limitations.

1.2.3 A Hierarchical Abstract Behavior-Based Architecture

The *Hierarchical Abstract Behavior-Based Architecture* allows the construction of complex task representations in the form of behavior networks. Within this architecture, the behaviors are built from two components: one related to perception, the other to action (Nicolescu & Matarić 2003*a*).

The perceptual component, called *abstract behavior*, encapsulates information about the behavior's preconditions and its goals. The active component, called *primitive behavior*, performs the actions that achieve the specified behavior goals under the given conditions. The *abstract behaviors* are the building blocks of the network representations, and the links between them represent task-specific precondition - postcondition dependencies. Within the resulting network, the activation of a behavior is dependent not only on its own preconditions (particular environmental states) but also on the postconditions of its relevant predecessors (*sequential preconditions* encoded as the network links). These links provide a simple and natural way of representing complex sequences of behaviors and, as we will see, the flexibility required to learn high-level task representations. In addition, using the links as task-specific activation conditions enables the reusability of behaviors. The network links provide a simple and natural way of representing complex sequences of behaviors and the flexibility required to learn high-level task representations.

The architecture also allows the construction of hierarchical representations; entire networks, representing tasks, can be abstracted into even higher-level components, in the form of *Network Abstract Behaviors*, and can be further combined to represent tasks of increasing complexity. The architecture and its components are described in more detail in Chapter 3.

1.3 Learning by Multi-Modal Demonstration

Among humans, teaching various tasks is a complex process which relies on multiple means for interaction and learning, both on the part of the teacher and of the learner. Used together, these modalities lead to effective teaching and learning approaches, respectively. However, in the robotics domain, task teaching has been mostly addressed by using only one or very few of these interactions. Human teachers rely on

concurrent use of multiple instructive modalities, including primarily demonstration, verbal instruction, attentional cues, or gestures. On the part of the learner, the process is also more complex than a one-shot teaching experience. *Students* are typically given one demonstration of the task and then they perform a set of practice trials under the supervision of the teacher, in order to show what was learned. If needed, during these runs the teacher provides *feedback cues* to indicate corrections (irrelevant actions or missing parts of the task). Alternatively, the teacher may also provide additional demonstrations that the learner could use for *generalization*. Most of these aspects are generally overlooked in the majority of robot teaching approaches, which focus mostly on only one, or very few of these instructive and learning modalities. We believe that considering these issues significantly improves the learning process by conveying more information about the task, while at the same time allowing for a very flexible robot teaching approach.

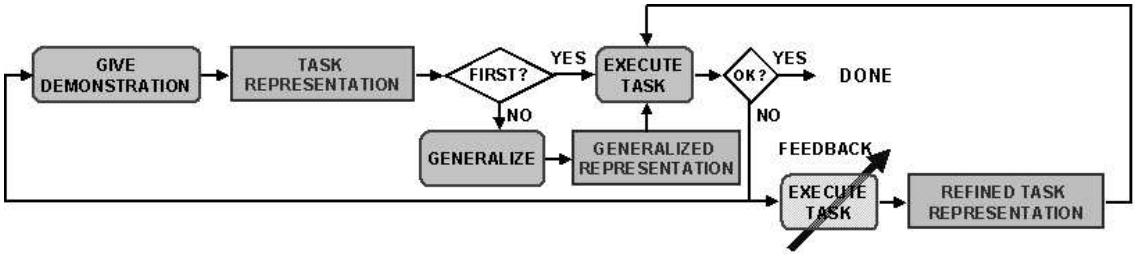


Figure 1.2: Learning and refining tasks through demonstrations, generalization and teacher feedback

This dissertation proposes a *multi-modal method* for learning representations of high level tasks, similar to what people use when teaching each other. Our overall strategy for learning and refining task representations is presented in Figure 1.2. The flexibility of this strategy consists in allowing the teacher to choose the methods considered most appropriate at any given time: after a first demonstration, either provide additional training examples or give feedback on what the robot has learned, during a practice trial. Our experiments show that similar effects can be achieved by following different teaching approaches (i.e., various combinations of demonstrations and feedback), allowing the teacher to adapt his or her teaching techniques to each particular case.

The thesis of this dissertation can be thus stated as follows:

Thesis: *Instructive demonstrations*, *generalization* over multiple demonstrations and *practice trials* can be successfully used for robot learning and refining of *high level representations* of complex tasks.

The following sections briefly describe the methods we developed for one-shot learning, generalization from multiple experiences and the use of practice for task refinement.

1.3.1 Single-Trial Learning

In order to learn a task from experience, the robot has to create a link between the observations (internal and external sensory information) gathered during the demonstration experience and its own skills (in our case behaviors) that would achieve the desired, observed effects. This is enabled by our particular architecture of the primitives (behaviors), which have both an active and a perceptual component. The perceptual component fires each time the observations match a primitive's goals, allowing the robot to identify during the demonstration the behaviors that are suited for and should be included in the corresponding task representation.

The advantage of putting the robot through the task during the demonstration is that the robot is able to adjust its behaviors (through their parameters) using the information gathered through its own sensors. If designed by hand, the parameters would have to be set by the programmer. Also, the observations gathered through the task experience provide temporal information for proper behavior sequencing, which would be tedious to design by hand for tasks with long temporal sequences. The methods for learning from a single demonstration are described in more detail in Chapter 4.

1.3.2 Generalization from Multiple Demonstrations

Generalization is an essential capability for a successful approach to learning by demonstration. Limited sensing capabilities, the quality of the teacher's demonstration, or the particularities of the environment may prevent a robot from learning the task correctly from only one trial.

The goal of the generalization mechanism is to build a task representation that encodes the specifics of each example while also capturing the common aspects among all demonstrations. Given the particular network-like representation we employ for the robot tasks, this is achieved by *merging* the corresponding networks at the nodes that are common, while maximizing the length of the common sequence of nodes between the tasks. Chapter 5 presents this methodology in more depth.

It is important to note that although we may rely on multiple demonstrations, since the goal is to reduce the amount of time and expertise that the process of robot controller design would require, the method is focused on learning from a very small number of trials.

1.3.3 Task Refinement through Practice

A limitation of our generalization mechanism is that it cannot account for situations when unnecessary task steps are repeatedly observed and included in the task representation, or when a robot consistently misses a relevant part of the task. To deal with such cases, this dissertation proposes the use of *practice* runs, during which the teacher observes the execution of the robot and is allowed to provide more accurate feedback to indicate where the problems occurred.

The advantage of this approach is that it does not require that the teacher have any knowledge about the specifics of the architecture and about how the learned task is encoded in order to provide appropriate guidance. The simple observation of the robot's performance is a sufficient indicator of what the robot has learned. As such, the teacher may indicate when seeing the robot performing unnecessary steps, and also may intervene when noticing that the robot has skipped essential parts of the task. Chapter 6 describes in detail how practice and teacher feedback improve the quality of the learned task representations.

1.4 Contributions

The main contributions of this dissertation are:

- The development of the *Hierarchical Abstract Behavior Architecture* as an extension of standard Behavior-Based Systems. This contribution includes the hierarchical behavior-network representations and an on-line learning algorithm that enables the automatic construction of these representations.

- The use of Abstract Behavior Networks and the learning algorithm for task teaching by demonstration in human-robot domains.

- Methods for generalizing and improving the accuracy of the learned representations by using multiple, but still sparse, demonstrations and feedback from the teacher.

The auxiliary contributions, developed in support of the ones above, are:

- The development of various behaviors, comprising a set of basic robot skills that allow a robot to track colored targets, pick up and drop objects, and open small doors.

- A vocabulary of *"guiding"* symbols, used for robot instruction and for improving the accuracy of the learned task representations.

1.5 Dissertation Outline

The reminder of this dissertation is organized as follows:

- **Chapter 1**: **Introduction**

  gives an introduction to the issues of knowledge transfer and communication in the human-robot interaction domains and motivates our use of action-embedded representations for addressing these issues.

- **Chapter 2**: **Related Work**

  provides a review of the relevant previous work in the area.

- **Chapter 3**: **Hierarchical Abstract Behavior Architecture**

  presents our behavior architecture, describes its use for representing and executing complex, hierarchically structured robotic tasks and gives illustrative examples demonstrating the use of the architecture.

- **Chapter 4**: **Learning from Experienced Demonstrations**

  describes the process of on-line construction of task representations from experienced demonstrations and interactions with a teacher and presents experiments showing the robot's learning abilities.

- **Chapter 5**: **Learning through Generalization from Multiple Examples**

  explains our method for combining several demonstrations into a unique task representation that best generalizes the presented task.

- **Chapter 6**: **Improving Learning through Practice and Teacher Feedback**

  describes the human-robot interactive approach for refining learned tasks through teacher-supervised practice.

- **Chapter 7**: **Conclusions**

  presents a summary of the work described in this dissertation.

- **Appendix A**: **Teaching Your Robot**

  provides the manual containing the information necessary for a non-expert user to teach a robot by demonstration.

Chapter 2

Related Work

This chapter presents a review of related work in areas of Robotics, Machine Learning and Artificial Intelligence. First, it presents examples of the most representative mobile robot control architectures from the perspective of their adaptation to the key features required in robot teaching by demonstration. Next, it discusses existing methods for encoding *task representations*, and it analyzes current approaches to task learning, with emphasis on techniques for learning from demonstration. The chapter also discusses current techniques for learning from multiple demonstrations. The purpose of this chapter is to relate the *Goal* of this dissertation to the existing work and give the motivation for the proposed solution.

The previous chapter introduced the goal of this dissertation as the ability to transfer complex task knowledge through *demonstration*, which does not require the need for programming skills or a strong robotics background on the part of the robot users. The discussion introduced a number of relevant key issues that influence the design of the solution: the choice of the control architecture, learning of task representations, approach for demonstration, and the ability to refine the learned tasks through multiple demonstrations and practice. This chapter discusses existing approaches to these problems and how they relate to the solution proposed in this dissertation.

2.1 Representation in Robot Control Architectures

As discussed in the previous chapter, choosing a suitable control architecture for robot teaching by demonstration is influenced by factors such as modularity, component reusability, robustness, and real-time properties, and by the ability to encode complex task representations, and to provide support for learning. This section presents the most representative approaches to robot control and discusses their capabilities relative to these issues. Although the focus will be on a restricted group of control architectures, it is important to note that there exists a continuous spectrum of approaches to robot control that bear relations to different classes of systems. Among these, the class of hierarchical, partial-order execution architectures will be discussed, as it is highly related to the *Hierarchical Abstract Behavior Architecture* proposed in this dissertation.

2.1.1 Reactive Systems

Control in *Reactive Systems* (RS) is based on a tight loop connecting the robot's sensors with its effectors, allowing for very fast response to changing and unstructured environments (Brooks 1986). The approach does not allow the robot to have memory, maintain state and any internal representations of the world, and thus is restricted to relatively simple classes of problems. The characteristics of these systems are summarized in Table 2.1 and discussed in more detail below.

Table 2.1: Summary of Reactive Systems capabilities with respect to our evaluation criteria

Representation	No
Task execution and representation richness	Simple (Reactive rules)
Robustness and real-time response	Very good
Modularity	Possible
Reusability	Possible
Support for learning	Only reactive policies

Ability to encode complex task representations. In the case of reactive systems, the minimal, if any, internal state information employed is used only to index the current environmental state into the possible set of situations, embedded in the reactive rules. This makes the systems unsuitable for more complex problems that require the use of internal models and memory.

Reactive systems implement collections of rules that map particular *situations* to particular *actions*. With respect to the organization, there are several ways in which these systems can be structured:

- **flat**: if the perceptual world can be divided into mutually exclusive or unique situations, there is only one action that can be triggered for a given situation, and the system is structured as a one-to-one mapping between those state and action spaces.

- **layered**: if there is more than one action that can be triggered for a given situation, a hierarchy of priorities for these actions is used to perform arbitration among them. These priorities can be pre-assigned at design-time, can be dynamic, or can be learned.

Examples of representative reactive architectures include *Universal Plans* (Schoppers 1987), systems with circuit semantics (Agre & Chapman 1990, Kaelbling & Rosenschein 1990, Horswill 1997), and T-R programs (Nilsson 1994).

In reactive systems there is no explicit representation of sequences: the principle in their design is that sequences, and intelligent behavior in general, should emerge from the dynamic interaction of the robot with the environment (Brooks 1990*b*), (Brooks et al. 1988). This is another constraint that limits RS to relatively simple tasks.

Robustness and real-time response. As mentioned above, the most important feature of reactive systems is their ability to respond quickly in dynamic, unstructured environments, which also gives them robustness for real-world domains. For problems which do not require complex reasoning abilities, RS are able to offer a fast reaction time.

Modularity. Depending on the particularities of the implementation, reactive systems may be constructed from modules, but many are also built from low-level actions. Therefore, both modular and non-modular controllers are supported within the reactive framework.

Reusability. The degree of reusability of various parts of the system is dependent on the specifics of the implementation. Rules and/or modules may be reused, while Universal Plans or T-R programs would have to be redesigned.

Support for learning. Reactive systems provide support for learning, but due to the limitations in task representation expressiveness, learning is limited to reactive policies.

Considering the above criteria, although the real-time and robustness properties would make reactive systems a good choice for the dynamic human-robot domains, their limitations in terms of task representations and learning would not make them suitable for our proposed challenge.

2.1.2 Deliberative Systems

The principle for control in *Deliberative Systems* (DS) is to use all the available sensory information and the stored knowledge to reason about what action should be performed. Typically, the sensory information is used to build an internal model of the world, which is next used to plan for possible paths that reach a given goal. Although the architecture allows a robot to look into the future and thus generate complex action sequences, it is best suited for environments that do not change rapidly. Real-world, unpredictable environments would require continuous updates of the internal world model and re-planning, which is time-consuming. Table 2.2 summarizes the characteristics of *Deliberative Systems* with respect to our evaluation criteria, features which are discussed in more detail below.

Ability to encode complex task representations. In contrast to reactive systems, *Deliberative Systems* make extensive use of internal representations of the world, since they require complete or very detailed world models in order to be able to reason about what actions to take next. World models are preprogrammed or built from sensory information, and give the ability to look ahead and predict the outcomes

Table 2.2: Summary of Deliberative Systems capabilities with respect to our evaluation criteria

Representation	Yes (extensive)
Task execution and representation richness	Complex (Plans)
Robustness and real-time response	Low
Modularity	Yes
Reusability	Yes
Support for learning	Operator preconditions/effects and plan caching

of possible actions in various situations. Thus, along with additional knowledge about the tasks, they are essential for constructing a plan that leads to the goal.

Deliberative systems enable the representation of complex sequences of actions, through their ability to reason about and represent possible paths to a goal. In the classical deliberative approach, once a plan is constructed, it is executed step by step, until the goal is achieved. However, most real-world domains are dynamic, and thus the world model on which a plan is based may change frequently, rendering the plan obsolete. Solutions to this problems include re-planning after a small number of steps (Moravec 1977, Moravec 1990), monitoring (Nilsson 1984), and incremental planning or plan caching (Firby 1989, Georgeff & Lansky 1987).

Robustness and real-time response. These properties follow directly from the discussion above. Since world model updating and replanning are both time consuming processes, when faced with a sudden change in the environment, a robot would not be able to find an action to perform within the short time available. The inability to react in real-time is thus an important factor for decreasing the robustness of the system.

Modularity. The building blocks in a *Deliberative System* are most often high-level, symbolic operators, with well-defined preconditions and post-conditions (Fikes & Nilsson 1971). These operators also enable a hierarchical decomposition of the tasks, resulting in a modular task representation.

Reusability. Since operators are self-contained modules, in the sense that they are completely specified by their preconditions and effects, they may be easily reused for other tasks.

Support for learning. In *Deliberative Systems* learning has been investigated at the level of learning plan operators (Wang 1996, Wang & Carbonell 1994), refining operators (Carbonell & Gil 1990), operator preconditions (Mitchell, Utgoff & Banerji 1993), procedural planning knowledge (Pearson 1996) and learning plans from experience (Carbonell 1983). Thus, although *Deliberative Systems* offer the ability to construct and represent solutions for complex, sequential tasks, with respect to our goal, their real-time and robustness properties do not make them a suitable choice for noisy, dynamic human-robot environments.

2.1.3 Hybrid Systems

Reactive and deliberative systems have advantages and disadvantages that complement each other. Reactive systems are responsive to unpredictable environments but can only be applied to relatively simple tasks. Deliberative systems can build and execute complex plans, but are impractical for many real-world domains.

In single robot control, the most common approach used to merge the advantages and bridge the gap between reactive and deliberative architectures are *Hybrid Systems*, which employ both a symbolic deliberative layer and a reactive layer. The deliberative component allows for looking ahead and reasoning about possible paths to the goal, relying on internal world representations and operating on a long time-scale. The reactive component deals with the robot's immediate needs, thus acting on a much shorter time scale. In order to provide a smooth interaction between the two layers, such systems, also called *three-layer systems*, require a middle layer whose role is to resolve the conflicts and the difference in the time scale and representation used by the other two layers (Gat 1998). Building the middle layer, which handles the complex interaction between these two layers is however the biggest challenge of hybrid systems design.

In Table 2.3 we present a summary of how the Hybrid Systems' capabilities match the challenges we proposed for our work.

Table 2.3: Summary of Hybrid Systems capabilities with respect to our evaluation criteria

Representation	Yes
Task execution and representation richness	Plans and reactive rules
Robustness and real-time response	Good
Modularity	Yes
Reusability	Partial
Support for learning	Operator preconditions/effects and plan caching

Ability to encode complex task representations. Through their deliberative component *Hybrid Systems* make extensive use of internal representation of the world and stored knowledge. Similar to deliberative systems, they require this information in order to be able to reason about possible paths to the goal.

Hybrid Systems combine the representational richness of complex plans with the simplicity of reactive modules. With respect to the execution, both deliberative and reactive layers provide a control component which has to be coordinated by the system's middle layer in order to avoid potential conflicts.

There are various approaches to designing hybrid systems. Agre & Chapman (1990) used a planner to give advice to the reactive control system, which could choose to use or ignore it. Arkin & Balch (1997) proposed a hybrid strategy that integrated a symbolic, deliberative level with a reactive (schema-based) controller, as a selection tool for the behavioral composition and parameters used during execution. Other representative hybrid architectures are Shakey (Nilsson 1984), RAPs (Firby 1989), SSS (Connell 1992), 3T (Bonasso, Firby, Gat, Miller & Slack 1997) and architectures relying on the Discrete Event Systems (DES) theory (Kosecká & Bogoni 1994, Huber & Grupen 1997).

Robustness and real-time response. The reactive component of *Hybrid Systems* enables a robot to react to immediate changes in the environment, increasing the level of robustness of an otherwise pure deliberative architecture. However, the time for reaction may be longer than in the purely reactive systems, due to possible conflicts between decisions taken by the reactive layer (responding to sudden challenges) and the deliberative layer (trying to achieve the task).

Modularity. *Hybrid systems* inherit their modular properties from their reactive and deliberative layers. At the symbolic level, the representations are built from abstract operators, which allow for a modular, hierarchical task decompositions. At the reactive level, as previously discussed, it is also possible to use a modular design.

Reusability. The components at the higher level, similar to the deliberative systems case, can be reused without any changes. At the lower level, depending on the choice of implementation, components may or may not be reused.

Support for learning. Hybrid systems can use learning techniques employed both in reactive and deliberative systems. Benson & Nilsson (1994) describe a hybrid architecture that learns the effects of an agent's actions in the environment, such that they could be used to construct more reliable plans.

So far in our analysis of mobile robotic architectures, *Hybrid systems* provide the best support for the issues involved in our challenge. The drawback of using such systems, however, is the complexity involved in designing the middle layer, responsible for the coordination of two significantly different control layers. We will consider *Hybrid Systems* in greater detail in the next section, which will present a comparison between these systems and behavior-based control.

2.1.4 Behavior-Based Systems

Behavior-Based Systems (BBS) are a biologically inspired approach to control in complex, dynamic environments, and thus are best suited to unstructured, real-world domains. They are built from goal/task-achieving modules (behaviors) that are executed continuously and in parallel, and have the ability to maintain state and build representations. However, distributing these representations effectively over the behavior structure is a major challenge and an important reason why the majority of BBS to date have not used complex representations.

In Table 2.4 we summarize the capabilities of BBS with respect to our evaluation criteria, and discuss these issues in more details next.

Table 2.4: Summary of Behavior-Based Systems capabilities

Representation	Underused ability
Task execution and representation richness	Relatively simple
Robustness and real-time response	Very good
Modularity	Yes
Reusability	Possible
Support for learning	Underused ability

Ability to encode complex task representations. The ability of behaviors to store state and to communicate with each other allows them to construct world models/representations. Due to the complexity of the representation construction process, this ability of BBS has been underused, but there is no intrinsic reason that would preclude the use of representation in BBS.

An early example of embedding representation and successfully integrating a deliberative component into BBS was done by Matarić (1992). The representation was constructed from behaviors, and was successfully used in the navigation domain for mapping and path planning.

The limited use of representation in BBS influences the complexity of the tasks to which they can be applied. Most often, behaviors are invoked by built-in, reactive conditions, which makes them unnatural for complex problems, requiring temporal sequencing. There are numerous approaches to the problem of behavior activation, also known as the *action selection* problem (Pirjanian 1999).

Maes (1990*b*) and Maes (1990*a*) describe a dynamic action selection mechanism for a situated agent, based on spreading activation within a network created from a given behavior repertoire. Brooks (1990*a*) and Connell (1990) use a selectionist approach to behavior arbitration. In that case, although behaviors are executed in parallel, and more than one behavior may produce an output at a given time, they are subjected to a prioritization scheme, which ultimately selects a "winner", whose outputs are propagated to the actuator. Other action selection mechanisms include fusion-based methods (Saffiotti, Ruspini & Konolige 1993, Steels 1994), in which outputs from various behaviors are blended together into a new

actuator command, voting mechanisms (Rosenblatt 1997), and winner-take-all strategies, which select the behavior with the largest number of votes received. A detailed description of action-selection strategies can be found in (Pirjanian 1999).

Robustness and real-time response. Since behaviors are fast-running processes, directly connecting the robot's sensors and its effectors, they enable the system to react in real-time to any environmental challenge. This also gives robustness to the system, since any changes in the world would be immediately processed and allow the robot to take a decision as soon as they occurred.

Modularity. BBS are modular by the nature of their components. They are constructed in a bottom-up fashion, starting with simple, survival modules at the bottom, and incrementally continuing with more complex capabilities toward the top.

Reusability. Typically, in order to design an behavior-based controller, behavior activation conditions are customized to capture the particularities of the tasks. Reusing behaviors for different tasks requires continuous changing and customization of behaviors, although the underlying behavior process remains the same.

Support for learning. Although the majority of approaches to learning in BBS have focused on learning policies, these systems have the ability to learn more complex task representations.

The analysis of the above characteristics of BBS shows that several of their potential abilities are typically underused, although they would be able to provide full support for learning, behavior reusability and the ability to encode complex task representations. BBS and Hybrid Systems have similar capabilities, although each of them has its own particulars (Arkin 1998, Matarić 1997*a*). Both systems have the same expressive and computational abilities: while Hybrid Systems use world models and search techniques to plan and look ahead, BBS distribute the representation across the behavior structure and are able to reason within the same time scale as the entire system, eliminating the need for a middle layer. Also, both have good real-time and robustness properties; however, while Hybrid Systems are well suited for environments with few real-time demands and where more elaborate internal models have to be used, BBS are best suited for environments with significant changes which also require some looking ahead.

Due to these essential properties, we consider that it is of particular interest to address the limitations of BBS and to extend BBS with abilities to encode complex representations, reusability and task learning (Nicolescu & Matarić 2002).

2.1.5 Hierarchical Architectures

A class of systems that is important to address due to the high relevance to the architecture proposed in this dissertation is that of hierarchical, partial-order execution architectures.

Such systems employ a hierarchical task structure, either completely provided *a priori* (Nicolescu & Matarić 2002) or dynamically expanded at execution time (Pearson, Huffman, Willis, Laird & Jones 1993, Simmons 1994, Tambe, Johnson, Jones, Koss, Laird, Rosenbloom & Schwamb 1995). The building blocks of these hierarchies are extended-time, continuous execution modules, which are activated based on environmental conditions and information generated by other modules. These representations use such conditions to move forward at any level of hierarchy and also downward in the hierarchy.

A hierarchical architecture that provides a more sophisticated control flow and allows for more expressive representations was presented by Sycara, Williamson & Decker (1996), and has been implemented in an Internet-based multi-agent system. Conditional branches are enabled through the use of "provision links", which relate the outcome of an action to the provision (i.e., the information needed by and action) of other action. The framework allows for both periodic and aperiodic actions, and also provides support for triggered actions, which are enabled in response to external events.

Hierarchical architectures for behavior control have also been developed for agents embedded in virtual environments. Bindiganavale, Schuler, Allbeck, Badler, Joshi & Palmer (2000) describes a *Parameterized Action Representation (PAR)*, to hierarchically encode the actions of a virtual human agent. The *Hierarchical Agent Control Architecture (HAC)* presented in Atkin, King, Westbrook, Heeringa, Hannon & Cohen (2001) uses three hierarchies: for action, for sensors, and for context. The hierarchy for structuring the sensory information into increasing levels of abstraction is similar to the goal representation for the *abstract behaviors*, but it is not linked with the behaviors whose goals it represents. In the above two architectures

the knowledge about the ordering of the steps in a task is maintained at a higher-level responsible with the activation of high-level actions or PARs (Atkin et al. 2001). While we encode this task-specific ordering information into behavior links, both types of representations provide support for successfully achieving the execution of a task having a hierarchical decomposition.

Having reviewed the main classes of robot control architectures, next we consider their ability to incorporate learning, as it is one of the key challenges of the work presented in this dissertation.

2.2 Task Representation in Robot Learning by Demonstration

2.2.1 Terminology

Learning is one of the greatest challenges in Artificial Intelligence and a key issue in robotics. It implies, in principle, the ability of a robot system to autonomously acquire new concepts and/or skills, to adapt to changes in the environment and/or task, and to improve its performance over time. Experience and/or instruction provided by a teacher may facilitate the learning process, by providing additional knowledge, advice, or specific instructions. The level at which learning occurs is dependent on the abilities that are desired from the system. To better define the focus of our learning problem and to be able to present the difficulties at the various learning levels, we are interested in distinguishing between *skill* and *task*-level approaches for teaching by demonstration, differentiated by the complexity of the capabilities being learned. The literature on learning from demonstration does not make a clear distinction among these various levels, and in numerous cases the notion of *task learning* has been used to refer to learning capabilities with *skill-level* complexity. To clarify the distinction we are aiming to make, and to explain what is implied by *skill* and *task* complexity, along with the complexity that each is able to encode, we propose definitions of these notions and discuss how they relate to existing terminology. As will be seen throughout the remainder of this chapter, these distinctions are essential for the design of systems that learn by demonstration.

> We consider a **skill** to be a pattern of activity which describes an aptitude or ability that achieves or maintains a particular goal.

Kaiser & Dillmann (1996) denote a *skill* as *"the learned power of doing a thing competently,"* giving *peg-insertion* and *door-opening* as examples of such skills. The system performs *actions* that achieve a particular goal, associated with each given skill. In Friedrich & Kaiser (1995), *elementary skills* (like *peg-insertion*) are defined as *"perception-action transformations involving no model knowledge, that represent basic capabilities of the robot."* These definitions convey the general idea of *skills* being temporal extended activities, relying on simpler motor capabilities also called *primitive actions* (Schaal 1999). These elementary actions can be *"simple point-to-point movements"* such as *go-forward* or *go-backward* and they serve as building blocks for higher-level compounds called either *movement primitives*, *movement schemas* (Arkin 1987, Arbib 1992), *basis behaviors* (Matarić 1997*a*), *units of action*, or *macro actions* (Schaal 1999). In turn, these are defined as *"sequences of actions that accomplish a complete goal-directed behavior"* (Schaal 1999). The blurry boundary between the meanings ascribed to these terms is clear, as Schaal argues that such primitives can be as simple as an elementary action for symbolic approaches to imitation (e.g., *go-forward*, *go-backward*), but that in fact it would be more useful for such primitives to encode *"complete temporal behaviors"* (like *grasping-a-cup*, *walking* or a *tennis-serve*), in order to provide a more compact state-action representation. In this dissertation we choose to take the second interpretation, and distinguish *primitive actions* as being the building components for *skills*.

Skill learning may thus be regarded as the learning of abilities (such as *avoiding obstacles*, *following light*) "from scratch," by enabling the robot to create a mapping between situations and actions. However, in some approaches more elaborate skill learning may involve creating situation-behavior mappings, such as learning to coordinate behaviors for walking in legged robots (Maes & Brooks 1990).

As discussed in the previous chapter, another factor that influences the complexity of any robot capability is the ability to use or incorporate temporal sequences. *Skills* could range from purely reactive actions with no sequence capabilities, to reactive skills in which sequencing may emerge from the interaction with the environment, and finally to skills with explicit sequence representations. An example of a completely reactive skill is learning to navigate a maze (Hayes & Demiris 1994), in which the robot learns to coordinate its "left", "right" and "forward" movements based on the state of the environment (e.g., opening to the

left, on corridor, etc.). Schaal (1997) shows the use of demonstration to speed up reinforcement learning for acquiring a pole balancing skill. Although sequences are not explicitly represented in the controller, the overall resulting behavior is a sequence of actions that allows the robot to perform this activity in an effective and competent manner.

The difference between *skill* and *task-level* complexity is described in Friedrich & Kaiser (1995) as follows: a *program schema (or task-level program)* is a *"a sequence of elementary skills including proper application conditions."* In Voyles & Khosla (2001) a set of such *skills* (or "sensory motor primitives") is said to *"approximate the set of primitives that a human uses to perform a task from the particular application domain."* Assuming an existing set of such skills, a task may be regarded as follows:

> We consider a **task** to be an activity involving the coordination of an existing set of *skills* in order to achieve a given set of goals.

This definition does not imply that tasks could not be directly built from elementary actions. However, such a non-modular approach is in general quite difficult, as it does not benefit from the ability to rely on already existing capabilities, either within the same or across different tasks. Designing such systems becomes difficult, as any given task would have to be completely re-written using a set of low-level elementary actions, and would result in cumbersome task representations.

Various types of control structures may be employed in building and executing of task representations: sequences, unordered execution, conditional branches, periodic skills, loops, and externally enabled actions, which may be embedded in both flat and hierarchical representations (Sycara et al. 1996). In the discussions following in this dissertation, it is considered that task-level learning is performed at a higher level of abstraction and takes into consideration principled methods for the representation and execution of complex problems which typically require temporal sequences and hierarchical decomposition. Although the proposed task learning approach is focused primarily on learning from demonstrations of sequenced tasks, the control architecture provides support for using other control structures. The current capabilities of the architecture are presented in more detail in Chapter 3 (Section 3.6).

This dissertation focuses on the issue of *task learning* to address the challenge of automating control system design. The following sections present current approaches to skill and task learning, and describe their applicability domains and the expressiveness of the learned representations.

2.2.2 Symbolic vs. Non-Symbolic Representations

Another new distinction that can be made with respect to the representations of robot skills/tasks is between *symbolic* and *non-symbolic* approaches to representation.

One form of encoding a robot's capabilities is in the form of a *control function (or policy)*. Such representations are defined as "*computing a command u in a state x at a time t*," in order to achieve a goal associated with the skill. Given a sufficient number of training examples, the policy is computed through function approximations techniques (Kaiser 1997). Such methods constitute the class of *non-symbolic* representations for robotic controllers, and are most often applied in articulated control.

In the context of learning from demonstration, the class of symbolic task representations includes hierarchical structures (Ikeuchi, Kawade & Suehiro 1993, Ikeuchi & Suehiro 1992), graphs (Friedrich & Dillmann 1995, Voyles & Khosla 2001), execution tree representations (Ogata & Takahashi 1994) of assembly tasks, or hierarchically structured knowledge of a flying task (van Lent & Laird 1999). In the area of learning assembly tasks, the high-level, symbolic representations are also built as sequences of particular types of contacts between the manipulated objects (Tominaga, Takamatsu, Ogawara, Kimura & Ikeuchi 2000). Executing the task involves reproducing the same sequence of contacts, using an *a priori* designed set of skills that can reproduce the required contact transitions. The high-level representations built by these approaches are generally separated from the level of robot control, which means that additional mechanisms are necessary to further process them and convert them into actual robot commands (Tominaga et al. 2000). The architecture proposed in this dissertation allows building of abstract task representations in terms of existing robot skills. As such, the robot is able to use the learned representation directly for performing the task.

2.2.3 Skill Learning

At the level of *skill* learning, the majority of approaches have been applied to learning movement primitives (Jenkins & Matarić 2003, Ramesh & Matarić 2002) or navigation skills (Demiris & Hayes 2002).

In the area of mobile robotics, Michael Kasper & von Puttkamer (2001) present a behavior-based approach for learning reactive motor behaviors (*door-passage, wall-following*) and outline a strategy for learning history-dependent behaviors. An interesting aspect of this work is that "teaching" can be performed by an already existing behavior running on the robot: this enables *behavior cloning*, in which the same functionality can be obtained by using different sensors for input. Also in the mobile robot domain, Hayes & Demiris (1994) demonstrated maze learning, i.e., learning turning behaviors, by following another robot teacher. The robot uses its own observations to relate the changes in the environment with its own forward, left, and right turn actions, and constructs a reactive policy representation of the task.

In the area of humanoid robotics, Jenkins & Matarić (2002) demonstrate learning of a set of *basis behaviors* or *perceptual-motor primitives* from data of multiple observed human motions. The approach, implemented on a 20 DOF dynamic humanoid simulation, uses spatio-temporal non-linear dimension reduction techniques to segment the human motion data. The obtained segments are clustered into *action units*, which are further generalized into parameterized primitives. The approach provides a unique model for robot imitation (Jenkins, Matarić & Weber 2000), as the obtained primitives can be used both for recognition and generation of complex humanoid movement through sequencing or superposition.

Various approaches for learning of manipulation skills, such as performing a peg insertion operation have also been demonstrated. While Kaiser & Dillmann (1996) use a control function representation for the learned skill, in Onda, Suehiro & Kitagaki (2002) the skill is represented as a sequence of contact state transitions that achieve a cylindrical peg insertion.

2.2.4 Task Learning

The existing approaches for task learning from demonstration have focused mostly on learning manipulation or assembly tasks using humanoid robots (Ikeuchi & Suehiro 1992). The representations built in these cases may be constructed in terms of high-level operators constituting capabilities of the robot (Kuniyoshi et al. 1994) or as sequences of events that the robots aim to reproduce at the time of the execution. Via-point representations of robot trajectories have also been used in humanoid learning of playing a tennis forehand (Miyamoto & Kawato 1998), or the game of kendama (Miyamoto, Schaal, Gandolfo, Gomi, Koike, Osu, Nakano, Wada & Kawato 1996). In these approaches, the use of a bi-directional theory for robot control employs a flow of information between the motor command and the trajectory planning levels, which helps to coordinate the high-level decisions with the reactive motor responses.

Pook & Ballard (1993) use Hidden Markov Models (HMM) to learn an egg-flipping task, built from the robot's *grasp, carry, press* and *slide* skills. The demonstrations, provided through teleoperation, allow the robot to learn the task sequence from a small number of demonstrations of the task. Another example that uses HMMs demonstrates learning of a *peg-in-the-hole skill* (Hovland, Sikka & McCarragher 1996), in which the states of the model are represented by the possible configurations of the assembly and the transitions correspond to moving the peg with various velocities in up to eight possible directions.

In the mobile robot domain, Hugues & Drogoul (2002) present a mechanism for learning a slalom task. The robot learns by creating a mapping between features in its visual field and wheel velocities. However, this solution is highly dependent on the particulars of the environment in which the task is being demonstrated. Since it relies on features detected on walls or doors, the robot would fail to reproduce the task in a different environment, even if the positions of the slalom poles and their relations to each other would be identical to those that are obtained during training.

Task learning from demonstration has also been addressed for authoring Intelligent Tutoring Systems (ITS) (Munro, Johnson, Pizzini, Surmon, Towne & Wogulis 1997). Since it is difficult to design autonomous tutors by manually encoding the knowledge of experts in a particular domain, systems for authoring ITS focus on learning this complex procedural knowledge and plans from demonstrations provided

by the experts. This is a very challenging problem, as the tutor needs a lot of information in addition to just being able to demonstrate a procedure to students (e.g., monitoring the students, recognizing valid sequences of actions and answering student's questions). The approach presented by Angros (2000) uses both training examples and autonomous experimentation to learn hierarchical task knowledge, while in the same time acquiring general-purpose operators that contain reusable domain knowledge. The advantage of using experimentation is that it provides the learning agent with more data for learning, thus helping refine operator preconditions, and reducing the instructor's effort in providing the demonstrations.

2.3 Robot Learning by Demonstration

This section presents the most representative approaches to teaching robots by demonstration with respect to the various means they employ for providing the demonstrations. It also includes a discussion of methods that integrate *demonstration* with other learning techniques.

2.3.1 Segmentation and Identification of Demonstration Traces

The main difficulty of any learning by demonstration approach is the interpretation of the long sequences of observations gathered from the instruction, as the robot has to segment the continuous stream of data coming from its sensors, then translate them into appropriate *skills* or *tasks*.

If the robot already has a basic set of capabilities, interpreting the demonstration becomes a problem of creating a mapping between the perceived action of the teacher to a set of existing primitives (Schaal 1999). Alternatively, the robot may also build a set of elementary operations by segmenting the demonstration trace, and then learn from them a general program structure (Friedrich & Dillmann 1995).

The method for skill identification proposed in this dissertation relates the observations made during the demonstration to the known goals of each of the robot's skills. This approach for identification takes inspiration from psychology: Meltzoff & Moore (1997) argue that for imitation, in early infancy, matching

is based on goal states for the motor system, and that at later developmental stages matching may be based on a temporal sequence of goal states, or on a transition between goal states.

Similar to the work proposed in this dissertation, Voyles & Khosla (2001) assume the existence of a set of elementary skills, and employ collections of gesture recognition agents (such as *touch*, *force*, etc.), and gesture interpretation agents (such as *guard-move, rotate*, etc.) to identify the component skills from the demonstration. The resulting task has the form of a finite state machine, in which each node incorporates multiple agents executing in parallel to achieve a desired action. However, this representation is not robust for dynamic domains, as the nodes themselves do not have any ability to detect changes in the environment. The architecture proposed in this dissertation relies on *behaviors*, which continuously receive sensory information and thus are able to react to unexpected situations.

Another approach for identifying instances of known primitives in new demonstration data is presented in Voyles, Morrow & Khosla (1997). Using Principal Component Analysis (PCA), primitives such as guarded moves and edge-to-surface alignment for a robotic arm can be learned and subsequently recognized during further demonstrations. The method is however limited by the condition that the operations should be decoupled (such as a *zguard* and a *yroll* movement of a PUMA robot arm), and is restricted to learning very simple and short sequences or combinations of movements.

Pook & Ballard (1993) use Learning Vector Quantization (LVQ) to identify patterns corresponding to existing robot operators in the sensory data obtained during teleoperation of those skills. A window sliding approach identifies the patterns in the overall sensory stream of a task demonstrated through teleoperation. The sequences of identified patterns are next passed to an HMM to segment the task into a sequence of primitives. However, the structure of the HMM is designed *a priori*, which limits the generality of the approach in learning arbitrary tasks.

Within the domain of learning assembly tasks from observation, various approaches proposed a segmentation of the demonstration trace based on contact relations between the objects involved in the task (Tominaga et al. 2000, Ikeuchi et al. 1993) or by the relation between the manipulated objects and the

teacher's hand (Kuniyoshi & Inoue 1993). The robots then use predefined skills to generate the same contact transitions as observed during the demonstration.

2.3.2 Learning by Observation

In the area of *task learning by observation* the most representative approaches to learning symbolic, hierarchical representations of assembly tasks have been proposed by Kuniyoshi et al. (1994) and Ikeuchi & Suehiro (1992). To interpret the demonstrations, these systems rely on complex computer vision techniques that segment the input data and allow for the recognition of the actions performed by a human demonstrator. However, due to the limitations of the vision systems, testing is performed in constrained environments with simple geometric blocks, and relying on accurate *a priori* information about the scene. Brand (1997) developed a system that does not require *a priori* knowledge about the scene to build a summary of object manipulation activities from continuous video. In it, visual events (such as varying spatial relations between hands, tools, and objects) are linked to causal events represented in an action grammar, thus maintaining a consistent causal structure over time. The method has been used to interpret a video of a disassembly task, performed using real tools and objects, but was restricted to a limited number of gestures of a human arm (e.g., *touch, put, grab*, etc.).

The difficulties encountered in the above examples are to be expected, as the methods completely rely on understanding image sequences of uninstrumented tasks. To address these issues, researchers have developed marker-based observation systems that can be used for the demonstration. By instrumenting the body of the agent performing the demonstration or the objects (Maeda, Ishido, Kikuchi & Arai 2002) with markers, the system can easily compute the Cartesian coordinates of manipulated parts, or the arm joint angle coordinates of the teacher.

2.3.2.1 Learning by Imitation

A particular case of learning by observation is *imitation*. This phenomenon, well studied in biology, developmental psychology, neuroscience and linguistics, is considered by Thorpe (1963) to be the act of

"copying of a novel otherwise improbable act or utterance, or some act for which there is clearly no instinctive tendency". A more formal definition of imitation is given by Mitchell (1987): *"... imitation occurs when something C (the copy) is produced by an organism and/or machine, where: C is similar to something else M (the model); registration of M is necessary for the production of C; and C is designed to be similar to M."*

Beside the functional role of learning new behaviors, imitation has an important role in developing social capabilities necessary to form interpersonal relationships (Dauthenhahn 1994). In robotics, Dautenhahn (1995) developed a "Huegellandschaft" habitat in which robots follow each other by maintaining body contact, to learn about each other's internal models or to detect if continuing the interaction is beneficial for the robot's current internal state. The interaction allows robots with different sensory capabilities to learn how to combine their abilities, and the social relationships they develop help them perform actions that they could not be able to do by themselves.

In the robotics domain, imitation usually takes the form of passive observations of the teacher's activities (Brand 1996). However, active approaches to imitation have also been developed. Demiris & Hayes (1999) rely on the use of forward models which, given a current state of the controlled object and the command to be applied to it, predict the object's next state. These models allow a robot to detect if the movements of the teacher match with any of its existing skills by testing the predictions of the models associated with those skills against the teacher's trajectory. If the teacher performs a movement that is not in the robot's repertoire, the passive imitation approach is employed to acquire the newly demonstrated behavior.

Imitation learning also allows robots to learn from each other a vocabulary of "words" representing properties of objects in the environment or actions shared between the teacher and the learner. The connectionist approach presented in Billard & Dautenhahn (1998), and Billard & Hayes (1998) also enables the robots to build sequences of such "words."

2.3.3 Learning by Experience

Another approach for providing demonstrations for robots is by actively involving them in the demonstration process. Using methods such as teleoperation, virtual reality environments, or teacher following, a teacher makes a robot perform and experience the tasks to be learned through its own sensors. The advantage of using such techniques is that the robot is freed from interpreting and relating the actions of a different body to its own. The acquired sensory information is in terms of the robot's own structure and sensory-motor skills. Below we discuss several representative approaches to learning through experienced demonstrations.

2.3.3.1 Teleoperation

Teleoperation is a very direct approach for teaching a robot by demonstration. Using data gloves (Voyles & Khosla 2001) or multiple DOFs trackballs (Kaiser & Dillmann 1996), a skilled operator controls the robot to perform a given task. Sensory information is typically gathered through sensors placed on the robot, which record the relevant parameters (such as position, pressure or forces applied) during the demonstration. These techniques enable robots to learn motion trajectories (Delson & West 1996) or manipulation tasks (e.g., Yang, Xu & Chen (1993)).

Using such "lead-through" teaching approaches (Todd 1986) requires that the demonstration be performed by a skilled teacher, as the performance of the teacher in demonstrating the task has a great influence on the learned capabilities. Another difficulty that may arise is that the teleoperation may be performed through instruments that are different than what the human operator would use in accomplishing the task. Also, the actual manipulation of the robot may influence the accuracy of the demonstration. These problems can be solved by training the robot in virtual environments, as discussed next.

2.3.3.2 Virtual Reality

The transfer of skill/task knowledge through experience does not necessarily require that the operator control a real, physical robot (Onda et al. 2002).

Ogata & Takahashi (1994) demonstrate a system for teaching an object manipulation task, in which the teacher performs an assembly task in a simulated, virtual environment. The demonstration is converted into a high-level, symbolic task representation, which is further translated into robot commands.

In another approach, Friedrich, Hofmann & Dillmann (1997) use a data glove to manipulate objects in a virtual environment. The task is represented as sequences of states defined as relations between the manipulated objects. At execution time, the sequence is mapped onto a set of symbolic operators that would produce the same sequence of effects as recorded during the demonstration.

2.3.3.3 Teacher Following

In the mobile robot domain, the approach that has been generally used for teaching by demonstration relies on following of a human or another robot teacher. The existing methods, however, have focused on learning reactive policies (Hayes & Demiris 1994), trajectories (Gaussier, Moga, Banquet & Quoy 1998) or properties of objects in the environment (Billard & Dautenhahn 1998).

The task demonstration technique presented in this dissertation also employs a teacher-following method for providing the demonstration (Nicolescu & Matarić 2001*a*). However, the learning approach we propose enables learning of high-level representations of sequential tasks, built from a set of skills (behaviors) available to the robot.

2.3.4 Combining Demonstration with Other Learning Techniques

Mixed approaches to learning by demonstration, which use the observations of an instructor in conjunction with other learning techniques, have also been proposed. This section presents the most representative of these methods and discusses how they improve upon methods relying on a single learning approach.

2.3.4.1 Demonstration and Reinforcement Learning

The most common approach used in conjunction with teacher demonstration is reinforcement learning (RL). In this context, demonstration is used in order to speed up the learning process of a standard RL system.

Lin (1991) proposed *experience replay* as a method to speed up reinforcement learning. In this approach an agent uses a given demonstration by repeatedly presenting the steps of that lesson in chronologically backward order. With this, a simulated robot is able to learn three different behaviors: *docking*, *wall-following* and *door-passing*, and the results presented demonstrate a significant increase in learning speed over traditional reinforcement learning approaches. Schaal (1997) used model-based reinforcement learning to speed up a system in which a 7 DOF robot arm learned the task of balancing a pole from a brief human demonstration. Price & Boutilier (1999) and Price & Boutilier (2001) proposed an *implicit imitation* framework, which uses reinforcement learning techniques that allow an agent to incorporate knowledge extracted from the observation of other agents. The observations of such skilled teachers can speed up learning by providing a RL agent information about situations not previously experienced. Compared with the standard approaches for imitation and teaching, the approach has the advantage that it allows the transfer of information between agents that do not necessarily have the same objectives and/or set of actions. The learning agent uses the knowledge implicit in observations of other agents to adapt it to its own needs: the observations focus the learner's attention to states of high interest for the mentors and thus help the learner find novel and better solutions for its own policy. Since the learner's and the mentors' policies may be different, the implicit imitation approach allows the learner to discard the influence of the teacher if it is considered irrelevant. In addition, the expert agent does not have to play the role of a teacher explicitly, and thus the method allows knowledge to be transfered even in the case of an unwilling teacher or from a teacher unable to change its behavior for this purpose. Another key aspect is that the implicit imitation framework allows for natural integration of multiple agents' expertise and thus enables learning from multiple skilled mentors, in contrast with standard approaches that rely on the experience of only one teacher. The approach produces excellent results in the the grid worlds with various sizes and structures. However, in real-world domains the number of an agent's possible states is much larger, and the observability of those states becomes an important issue. Dynamic domains may also pose problems, as a learner would continuously be forced to update its policy based on the new state of the environment:

previous high-rewarding states may now become blocked, and the learner would have to reassess its state values.

2.3.4.2 Demonstration and Conditioning

For robot teaching, researchers have also drawn inspiration from behaviorist psychology and animal training, which use rewards to stimulate the activation of a desired behavior under certain conditions.

Farrell, Maglio & Campbell (2001) introduce the notion of *programming by conditioning* which merges examples with conditioning through rewards to teach an animated fish how to swim. Users can interact with the fish by tapping on a sensitive screen, and demonstrate behaviors such as following or fleeing from another fish, by dragging it in the desired direction. The fish record the initial state of the tank and the actions for each given demonstration, and will further perform those actions each time they come across the same situations. The reward mechanism allows a user to interact with a fish at a high level of abstraction: rewards are given after several events when the fish behaved in a desired manner. Thus the rewards do not relate specifically to a certain action, but rather to a sequence of actions that the fish will learn to correlate.

Kaplan, Oudeyer, Kubinyi & Miklûsi (2002) propose the use of *clicker training* to teach a Sony Aibo robotic dog to perform sequences of actions such as spinning in place. The approach is inspired from dolphin and dog training and is based on the idea of operant conditioning: initially a clicker sound is associated with a primary reinforcer by giving the dog a reward. After making this association, the clicker becomes a secondary reinforcer and can act as a positive cue, meaning that a reward will be coming. For training, the animal is guided to perform the desired action through the use of the clicker. When the dog performs the desired behavior the teacher gives the actual reward. Word commands are associated with the behavior after the actions have been learned and then the behavior is tested and refined by rewarding the animal only when the exact behavior is performed.

2.3.4.3 Demonstration and User Intent

Incorporating information about user intentions regarding various aspects of a demonstration has also been considered for improving the quality of learning by demonstration.

Friedrich & Dillmann (1995) consider fusing user intention with demonstration information as additional means for instruction. The approach enables the robot to learn the correct task successfuly, but may become burdensome for the teacher as he or she needs to provide (at each step) information on what goals she/he had in mind, and what actions/used objects were relevant. In contrast, the method for task refinement presented in this dissertation relies solely on the teacher's observation of the robot's execution during practice (Nicolescu & Matarić 2003*b*).

User intent may also be inferred from the teacher's gestures during a demonstration (Yang, Xu & Chen 1997). Zollner, Rogalla, Dillmann & Zollner (2002) present an approach for distinguishing among multiple types of *pick & place* operations. In this approach, information from a camera head, and a data glove equipped with tactile sensors, a magnetic tracker, and force sensors, is incorporated to detect fine manipulations. Finger movements and forces on the fingertips are used by a Support Vector Machine (SVM) classifier to detect various types of grasping manipulations.

2.4 Learning from Multiple Demonstrations

The majority of approaches that rely on learning from multiple demonstrations are focused on performing function approximations for trajectory or movement primitive learning. Only a few methods to date have addressed this issue at the task level.

Ogawara, Takamatsu, Kimura & Ikeuchi (2002*b*) use a Hidden Markov Model (HMM) representation of robot arm trajectories in order to group them into independent clusters representing separate movement primitives. HMMs have also been applied for recognition of hand-written digits (Yang et al. 1997), among other tasks.

A connectionist approach for learning *wall-following* and *corridor-centering* skills for a Nomad Super Scout robot is presented in Larson & Voyles (2001). However, since the method relies on continuous consistency in the training sensory data, characteristic of goal-maintenance type of skills, the approach is not able to learn goal-achievement capabilities. Gaussier et al. (1998) designed a neural network architecture for learning to imitate sequences of movements, inspired from the memory processes in the brain. The method allows a KOALA mobile robot to learn to perform motion zigzag, square and star trajectories by following another robot teacher, but is dependent on very accurate teacher following techniques.

Another representative connectionist example is ALVINN (Pomerleau 1991). This system trained an artificial neural network to map camera images to steering directions taken by a human driver, in order to design an autonomous land vehicle. ALVINN learned a significant level of skill in driving on various roads and under different conditions after only 10 minutes of training.

In task-level learning, Pook & Ballard (1993) use HMMs to learn a task sequence from multiple demonstrations. Ogawara, Takamatsu, Kimura & Ikeuchi (2002*a*) present an approach to learning grasping manipulations, based on a small number of demonstrations. In their method, tasks are represented as sequences of interactions between a robot hand and source and target objects. Multiple demonstrations of the same task (i.e., sequences of hand-object interactions) are generalized using a dynamic programming approach for multiple sequence alignment (Fuellen 1997). This approach processes in batch all the examples given, and returns the sequence of task interactions that are present in all the examples. The common sequence of *"essential interactions"* constitutes the generalized representation for the task. This method is similar to the generalization approach proposed in this dissertation, in that it looks for the demonstrated task steps that have been present across all demonstrations. However, their approach relies on the fact that the steps essential for the task are always detected (i.e., they appear in all task models), which may not always be true due to the limited sensing capabilities of the robots and to the partial observability of the environment. By considering as part of the generalized task only the steps belonging to the common sequence, the alternate paths of execution that may be detected as a result of the generalization are ignored. The generalization method presented in this dissertation is incremental, allowing the incorporation of each

new demonstration after it was provided (Nicolescu & Matarić 2003*b*). This allows the teacher to stop giving new demonstrations immediately after observing that the robot had learned the important steps of the task.

2.5 Approaches to Skill/Task Refinement

Generalization from multiple examples is an essential means for refinement of skills or tasks learned from a teacher's demonstration. However, generalization alone is not sufficient if inconsistencies or repeated erroneous observations are perceived. To allow for increased learning accuracy, methods for revising the skills after the initial learning step are needed. Very few methods in learning by demonstration address this problem and the majority do not allow for further improvement or adaptation if the task is not learned correctly in the first set of trials.

One technique for refinement of skills after learning has been proposed by Kaiser (1997) and Kaiser & Dillmann (1997). Within this approach *good/not good* feedback was given at the end of a run in which the robot performed the demonstrated skill. This method also considered the refinement of learned skills through practice, by using an exploration element that alters the skill during execution. The *good/not good* feedback was used to assess the quality of the exploration. However, giving such delayed reward generates problems of credit assignment, reducing the accuracy of the task refinement process.

In this dissertation we propose a similar approach for refining of skills through practice under the teacher's supervision. However, by allowing the teacher to provide feedback during or right after an incorrect task step occurred, the approach enables the robot to identify the irrelevant actions precisely. In addition, the teacher may also intervene in situations when the robot had skipped essential parts of the task, and allow the robot to incorporate them into its task representation.

2.6 Other Approaches to Task Learning

2.6.1 Reinforcement Learning

One of the most widely used approaches to learning is reinforcement learning. What makes the method appealing is that the system learns on its own, through trial and error, relying only on its own experiences and the received reward. However, in order to provably converge to an optimal policy, the system requires a very large number of trials, such that the entire state/action space is covered. For physical robots, this is an extremely important problem, since due to time and power capacity limitations it is impossible to run an infinite number of trials.

Based on the principle of learning from reinforcement, several algorithms have been developed, along with proofs of optimity. Among the most important are the method of temporal differences, introduced by Sutton (1988) (proof in Dayan (1992)) and Q-learning (Watkins 1989, Watkins & Dayan 1992).

Examples of representations employed to encode skills learned from reinforcement include lookup tables (Watkins 1989), logic expressions (Kaelbling 1990*b*), (Kaelbling 1990*a*), classifier systems (Dorigo & Colombetti 1994), (Booker 1988), U-Trees (McCallum 1996), and neural networks (del R. Millan & Torras 1992). However, the majority of these approaches address the problem of *skill learning*, or learning reactive policies. With respect to sequence representation and learning, they are restricted to sequencing through the environment. Explicit sequences, such as the ones triggered through some form of internal state, are not represented or learned.

An approach for representing and learning hierarchical decompositions of tasks, and for sequencing the sub-task modules was introduced by Singh (1991), in the form of *compositional Q-learning*. Similarly, Colombetti & Dorigo (1994) present a classifier-based system that coordinates a hierarchy of behaviors. Sun & Peterson (1998), Sun, Peterson & Merrill (1998), and Sun & Sessions (2000) describe a hybrid model that integrates connectionist reinforcement and symbolic learning methods, to acquire procedural and declarative knowledge (in neural and symbolic representations respectively) for solving sequential

tasks. These systems are mostly applied to simple navigation domains, achieving sequencing of low-level actions, rather than of higher-level skills.

In the context of behavior-based robot learning, the majority of the reinforcement learning approaches have been at the level of learning *skills,* in the form of situation-behavior mappings. The method, in various forms, has been successfully applied to single-robot learning of various tasks, most commonly navigation (Dorigo & Colombetti 1997, del R. Millan 1996), hexapod walking (Maes & Brooks 1990), box-pushing (Mahadevan & Connell 1991), juggling in humanoid robots (Schaal & Atkeson 1994), and has also been successfully applied to multi-robot learning (Matarić 1997*b*), (Matarić 1994).

A more comprehensive review of reinforcement learning methods can be found in Kaelbling, Littman & Moore (1996).

There are many situations in which the use of reinforcement learning techniques is not appropriate, due to learning time limitations or simply because learning by trial anderror might not be a feasible option for the given task.

2.6.2 Methods for Learning from Examples

Along with the widely used methods for task learning previously discussed, other learning strategies have also been applied or have the potential for use in the robotic domain.

Inductive learning is a learning method that uses multiple examples in order to make predictions about future examples. A representative approach is presented by Sammut, Hurst, Kedzier & Michie (1992), who demonstrate the learning of a complex flying skill. Their approach builds one separate decision tree representation for the elevator, rollers, thrust and flaps of an aircraft from the observation of human teacher performances.

Case-based learning is an instance-based learning method based on storing and generalizing over given training examples. It is also called a *lazy learning* technique, since generalization is deferred until a new instance arrives, which has to be classified according to the existing knowledge. Cases similar to the new instance are used to construct a new solution, which, together with the results of applying it, is also

added to the existing knowledge. A particularity of case-base learning among the general instance-based learning techniques is that the instances are typically represented with richer, symbolic representations (instead of real-valued points), and therefore the mechanisms for retrieving similar instances are more elaborate.

Examples of case-based learning approaches have been applied to robot control (Thrun & O'Sullivan 1996), tracking and catching a ball (Aha & Salzberg 1993), and learning robot kinematics (van der Smagt, Groen & van het Groenewoud 1994). Atkeson, Moore & Schaal (1997) present a comprehensive review of case-base learning techniques for robot control.

Memory-based learning is a special form of case-based learning, in which details of all the experiences are memorized and stored. In the robotic domain, the method has been successfully applied for learning pole balancing, juggling and billiards (Moore, Atkeson & Schaal 1995, Schaal & Atkeson 1994).

Explanation-based learning is an analytical method for concept learning that relies on an already existing (usually symbolic) model of the domain, which is refined with each new observed example. Prior knowledge about the domain is used to explain how the observed training example satisfies the domain theory, then used to analyze the explanation in order to determine the conditions under which it holds. In the end, these conditions are also incorporated into the domain model.

In robotics, explanation-based learning techniques have been applied to learning sequences of operations in manipulation tasks (DeJong & Mooney 1986, Segre & DeJong 1985), recognition of environmental features (O'Sullivan, Mitchell & Thrun 1997), and navigation (Mitchell & Thrun 1993).

2.7 Learning to Improve Performance

An important issue related to the challenge of task learning is *learning models* that allow the improvement of a robot's performance.

Within the behavior-based framework, Matarić (1992) presents an example of learning a model of the environment, Michaud & Matarić (1998*a*) and Michaud & Matarić (1998*b*) describe an approach for

learning models of behavior dynamics from behavior history, and Goldberg & Matarić (1999), Goldberg & Matarić (2000*a*), Goldberg & Matarić (2000*b*) address the problem of learning models of interaction dynamics of behavior based systems.

Although such learning approaches do not address the specific problem of *task learning*, they deal with a gradual modification of the existing skills and policies, in order to maintain or increase the performance of the system and to adapt to dynamic, changing environments (Goldberg 2001).

2.8 Summary

This chapter presented a review of selected related work in the areas of Robotics, Machine Learning and Artificial Intelligence. It made an analysis of existing robotic control architectures, and described the most representative approaches to learning by demonstration from the perspective of the challenges set in this dissertation.

Chapter 3

Hierarchical Abstract Behavior Architecture

This chapter presents the *Hierachical Abstract Behavior Architecture*, an extension of standard behavior-based control, that allows for the representation and execution of complex, sequential, hierarchically structured tasks. It introduces the notion of *abstract* and *primitive* behaviors and their use in forming hierarchical robot task representation. The chapter also provides a presentation of the *behavior network* construct as the basis for task representation used in this work, and gives illustrative examples of behavior networks use.

3.1 Adapting Behaviors for Representation

In behavior-based control (BBC) behaviors typically consist of a collection of rules, taking inputs from sensors or other behaviors in the system, and sending outputs to the effectors, or other behaviors. The inputs determine the activation level of a behavior: whether it is on or not, and in some systems by how much. These are the activation *conditions* for behavior execution. For the purposes of the representation, we distinguish the following two types of activation conditions (behavior preconditions):

- *world preconditions* - conditions that activate the behaviors based on a particular state of the environment;

- *sequential preconditions* - task-dependent conditions that must be met before activating the behavior.

These are often postconditions of other existing behaviors, which allow for the description of complex temporal sequences.

In standard behavior-based control, both types of preconditions are encoded and tested together, without discrimination, thus hard-coding a particular solution. To change tasks and goals, one often makes the most changes to these preconditions, while much of the rest of the behavior remains unchanged. We achieve the ability to manipulate and change these conditions at an abstract representation level, separate from the behavior repertoire/library, by introducing *abstract behaviors*.

With those, behaviors are treated as high-level operators, and without loss of robustness can be employed to generate various strategies or plans for specific tasks. While classical planning requires a specific initial state, BBC provides general controllers that can handle a variety of initial conditions. With the use of abstract behaviors, we generate networks that are BBS, being triggered by whatever condition the environment presents.

In their operation, behaviors individually or as a group achieve and/or maintain the goals of the system, thus accomplishing the task. This methodology lends itself to the construction of highly effective special-purpose systems. This is thus both a strength and a weakness of the BBS approach. In order to lend generality to a given system, we first looked for a way to make the behaviors themselves more general, while still assuring that they would achieve and/or maintain the goals for which they are designed.

The key step in adapting specialized behaviors to more general use is in the separation of the activation conditions from the outputs or actions. By separating those conditions from the actions, we allow for a more general set of activation conditions for the behavior's actions (Figure 3.1). While this is not necessary for any single task, it is what provides generality to the system for handling multiple tasks. The pairing of a behavior's activation conditions and its effects, without the specification of its inner workings, constitutes an *abstract behavior*. Intuitively, this is simply an explicit specification of the behavior's execution conditions (i.e., preconditions), and its effects (i.e., postconditions). The result is an abstract and general operator much like those used in classical deliberative systems (Fikes & Nilsson 1971). The behaviors that

do the work that achieves the specified effects under the given conditions are called *primitive behaviors*, and may involve one or an entire collection of sequential or concurrently executing behaviors, again as is typical for BBS.

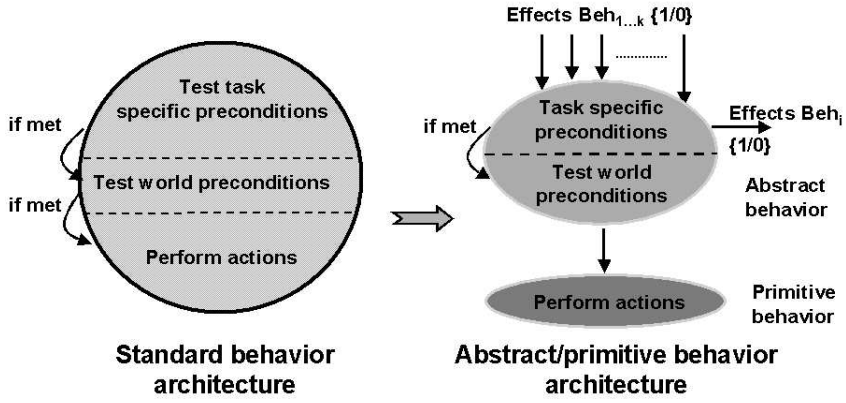


Figure 3.1: Adaptation of typical behaviors for abstract representations

Abstract and primitive behaviors can both be quite complex, just as they are within any system embedded in a physical environment. The abstract behavior conditions, as in any BBS, are typically far from low-granularity states, but are instead abstracted, either by hand or through a generalization process. If they were not, the benefits of using behaviors as a high-level representation would be lost. Similarly, the primitive behaviors are no lower level than standard BBS behaviors, meaning they are typically time-extended sequences of actions (e.g., go-home), not low-granularity single actions (e.g., turn-left-by-10-degrees).

Behavior networks are a means of specifying strategies or general "plans" in a way that merges the advantages of both abstract representations and behavior-based systems. The nodes in the networks are abstract behaviors, and the links between them represent precondition and postcondition dependencies. The task plan or strategy is represented as a network of such behaviors.

As in any BBS, when the conditions of a behavior are met, the behavior is activated. Similarly here, when the conditions of an abstract behavior are met, the behavior activates one or more primitive behaviors which achieve the effects specified in its postconditions. The network topology at the abstract behavior level encodes any task-specific behavior sequences, freeing up the primitive behaviors to be reused for a variety of tasks. Thus, since abstract behavior networks are computationally light-weight, solutions for

multiple tasks can be encoded within a single system, and dynamically switched, as we demonstrate in our implementation.

In the next sections we present the structure and functionality of abstract and primitive behaviors, then the construction of networks and their use.

3.2 Behavior Representation

3.2.1 Abstract Behaviors

Adapting specialized behaviors to general use requires a separation between the execution conditions and actions. We group these execution conditions and the behavior effects into abstract behaviors which have the role of activating the primitive behavior(s) that achieve the specified effects. In order to include behavior effects into the abstract representation we provide abstract behaviors information about the behavior's goals and a means of signaling their achievement to other behaviors that may utilize (and in fact rely on) these effects.

An important characteristic of our behaviors that makes our architecture very well suited for high-level, complex tasks, is that they are *parameterizable*. The behavior goals are represented as "predicate-like" structures in terms of the behavior parameters. The quotes above are used to stress that the effects are abstracted environmental states (continuously computed from the sensors) and not high-level symbols that are not grounded in perceptions. Thus, our behaviors become even closer, in terms of functionality, to the abstract operators used in symbolic architectures, allowing for multiple parameter bindings and therefore multiple and different goals for only one behavior, while still maintaining the real-time properties of behaviors.

The binary state of a behavior's goals (achieved or not) is fed into a behavior output connected to all the behaviors that require that condition to be true before they can become active. In this way, the information about the task-specific preconditions can be automatically obtained from the behavior network precondition-postcondition dependencies and dynamically changed (by simply rearranging the links) if

networks need to be switched at run-time. This allows for obtaining multiple solutions while using the same behaviors and maintaining the goals for which they have been designed.

As with operators in a plan, behaviors can undo each other's actions while trying to achieve their own goals (Chapman 1987). In BBS, such undesirable competition is typically handled either by mutually-exclusive behavior execution conditions, or by the behavior coordination mechanism (Pirjanian 1999). In this work, we take the former approach, and use explicit inhibition between behaviors to prevent destructive competition. This methodology directly fits into the behavior network representations: the network topology also includes inhibitory links between competitive behaviors.

Structurally, behaviors are composed of a set of processes, running continuously and concurrently with other behaviors, and an interface of input and output ports with which they can communicate with other behaviors. The implementation presented here utilizes the Port-Arbitrated Behavior paradigm presented by Werger (2000).

An abstract behavior has the following Input Ports:

- *UseBehavior* (binary input): signals if the behavior is used in the current network controller. The behavior is *enabled* if the port has a value of 1, and *disabled* otherwise.
- *ActivLevel*: sums the activation messages received from other behaviors; its value represents the behavior's activation level.
- *Inhibit* (binary input): a value of 1 signals that the behavior is inhibited by another behavior, a value of 0 signals that it is not.
- *Sensory Input*: a set of inputs from the environment, required to compute the status of the behavior's goals and world preconditions continuously.
- $Preconditions_{1...k}$: inputs from predecessor behaviors, whose execution influence the activation of the current behavior.
- *Continue* (binary input): coming from the corresponding primitive behavior(s) (discussed below).

The Output Ports of an abstract behavior are:

- *Active* (binary output): activates/deactivates the corresponding primitive behavior(s).
- *Effects* (binary output): signals the current status of the behavior's postconditions as computed from the sensory data.

A *disabled* or *inhibited* behavior does not perform any type of computation for the task. If *enabled* or *non-inhibited*, a behavior runs at a predefined rate at which it continuously checks or sends its inputs and outputs. However, only if *active* is the behavior allowed to send its action commands to the robot's actuator. In a discrete implementation, single activation and deactivation messages could be used per behavior, but this would not be as robust. Our system, as most BBS, uses continual messaging, in order to remain reactive to any changes that may occur (in the environment, the preconditions, etc.).

The abstract behavior activation mechanism is presented in the Section 3.3, which also presents the methods for encoding and executing tasks, in the form of behavior networks.

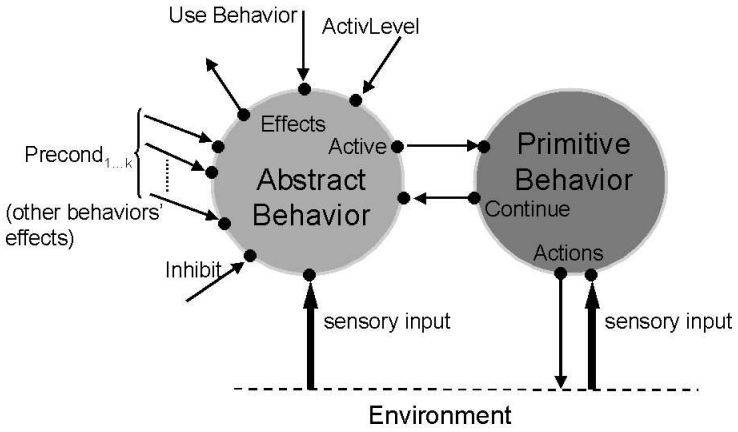


Figure 3.2: Structure of the inputs/outputs of an abstract and primitive behavior.

3.2.2 Primitive Behaviors

Primitive behaviors (see Figure 3.2) are activated by abstract behaviors via the $Active$ input; they are the behaviors that actually achieve the goals represented by the abstract behaviors.

Primitive behaviors use sensory information in order to compute the actions sent to the system's effectors via the $Actions$ output. The $Continue$ output is used to notify the corresponding abstract behavior that the execution of the primitive behavior is not yet finished, so that the abstract behavior continues to send activation. This output is used only in situations in which it is important that the execution of the primitive behavior not be interrupted, such as those caused by transience of sensory data. In these cases, it is necessary to extend the execution of the behavior until its completion. In all other situations, the abstract behavior can stop sending its activation at any time, according to its current conditions, thus deactivating the primitive behavior.

3.3 Behavior Networks

3.3.1 Components and Structure

The purpose of our abstract representation is to allow behavior-based systems to benefit from two important characteristics of symbolic systems.

First, in order to allow BBS to perform complex temporal sequences, we have embedded in the abstract behaviors the representation of the behavior's goals and the ability to signal their achievement through output links to the behaviors that are waiting for the completition of those goals. The connection of an $Effects$ output to the precondition inputs of other abstract behaviors thus enforces the order of behavior execution. The advantage of using behaviors can be seen again when the environment state changes either favorably (achieving the goals of some of the behaviors, without their being actually executed) or unfavorably (undoing some of the already achieved goals): since the conditions are continuously monitored, the system continues with execution of the behavior that should be active according to the environmental state (either jumps forward or goes back to a behavior that should be re-executed).

Second, we want to enable the automatic generation of behavior networks. As we described above, abstract behaviors specify the goals in a "predicate-like" form on the behavior's parameters, which makes them suitable for use with a general purpose planner in order to obtain a solution for a given task. Our

behavior networks, since they rely on real behaviors, also have the advantage that they could handle a variety of initial conditions within a single task representation, in contrast with typical plan-representations which are different for distinct initial conditions.

We distinguish among three types of *sequential preconditions*, which determine the activation of behaviors during the behavior network execution.

- **Permanent preconditions**: preconditions that must be met during the entire execution of the behavior. A change from met to not met in the state of these preconditions automatically deactivates the behavior. These preconditions enable the representation of sequences of the following type: *the effects of some actions must be permanently true during the execution of this behavior*.

- **Enabling preconditions**: preconditions that must be met immediately before the activation of a behavior. Their state can change during the behavior execution, without influencing the activation of the behavior. These preconditions enable the representation of sequences of the following type: *the achievement of some effects is sufficient to trigger the execution of this behavior*.

- **Ordering constraints**: preconditions that must have been met at some point before the behavior is activated. They enable the representation of sequences of the following type: *some actions must have been executed before this behavior can be executed*.

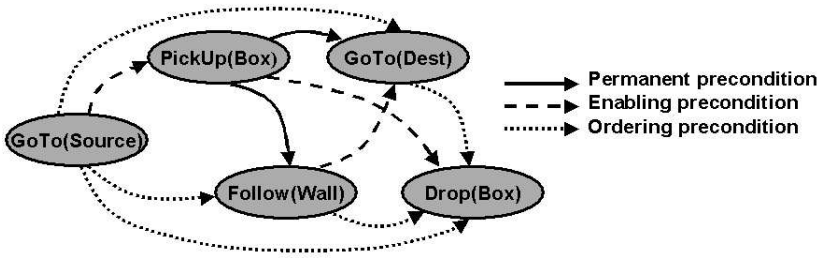


Figure 3.3: Example of a behavior network

From the perspective of a behavior whose goals are **Permanent preconditions** or **Enabling preconditions** for other behaviors, these goals are what the planning literature calls goals of *maintenance* and of *achievement*, respectively (Russell & Norvig 1995). In a network, a behavior can have any combination

of the above preconditions. The goals of a given behavior can be of maintenance for some successor behaviors and of achievement for others. Thus, since in our architecture there is no unique and consistent way of describing the conditions representing a behavior's goals, we distinguish them by the role they play as preconditions for the successor behaviors. Figure 3.3 shows a behavior network for a possible object transport task, using all three types of precondition-postcondition links.

3.3.2 Behavior-Network Execution

The behavior networks allow for two different modes of execution within the same representation, depending on the constraints on various parts of the task:

- *Sequential execution*, for the task segments containing temporal ordering contraints,
- *Opportunistic execution*, for the task segments for which the order of execution does not matter.

Sequentiality is enforced by the existence of precondition-postcondition dependencies between behaviors whose execution needs to be ordered. Opportunistic execution is achieved by not placing temporal dependencies between the behaviors which do not require a particular ordering. The ability to encode both these modes of execution within the same behavior network increases the expressive power of the architecture, through the embedding of multiple paths of execution within the same representation.

In a network requiring only *sequential execution*, since all behaviors are connected with ordering constraints, only a single behavior can be *active* at a given time. By introducing the *opportunistic* mode of execution, we allow multiple behaviors to be "suitable" for activation at the same time, if their own activation conditions are met. This raises the problem of concurrent access to the robot's actuators. To deal with this issue we choose the solution of *locking* the actuators that are used by a given behavior for the entire duration while that behavior is active. This provides a natural way of preventing multiple behaviors to have access to the same actuators, while still enabling the simultaneous execution of behaviors that control sets of actuators that are disjunct.

All the behaviors that are used in a network (i.e., have their *UseBehavior* port set) are continuously *running* (i.e., performing the computation described below), but only the behaviors that are *active* are sending commands to the actuators at a given time. A default **Init** behavior initiates the network links and detects the completion of the task.

Similar to Maes (1990*b*), we employ a continuous mechanism of activation spreading, from the behaviors that achieve the final goal to their predecessors (and so on), as follows: each behavior has an *Activation level* that represents the number of successor behaviors in the network that require the achievement of its postconditions. Any behavior with activation level greater than zero sends activation messages to all predecessor behaviors that do not have (or have not yet had) their postconditions met. The activation level is set to zero after each execution step, so it can be properly re-evaluated at each time, in order to respond to any environmental changes that might have occurred.

The activation spreading mechanism works together with precondition checking to determine whether a behavior should be active, and thus able to execute its actions. A behavior is activated **iff**:

(It is used in the current controller) AND

(It is not inhibited) AND

(Its controlled actuators are not locked) AND

(The *Activation level* $!= 0$) AND

(All **ordering constraints** $= TRUE$) AND

(All **permanent preconditions** $= TRUE$) AND

((All **enabling preconditions** $= TRUE$) OR

(the behavior was active in the previous step))

In the current implementation, checking precondition status is performed serially, but the process could also be implemented in parallel hardware. In the next section we introduce the concept of hierarchical behavior networks which allows for a hierarchical representation of a robot's task.

3.4 Hierarchical Behavior Networks

As robot tasks become more complex and begin to rely on previously developed skills, it is useful to have a hierarchical task representation that can encapsulate the complex dependencies.

The behavior network representation described so far allows only for *flat representations*, in which all the components are *abstract behaviors*. While the architecture is expressive and flexible, it does not have the modularity needed when new, more complex tasks would have to be created from already existing ones. The best solution is to specify the new task using abstractions of these existing modules, rather than combining their underlying behaviors into a bigger, flat network. In this way, only the precondition-postcondition dependencies at the higher level (between the two sub-networks) would have to be specified, reducing the connectivity of the network.

We enable this higher-level representation by introducing the notion of a *Network Abstract Behavior* (NAB), which abstracts away an entire behavior network into a single component. This allows the construction of hierarchical representations of robot tasks, whose components can be either *Abstract Behaviors* (ABs) or NABs, which can be further decomposed into lower level representations. An example of a generic hierarchical network representation is presented in Figure 3.4.

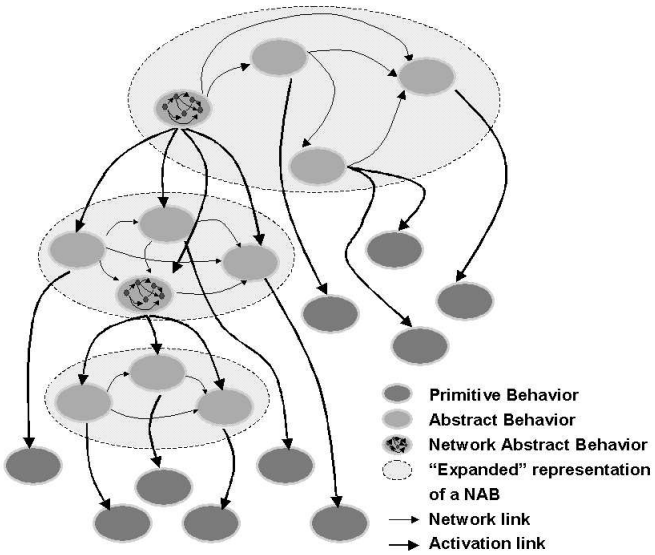


Figure 3.4: A generic hierarchical task representation

Functionally, a NAB is equivalent to a regular abstract behavior, in that it performs the same computation and plays the same role in the network. The postconditions of a NAB will be true when the execution of the subnetwork it represents is finished. The only difference between a NAB and an AB is in the connection of their *Active* output. For an abstract behavior, the *Active* output is connected to the *Active* input of the corresponding primitive behavior(s). For a NAB, the *Active* output is connected to the *UseBehavior* input of the corresponding component ABs or NABs. Thus, when a NAB is not active, all behaviors (ABs or NABs) which are a part of the subnetwork it represents are disabled, and therefore can be regarded as nonexistent for the task. When a NAB becomes active, all its underlying behaviors are enabled, and the subnetwork becomes the current "network" that is being executed. When the execution of the subnetwork finishes, the NAB signals to the successor behaviors the achievement of its goals, just as regular ABs do, and execution continues at the level of the network containing the NAB.

Formally, a behavior network is described as follows:

NETWORK-DESCRIPTION =

< Number of components (N), Component-Description, Topology-Description >

where,

Component-Description = AB-Description | ABN-Description >

AB-Description = < Component-ID, BehaviorID, Number of Parameters (P),

$\{$Parameter-Name, Parameter-Value$\}_P$ >

NAB-Description = < Component-ID, NETWORK-DESCRIPTION >

Topology-Description = < Number of Links (L), $\{$FromComp-ID, ToComp-ID, Link-Type$\}_L$ >

Link-Type = < Ordering | Enabling | Permanent >

This formalism describes a behavior network by the number N of its components (ABs or NABs), their descriptions, and the topological links between them. The **Component-ID** is a unique identifier of

the component within the network and the **FromComp-ID** and **ToComp-ID** are the IDs of the start and respectively end-points of a network link.

The next section presents an experimental validation of the architecture described above.

3.5 Experimental Validation

3.5.1 The Robot Testbed

The robot used to implement and test the described concepts is an ActivMedia Pioneer 2-DX mobile robot ($http://www.activrobots.com$), equipped with two rings of sonars (8 front and 8 rear), a SICK

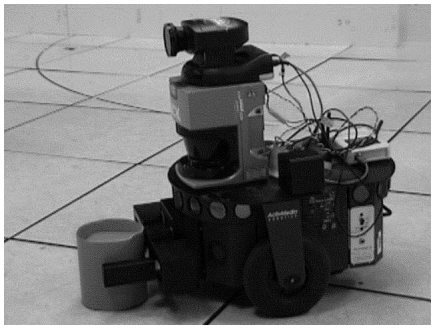


Figure 3.5: A Pioneer 2DX robot

laser range-finder, a pan-tilt-zoom color camera, a gripper, and on-board computation on a PC104 stack (Figure 3.5).

The behaviors were implemented using AYLLU (Werger 2000), an extension of the C language for development of distributed control systems for mobile robot teams.

3.5.2 Sequence Representation and Behavior Reusability

3.5.2.1 The Environment

To validate the proposed concepts, they were implemented on the described mobile robot given an object delivery task. The environment consisted of two sections, separated by a swinging door (Figure 3.6). The robot had to find a box, which could be in either section, and push it to the delivery point (or its Home).

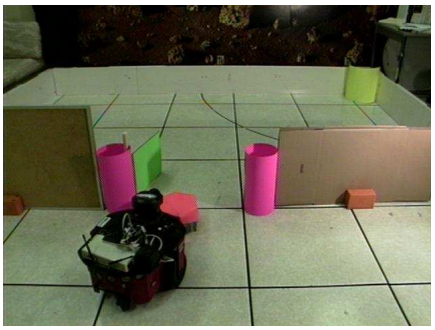


Figure 3.6: The environment for the delivery task

3.5.2.2 The Behavior Set

The networks for this task were constructed from the following repertoire of behaviors:

- **Localize** - the robot wanders around in order to localize itself with respect to its Home, represented by a colored beacon. Achieves *SideRoom(Home)=TRUE or SideRoom(OtherSide)=TRUE*.
- **GetBox** - the robot wanders in search of a colored box. Achieves *HaveBox = True* or signals *Timeout* in case the box cannot be found within a predetermined period of time in the current room.
- **GoTo(Door)** - the robot goes to the door. Achieves *AtPlace(Door) = True*.
- **OpenDoor** - the robot opens the closed door. Achieves *DoorOpened = True* and *HaveBox = False* (since the robot cannot carry anything in order to be able to open the door). Dealing with this type of conflicting goals is described in the next section.
- **GoThroughDoor** - the robot goes through the door to the next room. Achieves *SideRoom(RoomX) = True*; where RoomX can be either the room where Home is, or the other side.
- **GoTo(Home)** - the robot goes to its Home location. Achieves *AtPlace(Home) = True*.

3.5.2.3 Task Encoding with Behavior Networks

The BBS controller must accommodate various initial conditions: the robot may be in the same section as either the box and/or the delivery point, and the box may or may not be in the same section as the

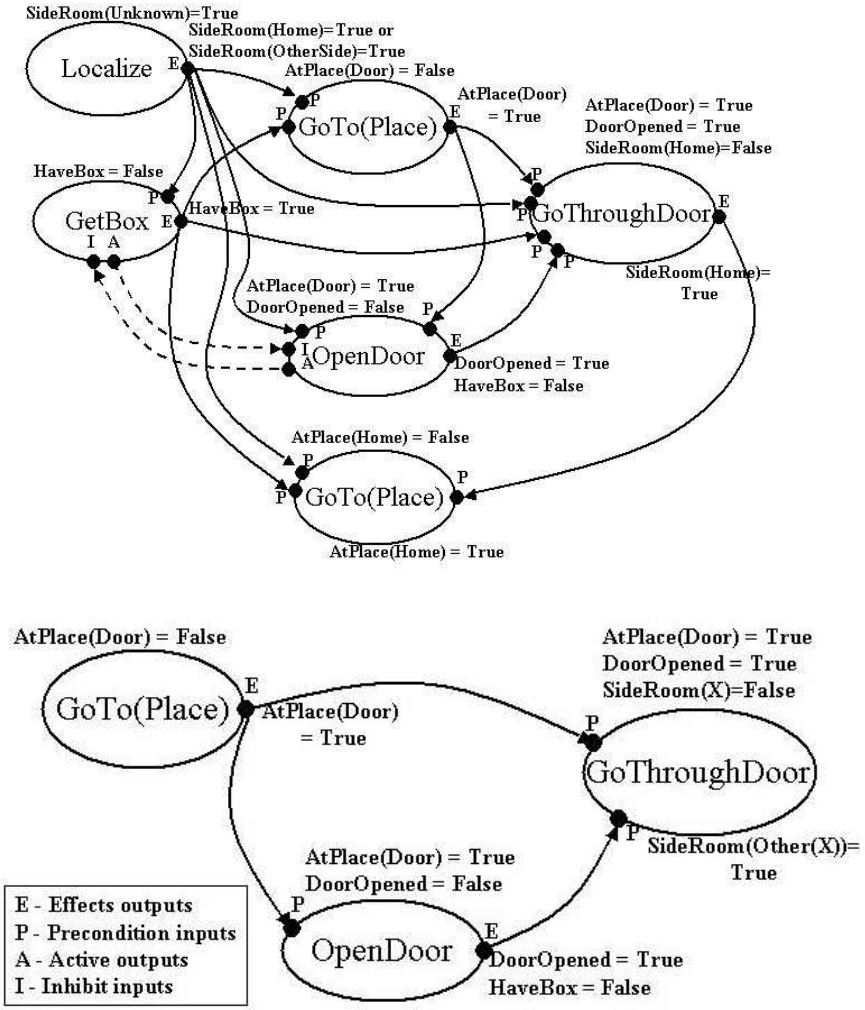


Figure 3.7: Structure of the behavior networks for the delivery task

delivery point. Note that this is not a large state space, which is why it lends itself to BBS solutions, but it is sufficiently versatile that it would require several different plans if pursued in a deliberative fashion. Our approach uses two networks which, together, account for all possibilities, and, as any BBS, adapts to uncertainty and changes that may occur (i.e., the robot or the box or both can be moved at any point).

Two different task plans were developed by hand for the delivery task and translated into behavior networks (Figure 3.7) that use the behavior set above. The robot automatically switched between the two

networks at run-time, according to predefined changes in the robot's internal state. This is the only built-in specific information in our system; it could have been avoided if external cues that could be sensed directly were available. In that case, we could have directly informed the robot when a network switch should occur.

It is important to note that since our networks rely on physically embedded behaviors which can handle a variety of initial conditions, we do not need to have a "plan" and thus a behavior network for each initial condition. Our solution makes use of only two alternate "plans" for the four possible initial conditions. This, of course, is not the only solution for the task, but we have chosen it because it captures the important aspects of the representation that we want to validate: 1) reuse of behaviors for different (sub)tasks without behavior redesign and 2) recompilation and dynamic switching between behavior networks.

The robot begins with the localization behavior (the only one for which all the execution conditions are met at that point) in order to determine in which room it is. Its goal of knowing the current location is a task precondition for all other behaviors (as can be seen by the network links from *Localize* to all other behaviors). Once localized, the robot starts looking for the box. If it finds it within a predetermined time, it continues to execute the current behavior network. If it fails to find the box, timeout is signaled, and the robot switches to another "plan", represented by the second behavior network. The alternate solution is to go to the other room, and look for the box there. The same *GoToDoor*, *OpenDoor*, *GoThroughDoor* behaviors are used in both networks. Even if the task-specific conditions that they are testing are different, no change has to be done to the behaviors themselves; they continue to run as before, only they check the $Effects$ outputs of a different set of behaviors. For example, the second network need not test the status of *GetBox* in order to go to the door and through it, as it would if the box had been found. At the completition of the alternate "plan" represented by the second behavior network, the robot switches again to the initial network and starts looking for the box in the room it is now in.

Each of the two behavior networks that we employed represents a solution to a different problem by itself: the first one is a solution for the delivery problem when both the robot and the box are in the same room and the second one is a solution for the task of going from one room to another. They both rely on

the same set of behaviors and the specifics of each tasks requires the behaviors to check different activation conditions in each case. However, due to the fact that those preconditions are embedded in the network topology (in the Effects-Precondition links), the behaviors can be reused without changes for different tasks and the tasks can be switched dynamically by simply rearranging those links.

3.5.2.4 Competitive Behaviors

In the delivery task, behavior competition arises between the *GetBox* and *OpenDoor* behaviors. While the former drives the robot to the box if it does not have it yet, the latter requires pushing the box aside in order to open the door. After getting the box, the *GetBox* behavior is no longer active and no longer inhibits *OpenDoor*. When *OpenDoor* becomes active, it inhibits *GetBox* until the door is opened. At that point it deactivates and in turns stops inhibiting *GetBox*, allowing the robot to again find the box and take it home through the opened door.

3.5.2.5 Results

To demonstrate the validity of the representation, and the ability to dynamically switch between behavior networks, we tested the delivery task from all four different initial conditions. For each of them we ran the robot four times, once with the door opened, in others with it closed. We found that irrespective of the initial conditions, the robot adapted itself to the state of the environment, activated the correct behavior network for that state, and executed its actions accordingly.

3.5.3 Hierarchical Representations

3.5.3.1 The Environment

The experimental setup for these experiments consists of a small **Orange** box and six cylindrical targets of the following colors: **Light Green, Yellow, Light Orange, Green, Pink** and **Orange** (Figure 3.8).

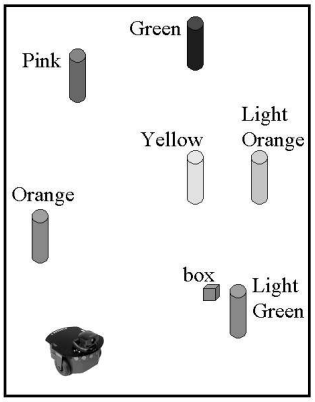
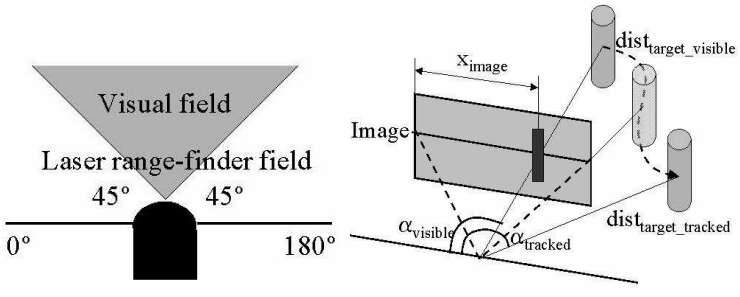


Figure 3.8: The environmental setup

3.5.3.2 The Behavior Set

For the experiments using hierarchical representations, the robot had a behavior set that allowed it to track colored targets and to pick up and drop colored objects:

- **PickUp(ColorOfObject)** - the robot picks up an object of the color *ColorOfObject*. Achieves *HaveObject = TRUE*.
- **Drop** - the robot drops what it has between the grippers. Achieves *HaveObject = FALSE*.
- **Track(ColorOfTarget, GoalDistance, GoalAngle)** - the robot tracks a target of the color *ColorOfTarget* until it gets at *GoalDistance* and *GoalAngle* to the target. Achieves *DistToTarget = GoalDistance AND AngleToTarget = GoalAngle*.



(a) Space coverage using laser rangefi nder and camera

(b) Principle for target tracking by merging vision and laser data

Figure 3.9: Merging laser and visual information for tracking

The **Track** behavior enabled the robot to follow colored targets at any distance in the [30, 200] cm range and any angle in the [0, 180] degree range. The information from the camera was merged with data from the laser range-finder in order to allow the robot to track targets that are outside of its visual field (see Figure 3.9). The robot used the camera to first detect the target and then to track it after it goes out of the visual field. As long as the target was visible to the camera, the robot used its position in the visual field (x_{image}) to infer an approximate angle to the target $\alpha_{visible}$ (the "approximation" in the angle comes from the fact that we are not using precise calibrated data from the camera and we compute it without taking into consideration the distance to the target). We get the real distance to the target $dist_{target\_visible}$ from the laser reading in a small neighborhood of the $\alpha_{visible}$ angle. When the target disappears from the visual field, we continued to track it by looking in the neighborhood of the previous position in terms of angle and distance which are now computed as $\alpha_{tracked}$ and $dist_{target\_tracked}$. Thus, the behavior gives the robot the ability to keep track of positions of objects around it, even if they are not currently visible, akin to working memory. This is extremely useful during the learning process, as discussed in the next section.

3.5.3.3 Task Encoding with Hierarchical Behavior Networks

The goal of the validation experiments is to demonstrate the key features of the presented architecture: hierarchical task representation, behavior reusability, and the ability for both sequential and opportunistic execution.

Toward this end, we considered a task consisting of sequencing of two subtasks: an **Object transport** task and a **Visit targets** task (Figure 3.10). The **Object transport** task required the sequential execution of its steps: go to the light green target, pick up the orange box, go through the gate formed by the yellow and light orange targets, go to the green target and drop the box there. As the figure shows, **GoThroughGate** itself had a subtask representation. The **Visit targets** task did not enforce the ordering of the target visits, thus allowing the robot to perform the task according to the particularities of the environment (i.e., visit the Pink, Light-Green, Yellow and Orange targets in the order in which they are encountered).

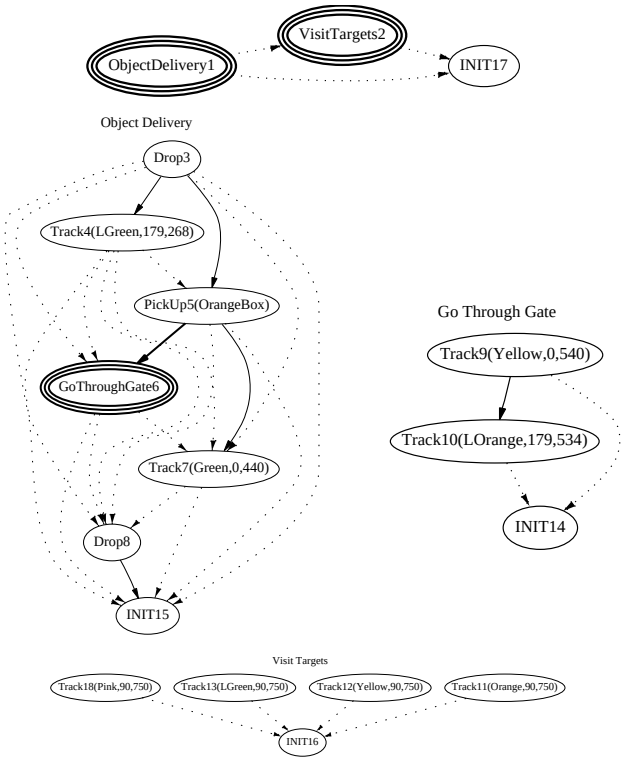


Figure 3.10: The hierarchical network representation. The subtasks (NABs) have 3-line borders, and the ABs have one-line borders. The numbers in the behaviors' names are their unique IDs.

3.5.3.4 Results

We performed 5 experiments; in all five, the robot correctly executed the task. The order in which the robot visited the targets during the **Visit targets** subtask is presented in Table 3.1. Since the robot's paths are different from one experiment to another, due to limited sensing and to uncertainty while searching for the colored targets, the robot opportunistically visited the targets as it encountered them.

Table 3.1: Order of target visits

Trial 1	Orange Pink Light-Green Yellow
Trial 2	Pink Yellow Orange Light-Green
Trial 3	Light-Green Yellow Orange Pink
Trial 4	Yellow Light-Green Orange Pink
Trial 5	Yellow Orange Pink Light-Green

We thus show that we can enforce *sequential* execution and that we can allow *opportunistic* execution, both within a behavior-based framework.

3.6 Expressive Power of Task Representations

The expressiveness of the proposed architecture can be analyzed within the formal framework of finite state automata and regular languages.

The behavior networks provide a directed acyclic graph (DAG) task representation. The behavior sequence that can be obtained by executing such a network is given by the graph's *topological* representation. This topology is obtained by applying a *topological sort* on the behavior network graph and constitutes a Finite State Automaton (FSA) representation of the task. Since the behavior network construct allows for alternate paths of execution, for certain FSA states there can be more than one possible transition, and therefore this FSA is nondeterministic. An FSA can only recognize regular languages, and thus the set of "strings" it accepts belong to a regular language over a given alphabet Σ. In our case, the "strings" constitute the set of behavior sequences that can be performed (i.e, the the tasks that can be encoded by the proposed representations) and the alphabet corresponds to the robot's set of skills.

The regular grammar that generates the language accepted by the proposed architecture has the following components:

- **A finite set N of nonterminal symbols.** Each symbol in this set corresponds to one of the robot's existing capabilities. Since the experiments presented throughout the following chapters rely on the behavior set described in Section 3.5.3.2, we consider the following set of nonterminal symbols: $N = \{t, d, p\}$, where t, d and p correspond to the **Track**, **Drop** and **PickUp** behaviors, respectively.

- **A finite set Σ of terminal symbols (disjoint from N).** We consider the following set of terminal symbols: $\Sigma = \{S, A\}$, where S is the *start symbol*.

- **A finite set P of production rules.** The following are the production rules that generate the complete set of "strings" (i.e, tasks) that can be built from the given alphabet of nonterminal symbols:

1. $S \rightarrow \epsilon$, where ϵ is the *empty* symbol.
2. $S \rightarrow t\,S$
3. $S \rightarrow d\,S$
4. $S \rightarrow d\,A$
5. $A \rightarrow p\,S$

Rules 3, 4 and 5 encode the fact that a **PickUp** behavior could only be performed if a **Drop** behavior has been previously executed. There is no restriction as to how many **Drop** behaviors could be performed in a row: although their execution would be redundant, it does not impose any applicability constraints.

Based on the proposed task representation and the current set of robot skills, the tasks that can be encoded are given by any regular expression on the existing vocabulary, with the constraint that for a current p (**PickUp**) in the task sequence there always has to be a d (**Drop**) somewhere in the sequence preceding it, but following the previous p or the start of the sequence.

In terms of semantic expressiveness, based on the definition of tasks given in Chapter 2, the architecture provides support for the following types of control structures:

- encoding of sequences that require a particular ordering of the task steps.
- encoding of unordered tasks, by allowing the robot to opportunistically choose among the task's steps based on the current state of the environments.
- execution of entire tasks or subtasks in a loop, as the control architecture supports the "reinstantiation" of networks after their completion and thus the ability to perform repetitive tasks.

Conditional branches, periodic skills, and skill activation based on external conditions are not currently supported, and they constitute subject for future work.

3.7 Discussion

The experiments presented in this chapter demonstated the advantages of the proposed architecture. They demonstrate the ability to encode tasks of increasing levels of abstraction, which facilitates the modular representation of higher-complexity tasks in the behavior-based framework. Also, by "abstracting" already known tasks into NABs, the complexity (connectivity) of the networks that might include those NABs as subtasks is greatly reduced. The abstraction eliminates unnecessary network links that would have to be specified **to** and **from** all the behaviors of a subtask in order to ensure proper execution and sequencing. For the experiment presented above, the number of network links would be increased from 31 to 60 if a flat representation had been used instead of the hierarchical one.

In the task representation presented above, based on a given set of behaviors, multiple instantiations of the same behaviors were used within the same NAB or in separate NABs, without customization or redesign, although in each case they had different activation conditions. Due to the fact that those pre-conditions were embedded in the network topology, the behaviors could be reused without changes in circumstances requiring different activation conditions.

3.8 Summary

This chapter presented a *Hierarchical Abstract Behavior Representation*, which extends the typical behavior-based architectures and addresses several of their limitations: the lack of abstract representation, which makes them unnatural for complex problems containing temporal sequences, and the lack of generality, which requires system redesign from a task to another. The architecture also allows for hierarchical task decomposition and both sequential and opportunistic task execution methods. In the next chapter we present an algorithm for learning such task representations from demonstrations by human and/or robot teachers.

Chapter 4

Learning from Experienced Demonstrations

This chapter presents an algorithm for learning behavior network representations of sequential tasks from a robot's own experiences of interacting with a human or a robot teacher. It presents the process of mapping the robot's observations to task representations and demonstrates the approach in various experiments, using human and robot teachers in clean or cluttered environments.

Teaching robots to perform various tasks by means of demonstration is a very natural approach to task-level learning. In this chapter we present a mechanism for learning representations of high-level tasks, based on a set of underlying skills already available to a robot. The approach allows the automation of robot controller design from the experience of a robot following a teacher's demonstration.

In our particular approach to learning, we use *learning by experienced demonstrations*. This implies that the robot actively participates in the demonstration provided by the teacher, and experiences the task through its own sensors. This is an essential characteristic of our approach, and is what provides the robot the data necessary for learning. In the mobile robot domain the demonstrations are achieved by following and interacting with the teacher. We assume that the teacher knows what skills the robot has, and also by what means (sensors) the robot can detect their achievement. The ability to learn from the observations gathered during the demonstration is based on the robot's ability to relate the observed states of the environment to the known effects of its own skills. The advantage of *putting the robot through* the task during the

demonstration is that the robot is able to adjust its behaviors (through their parameters) using the information gathered through its own sensors. In addition to experiencing parameter values directly, executing the task during the demonstration provides observations that contain temporal information for proper behavior sequencing, which would be tedious to design by hand for tasks with long temporal sequences.

The general idea of the learning algorithm is to relate the robot's observations during the demonstrations to its own skills. *The fact that the abstract behaviors embed representations of their goals enables the robot to create a mapping between what it observes with what it can perform.*

To validate the learning method, we focused on two types of domains (clean and cluttered) and we experiment with the robot's capability to learn from both human and robot teachers.

To describe the proposed approach for learning by demonstration we first describe the demonstration process, along with considerations about the observations gathered during this experience within the behavior-based framework. The following sections present the algorithm for constructing task representations from the experienced interaction with human or robot teachers and the experimental results.

4.1 The Demonstration Process

In its *learning* mode, during a demonstration, the robot physically follows the teacher, while all its available behaviors are continuously monitoring the status of their postconditions (without executing any of their actions). Whenever the observations match a primitive behavior's goals, this represents an example of the robot having seen something it is also able to do, and the corresponding *abstract behaviors* fires, allowing the robot to identify during its experience the behaviors that are relevant for the task being learned. The fact that the behavior postconditions are typically abstracted environmental states allows the robot to interpret high-level effects (such as approaching a target or being given an object). Thus, as mentioned before, *embedding the goals of each behavior into its own representation enables the robot to perform a mapping between what it observes and what it can perform*. This stands in contrast with traditional behavior-based

systems, which do not involve explicit goal representation and thus do not allow for any computational reflection.

If the robot is shown actions for which it does not have any representation, it will not be able to observe or learn from those experiences. For the purposes of our research, it is reasonable to accept this constraint; we are not aiming at teaching a robot new behaviors, but at showing the robot how to use its existing capabilities in order to perform more complicated tasks.

In our work, we are interested in the ability to use the proposed mechanism to learn from both human and robot teachers. It is desired that robots be able to use information learned from a human teacher to teach other robots in turn, thus allowing for an effective transfer of task knowledge from humans to robots and between robots themselves. With respect to the demonstration, a learner does not differentiate between a human or a robot teacher demonstration, both experiences being interpreted in the same way, but we expect that the performance of learning from a human is superior to that of learning from another robot. A human teacher facilitates the observations of the learner and waits for the robot so that it does not fall behind. A robot teacher demonstrates the task by simply executing it in front of the learner robot that follows it around, and in this sense the teacher plays only a naive role. Also, a robot teacher has to wonder around searching, due to its own limited sensing capabilities, thus affecting the observations of the robot learner.

4.2 Giving Instructions

Irrespective of the demonstration strategy being used, an important challenge for any method for learning by demonstration is to distinguish between the relevant and irrelevant information being perceived. Putting the entire responsibility on the learner to decide between relevant and irrelevant observations, such as when learning solely by observation, increases the complexity of the problem and leads to more complex, sometimes ineffective solutions. During demonstrations humans almost always make use of additional simple cues and instructions that facilitate the learning process and bias the learner's attention to the important

aspects of the demonstration (e.g., *"watch this"*, *"lift that"*, etc.). Although simple, these cues have a significant impact on the robot's learning performance: by relating them to the state of the environment at the moment when they are received, the learner is provided with information that may otherwise be impossible or extremely difficult to obtain only from the observed data. For example, while learning to go and pick up the mail, the robot can detect numerous other aspects along its path (e.g., passing by a chair, meeting another robot, etc.). These observations are irrelevant for getting the mail, and simple cues from the teacher could indicate that.

Therefore, in order for a robot to learn a task effectively, the teacher also needs to provide it with additional information beyond the perceived demonstration experience. To achieve this, we add *verbal instruction* to the existing demonstration capabilities of our system. With this, the teacher can provide the following types of information:

- **"HERE"** - indicates points in time during the demonstration when the environment presents aspects that are relevant for the task[1]. These indications are general (simple hints meaning *"pay attention now"*) and by no means spell out for the robot the representation of the presented task. While such indications allow the robot to distinguish some of the irrelevant observations, they may still not help it to learn the task perfectly. For this, generalization techniques (Chapter 5) and feedback-practice runs (Chapter 6) will be applied.

A robot teacher simply broadcasts a simple one-bit message when it has just accomplished execution of one of the behaviors in the task being demonstrated. The teacher's cue and the binary message carry the same information, that of considering the observations of the environment as relevant to the demonstrated task.

- **"TAKE", "DROP"** - instructions that induce the robot to perform certain actions during the demonstration (in this case **Pick Up** and **Drop** small objects), actions that would be otherwise impossible to

[1]The speech recognition system has been implemented after performing the experiments presented in this chapter. For these, instead of speaking the "HERE" command, the human teacher points out the saliencies by showing a bright color marker that can be detected by the robot's vision system.

trigger in a teacher-following-only learning approach[2]. In our case, we instruct the robot to open and close its gripper, when the task to be learned involves moving certain objects around.

- **"START", "DONE"** - instructions that signal the beginning and the end of a demonstration, respectively.

To demonstrate the benefit of using informative feedback cues, we used two different types of experimental validation setups: clean environments, in which only the objects relevant to the task are present, and environments with distractor objects that could hinder the learning process. The robot uses the teacher's indications to eliminate irrelevant observations, as described in Section 4.4.1.

Before presenting the algorithm for learning task representations, we discuss some key issues related to the nature of the observations gathered during a demonstration.

4.3 Key Issues for Task Observations

Due to the fact that our architecture is based on a behavior-based substrate, it is important to discuss the influence of BBS on the nature of the observations made by a robot during an experienced task demonstration.

Behaviors are time-extended processes, designed to achieve or maintain goals. In contrast with "operators" employed in classical symbolic architectures, whose execution is either instantaneous or has a well-defined, known duration, and always has the desired effects, behaviors operate in real-world, uncertain domains; it is typically not possible to determine *a priori* how long it will take a behavior to perform its actions before achieving the goals. The changes in the environment may cause a deactivation of a behavior before its goals are met, and a restart of the execution at a later point in time. Thus, unless the system is started in a state in which the goals of a behavior are already achieved, there will always exist an interval of time during which the behavior is active and performing its actions, prior to reaching the goal state. This

[2]At the time when the experimented presented in this chapter were performed, these commands were not yet implemented. In order to teach a robot that it had to pick up a box, the teacher would place it within the robot's open gripper. A reactive response to the presence of the object determined the robot to close the gripper and pick up the object. To teach a robot that it has to drop an object, the teacher would simply take it from the robot's closed gripper.

is an essential consideration that will be used later, to show how observations relate to and are mapped to representation. The possible evolution of a behavior's execution is represented by the state-transition diagram in Figure 4.1.

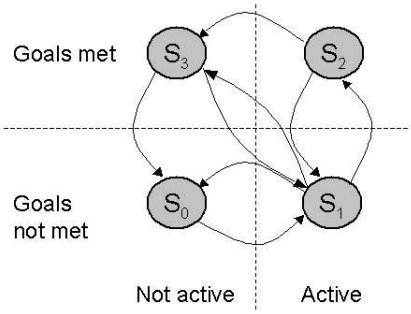


Figure 4.1: Evolution of a behavior's execution

The behavior evolution diagram captures another property of a behavior's goals, related to the types of conditions that behaviors can achieve. We distinguish between two types of goals (similar to those of operators in classical AI):

- **Maintenance goals:** conditions that have to be maintained for a time interval, as specified in the task description. This interval can either be a pre-determined amount of time, or be event-driven.
- **Achievement goals:** conditions that, once achieved, do not have to continue being maintained.

A behavior with a *maintenance goal* should continue to be active even after the achievement of its goal (transition from S_1 to S_2), while a behavior with an *achievement* goal is deactivated after attaining its goal (transition from S_1 to S_3 or from S_1 to S_0). For example, the goal of a *wall-following* behavior needs to be continuously maintained for the entire duration for which it is required to follow the wall, and therefore the behavior should remain active for that time interval. A *pick-up* behavior, however, is deactivated after grasping the object, as the condition it achieves (i.e., carrying an object) is persistent and does not have to be maintained through its actions.

In our architecture, the behaviors' goals are abstracted environmental states that are continuously evaluated based on the sensory information. During a demonstration of a task, a robot gathers observations of these states and records the time intervals during which the goals that they represent were detected as

being achieved. Depending on the particulars of the environment, or the style of the teacher, these intervals can take different amounts of time. In consequence, simply from these observations, a learner could not deduce if the duration of these intervals is important or even relevant for the demonstration.

Relying solely on the information provided by observation (i.e., time intervals during which conditions representing behavior goals have been detected as true), the learning process would only be able to capture the *ordering* aspect of the various stages of a demonstrated task, without the specifics about precise durations. Within the behavior-based framework it is also reasonable to assume that capturing the exact duration of the observed time intervals is not particularly relevant for the task representation because, as discussed above, the time required for a behavior to achieve its goals cannot be precisely determined. Different demonstrations (of different teachers, in different environments) of the same task might yield different observations, and something like *"Do X for time T"* might not even be relevant information. The relations between time intervals corresponding to two behaviors, however, help capture more powerful representations related to the notion of maintenance, such as *Do X while performing Y*. Details about this process are presented later in Section 4.4.2.

Thus, the class of tasks that we are interested in learning includes tasks for which the ordering of the steps is essential, but not their duration. For example, the robot can learn that it has to *pick up an object and then go home*, but not that it has to *follow a wall for 10 seconds*. Within this class of learnable tasks, relevant maintenance constraints can be learned from temporal relations between observations.

Another aspect of behavior *goals* is also important to address. As described in the previous chapter (Section 3.2.1), goals are abstracted environmental states, which are continuously computed and represented in a "predicate-like" form, which may or may not be dependent on the behavior's parameters. For example, for a **PickUp** behavior ,the goal would be represented as **HaveObject = TRUE**, whereas for a **Track(Color, Distance, Angle)** behavior, having as parameters the color of a target and the desired distance and angle to it, the goal is represented as the conjunction (**DistToTarget = GoalDist** and **AngleToTarget = GoalAngle**).

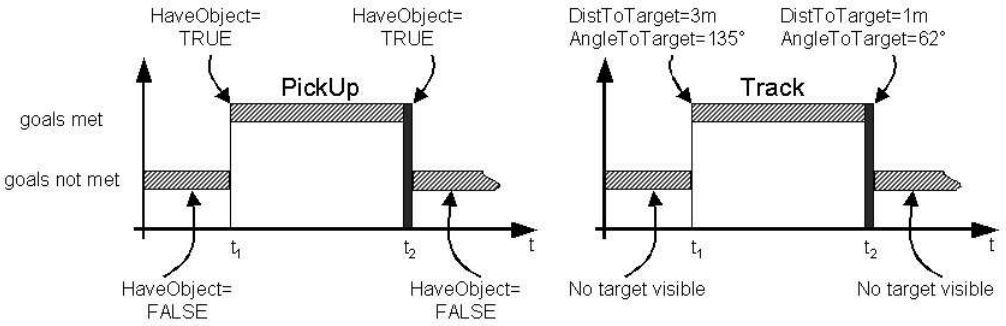


Figure 4.2: Observation of parametric and non-parametric behavior's goals

During a demonstration, since all behaviors are monitoring the status of their goals, the robot is able to detect when the postconditions of its behaviors became true. *For a behavior with non-parameterizable goals, the observation of its goals being achieved is represented by a time interval* (as discussed above) during which the conditions were met (Figure 4.2(left)). For a behavior with parameterizable goals the observation is also a time interval, representing moments in time when the conditions constituting the goals can be evaluated from the sensory data. This means that there exists a set of parameter values (computed from the sensory information for the conditions representing the goal), which, if being assigned as parameter values for that behavior, would be equivalent to the achievement of the behavior's goal in the same environmental configuration. *Thus, for a behavior with parameterizable goals, the observation is a time interval with a different set of parameter values for each moment of that interval* (Figure 4.2(right)). For the purpose of our work we consider that the last observed values for a behavior's parameters are the ones considered as the **goal values** for that behavior. Also, for all behaviors, we consider that the last observed moment is the one representing the true achievement of the behaviors' goals. We take this approach since we consider that the last moment when a behavior's postconditions were observed as being true is the most representative for the values that its parameters should take. While this may not hold true in all cases, it gives proper results for our behavior set and the class of tasks previously described.

4.4 Building Task Representation From Observations

This section presents the algorithm for constructing task representations from a teacher's demonstration.

4.4.1 The Learning Algorithm

Each node in a learned behavior network representation maintains the following information: behavior type, a unique ID (to differentiate between possible multiple **instances** of the same behavior), observed values of the behavior's parameters, interval of time I during which the behavior's postconditions were true, and a flag that shows whether the teacher has indicated any saliencies within I.

The general idea of the algorithm is to add to the network task representation an instance of all behaviors whose postconditions have been true during the demonstration, in the order of their occurrence. At the end of the teaching experience, the intervals of time when the effects of each of the behaviors have been true are known, and are used to determine if these effects have been active in overlapping intervals or in sequence. Based on the above information the algorithm generates the proper network links (i.e., precondition-postcondition dependencies), as described in the next section.

Behavior network construction

/* On-line processing (during the demonstration) */

1. At each time step, for each behavior:

- *If the behavior's postconditions have just become true:*

  ⇒ Add to the behavior network an instance of the behavior it corresponds to. (Along with it, save the time step as the start of behavior activation.)

- *Else, if the behavior's postconditions are true and have previously been true:*

  ⇒ Update the corresponding behavior in the network with its current parameter values, computed from observations, and any teacher-indicated saliency.

- *Else, if the behavior's postconditions have just become false:*

$\Rightarrow$ If in the network there is any previous behavior of the same type with an ending time within some ϵ from the starting time of the current behavior, merge the two behaviors (updating the information carried with the network nodes accordingly).

$/*$ Off-line processing (after the demonstration)$*/$

3. Filter the network in order to eliminate false indications of some behavior's effects. These are nodes with very small durations (determined experimentally as less than 2sec.) or unreasonable values of behavior parameters (detected distances to an object greater than 2 meters).

4. For each node representing a behavior instance J:

For each node representing a behavior instance K added to the network at a later time:

Compare the end-points of the interval I_j (corresponding to behavior J) with those of interval I_k (corresponding to behavior K):

- If $t2_j \geq t2_k$, then postconditions of J are **permanent preconditions** for K. Add this permanent link to behavior K in the network.
- If $t2_j < t2_k$ and $t1_k < t2_j$, then postconditions of J are **enabling preconditions** for K. Add this enabling link to behavior K in the network.
- If $t2_j < t1_k$, then postconditions of J are **ordering preconditions** for K. Add this ordering link to behavior K in the network.

The complexity of the learning algorithm is $O(n^2)$ in the number of behaviors in the learned network. This complexity is determined by the off-line stage of the algorithm, which requires a comparison of each behavior node with all its successors, in order to compute the proper behavior links between them.

4.4.2 Correctness of the Observation-Representation Mapping

In this section we demonstrate that the algorithm presented above produces a correct task representation. We start by stating the criteria for what we consider to be a correctly learned representation:

A learned representation of a task is **correct** if it enforces that **all** the stages of the real task be achieved in the same **order** shown during the demonstration.

The first aspect of the correctness criteria (that **all** stages of the demonstrated task be achieved) is addressed by the on-line processing segment of the learning algorithm. This process relates the observation of an interval $[s_i, e_i]$, corresponding to observing the goals of behavior Beh_i as being met, to having an instance of that behavior be a part of the task representation. Intuitively, this ensures that all the necessary behaviors needed to perform all the observed stages of the task are included in the representation.

The second aspect of the correctness criteria, related to the order in which the intermediary steps of the task are executed, is addressed by the off-line processing of the algorithm and requires a more detailed explanation.

At the end of the teacher-led demonstration the robot has a "back-bone" of the network, in the form of an ordered list of behaviors, inserted in the order in which their goals were detected as being achieved. From this, using the information about the specific time intervals corresponding to the time when the goals of each behavior have been true, the algorithm determines, for each pair of behaviors in that list, the type of precondition-postcondition relationships. Determining the dependence between two behaviors A and B is thus equivalent to *determining the type of overlap between the interval when the goals of behavior A were observed as true, and the interval during which behavior B was active, prior to achieving its own goals* (as described in Section 3.3), since this is what would describe the goals of A be preconditions for behavior B.

However, a demonstration provides only time intervals during which the goals of the behaviors were observed to be met, not when the behaviors should have been active. In Section 4.3 we concluded that a behavior should go through an *active* stage prior to achieving its goals and that the last moment of an observed interval is the one representative of achieving the goals. These considerations are represented in Figure 4.3. For a generic behavior B, the time the behavior should have been active (if the observed interval was $[b_1, b_2]$) is $[b_0, b_2]$, with the goals considered to be achieved at b_2 (the thick line). Thus,

the problem becomes finding how the interval relations between $[a_1 a_2]$ and $[b_0 b_2]$ map onto the existing precondition-postcondition dependencies (ordering, enabling, or permanent).

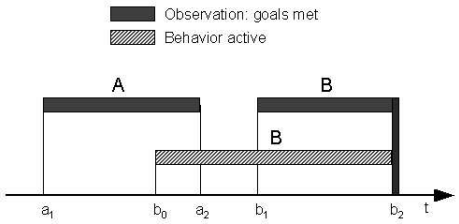


Figure 4.3: Interpretation of observation for two generic behaviors

Similar to the interval-based time representation of Allen (1983), we consider that between any two intervals there may be seven possible relations (Figure 4.4). Figure 4.5 shows how all the possible observations between two behaviors translate into precondition-postcondition dependencies: they follow directly from the the type of overlap between *the time when the postconditions of a behavior A are true* and *the time when behavior B should have been active*, based on the description of the behavior network links presented in Section 3.3.

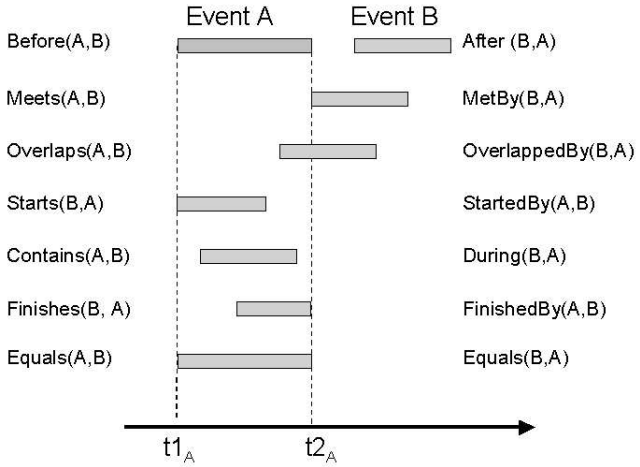


Figure 4.4: Allen's temporal relations

It is important to notice that for the observations **A starts B** and **A equals B** there is no associated relation. This is due to the fact that the goals of behavior A start to be true after the moment when behavior B is activated, thus not constituting **pre**-*conditions* for behavior B.

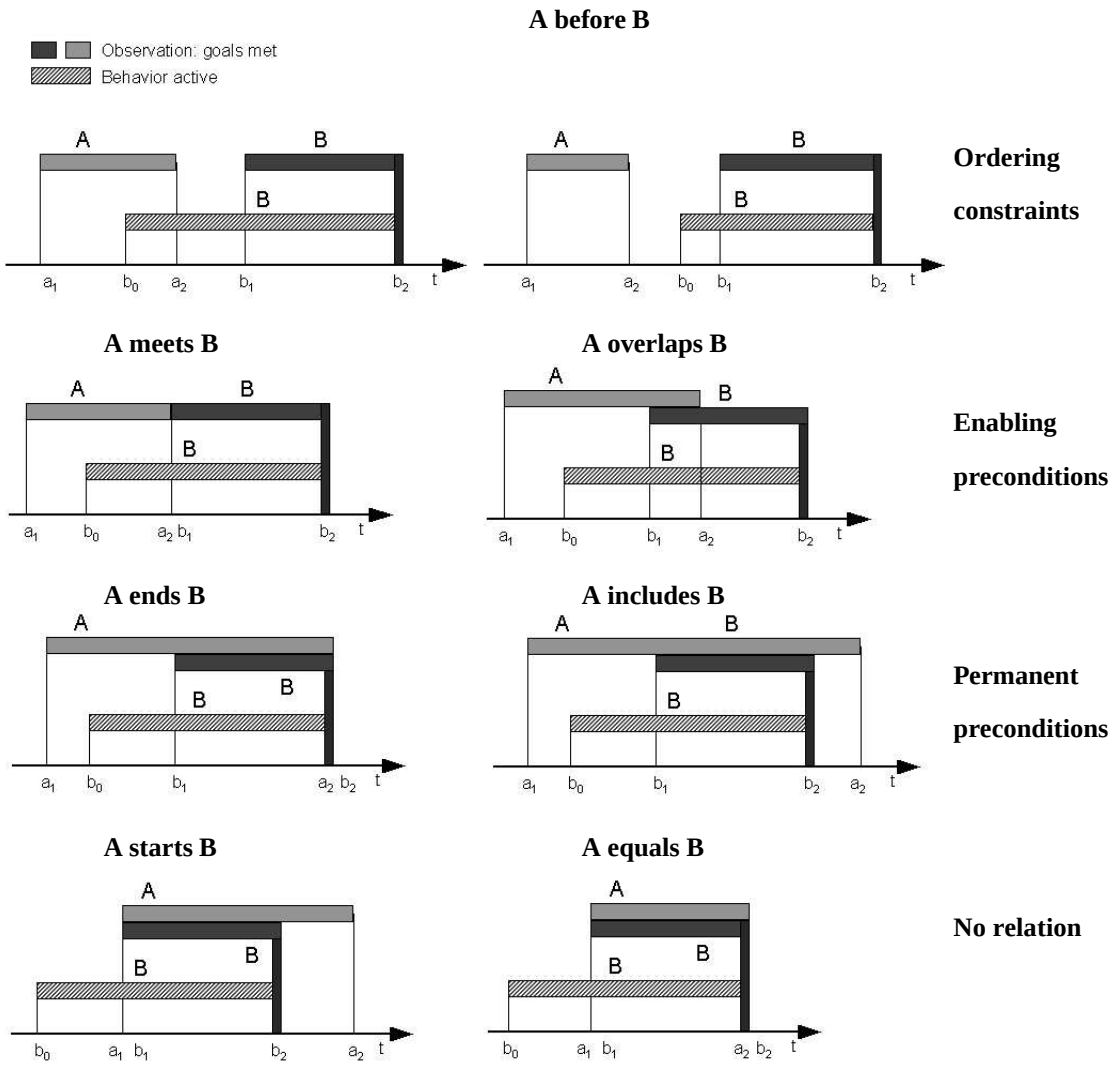


Figure 4.5: Mapping from observations to relations

Also, for all cases there exists another possible situation in which behavior B starts to be active before the goals of A are met, but in that case A would not be a predecessor of B. We do not consider those cases, since we assume that there will always be a predecessor-successor dependency between behaviors detected earlier in the demonstration and the ones detected later.

The mappings between the observations and the temporal relations are summarized in Table 4.1. Thus we conclude that the presented network construction algorithm is correct.

Table 4.1: Observation-Relation Mappings

Observation	Relation
A before B	Ordering constraint
A meets B	Enabling precondition
A overlaps B	Enabling precondition
A starts B	No relation
A includes B	Permanent precondition
A ends B	Permanent precondition
A equals B	No relation

4.5 Experimental Validation

We designed several different experiments that rely on navigation and object manipulation skills of the robots. First, we report on the performance of learning from human teachers in clean environments, followed by learning in cluttered environments. In the latter experimental setup we also address the issue of knowledge transfer between robots, in robot(teacher)-robot(learner) demonstration experiments.

4.5.1 The Robot Testbed

To implement and test our concepts we used the same testbed as in the previous chapter. The robot is a Pioneer 2-DX mobile robot, equipped with two rings of sonars (8 front and 8 rear), a SICK laser range-finder, a pan-tilt-zoom color camera, a gripper, and on-board computation on a PC104 stack.

4.5.2 The Behavior Set

The robot uses the behavior set described in Chapter 3 (Section 3.5.2.2) that contains the **PickUp**, **Drop** and **Track** behaviors.

4.5.3 Evaluation Criteria

Before presenting the experimental results we describe the evaluations criteria we used in order to analyze the results of our experiments, specifically the notions of success and failure.

The challenge we address is to enable a robot to *learn high-level task representations from human demonstrations*, relying on a behavior set already available to the robot. Within this framework, we define an experiment as **successful** iff all of the following properties hold true:

- the robot learns the correct task representation from the demonstration;
- the robot correctly reproduces the demonstration;
- the task performance finishes within a certain period of time (in the same and also in changed environments);
- the robot's reports on its reproduced demonstration (sequence and characteristics of demonstrated actions) and user observation of the robot's performance match and represent the demonstrated task.

Conversely, we characterize an experiment as having **failed** if either of the properties below holds true:

- the robot learns an incorrect representation of the demonstration;
- the time limit allocated for the task was exceeded;

4.5.4 Learning in Clean Environments

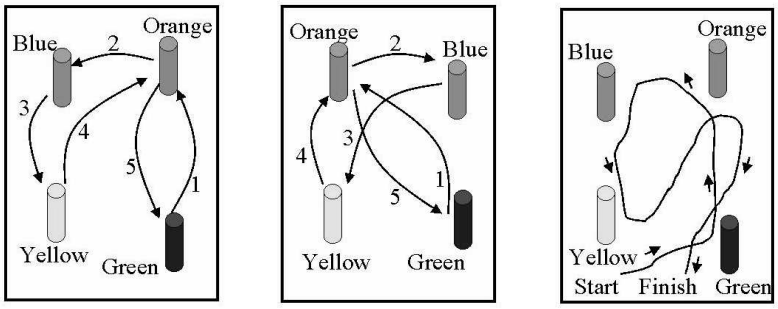
We performed three different experiments in a 4m x 6m arena, in which only the objects relevant to the tasks were present. During the demonstration phase a human teacher led the robot through the environment while the robot recorded its observations relative to the postconditions of its behaviors. The demonstrations included:

- teaching a robot to visit a number of cylindrical colored targets in a particular order;
- teaching a robot to slalom around cylindrical colored objects;
- teaching a robot to transport objects between a source and a destination location (marked by cylindrical colored objects).

We repeated these *teaching* experiments more than five times for each of the demonstrated tasks, to validate that the **behavior network construction** algorithm reliably constructs the same task representation for the same demonstrated task. Next, using the behavior networks constructed during the robot's observations, we performed experiments in which the robot reliably repeated the task it had been shown. We tested the robot in executing the task five times in the same environment as the one in the learning phase, and also five times in a changed environment. We present the details and the results for each of the tasks in the following sections.

4.5.4.1 Learning to Visit Targets in a Particular Order

The goal of this experiment was to teach the robot to reach a set of colored targets in the order indicated by the arrows in Figure 4.6(a). The robot's behavior set contains a **Tracking** behavior, parameterizable in terms of the colors of targets that are known to the robot. Therefore, during the demonstration phase, different instances of the same behavior produced output according to their parameters.



(a) Experimental setup (1)

(b) Experimental setup (2)

(c) Approximate robot trajectory

Figure 4.6: Experimental setup for the target visiting task

Figure 4.7 shows the behavior network the robot constructed as a result of the above demonstration. As expected, all the precondition-postcondition dependencies between behaviors in the network are **ordering** type constraints; this is evident in the robot's observation data presented in Figure 4.8. The intervals during which different behaviors have their postconditions met did not overlap and therefore the ordering is the only constraint that has to be imposed for this task representation. Five trials of the same demonstration

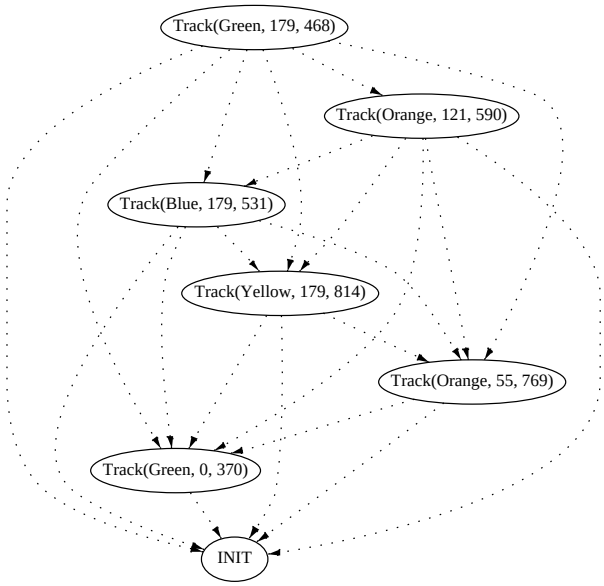
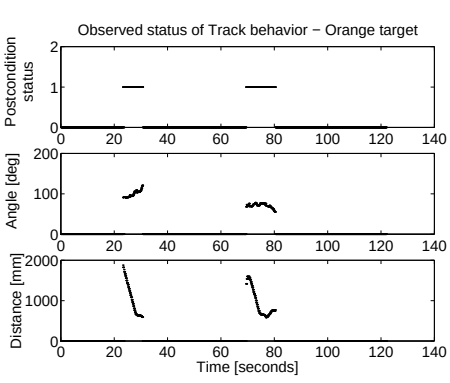


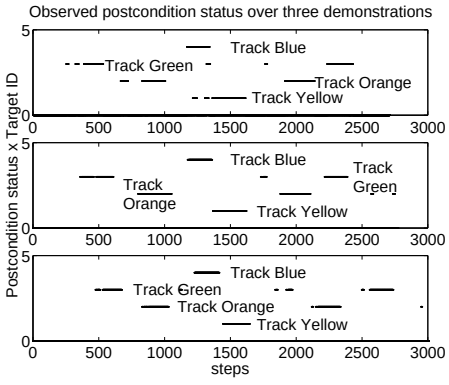
Figure 4.7: Task representation learned from the demonstration of the **Visit targets** task

were performed in order to verify the reliability of the network generation mechanism. All of the produced controllers were identical and validated that the robot learned the correct representation for this task.

To demonstrate the robustness of the control architecture and to demonstrate the advantage of learning high-level task representations, we performed the learned task in a different environment (Figure 4.6(b)). The robot correctly performed the task in this changed environment.



(a) Observed values of a Track behavior's parameters and the status of its postconditions



(b) Observed status of all the behaviors' postconditions during three different demonstrations

Figure 4.8: Observation data gathered during the demonstration of the **Visit targets** task

Figure 4.9 shows the time (averaged over five trials) at which the robot reached each of the targets it was supposed to visit (according to the demonstrations) in an environment identical to the one used in the demonstration phase. As can be seen from the behavior network controller, the precondition links enforce the correct order of behavior execution. Therefore, the robot will visit a target only after it knows that it has visited the ones that are predecessors to it. However, during execution the robot might pass by a

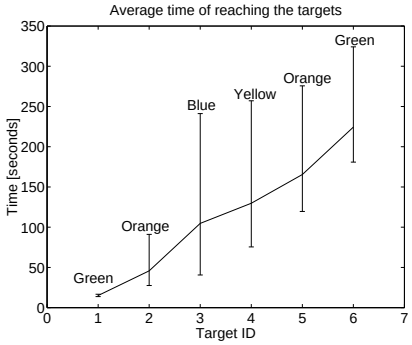


Figure 4.9: Averaged time of the robot's progress while performing the **Visit targets** task

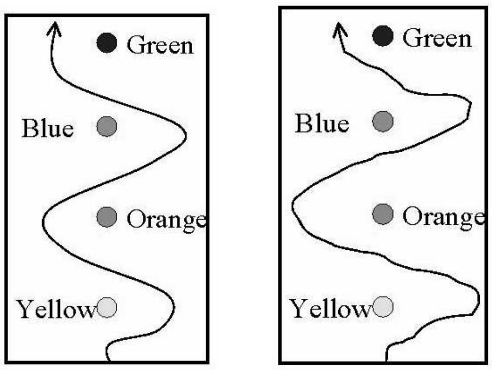
target that it was not supposed to visit at a given time. This is due to the fact that the physical targets are sufficiently distant from each other such that the robot could not see them directly from each other. Thus, the robot has to wander in search of the next target while incidentally passing by others. The robot does not consider the incidental visits as achievements of parts of its task, since it is not interested in them at that point of task execution. The robot performs the correct task as it is able to discern between an intended and an incidental visit to a target. All the intended visits occur in the same order as demonstrated by a human. Unintended visits, on the other hand, vary from trial to trial as a result of different paths the robot takes as it wanders in search of targets, and are not recorded by the robot in the task achievement process.

The robot's wandering in search of the next target is also the cause behind the large variance in traversal times. As is evident from the data, due to the randomness introduced by the robot's wandering behavior, it may take less time to visit all six targets in one trial than it does to visit just the first two in another trial.

In all experiments the robot met the time constraint, finishing the execution within 5 minutes, the allocated amount of time for this task.

4.5.4.2 Learning to Slalom

In this experiment, the goal was to teach a robot to slalom through four targets placed in a line, as shown in Figure 4.10(a). We changed the size of the arena to 2m x 6m for this task.



(a) Experimental setup

(b) Approximate robot trajectory

Figure 4.10: The **Slalom** task

During 8 different trials the robot learned the correct task representation as shown in the behavior network from Figure 4.11. For this case, we can observe that the relation between behaviors that track consecutive targets is of the **enabling** precondition type. This correctly represents the demonstration, since, due to the nature of the experiment and of the environment setup, the robot began to track a new target while still near the previous one.

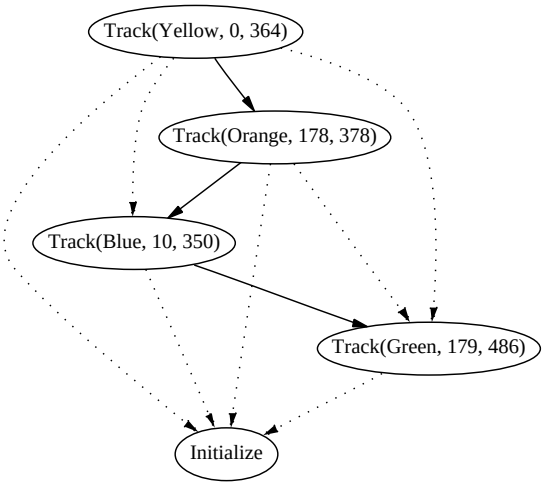
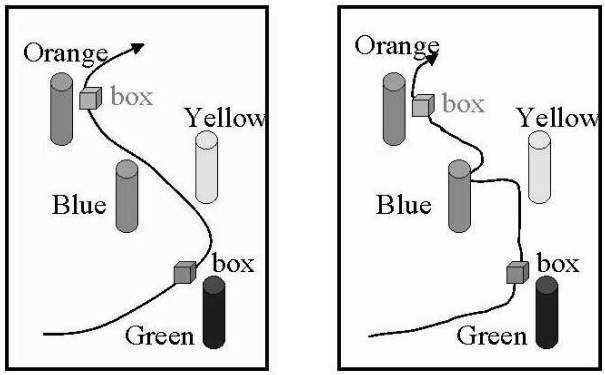


Figure 4.11: Task representation learned from the demonstration of the **Slalom** task

We performed 20 experiments, in which the robot correctly executed the slalom task in 85% of the cases. The failures were due to the robot's limited sensing capabilities and consisted of two types: 1) the robot, after passing one "gate," could not find the next one due to the limitations of its vision system; and 2) the robot, while searching for a gate, turned back towards the already visited gates. Figure 4.10(b) shows the approximate trajectory of the robot succesfully executing the slalom task on its own.

4.5.4.3 Learning to Traverse "Gates" and Transport Objects

The goal of this experiment was to extend the complexity and thus the challenge of learning the demonstrated tasks by adding object manipulation to the tasks and use the the robot's ability to pick up and drop objects.



(a) Traversing gates and moving objects

(b) Approximate trajectory of the robot

Figure 4.12: The **Object manipulation** task

The setup for this experiment is presented in Figure 4.12(a). Close to the green target there is a small orange box. In order to teach the robot that the task is to pick up the orange box placed near the green target (the source), the human led the robot to the box, and when sufficiently near it, placed the box between the robot's grippers. After leading the robot through the "gate" formed by the blue and yellow targets, when reaching the orange target (the destination), the human took the box from the robot's gripper. The learned behavior network representation is shown in Figure 4.13. Since the robot started the demonstration with

nothing in the gripper, the effects of the **Drop** behavior were met, and thus an instance of that behavior was added to the network. This ensures correct execution for the case when the robot might start the task while holding something: the first step would be to drop the object being carried.

During this experiment, all three types of behavior preconditions were detected: during the demonstration the robot is carrying an object for the entire time while going through the gate and tracking the destination target, the links between **PickUp** and the behavior corresponding to the actions above are **permanent** preconditions. **Enabling** precondition links appear between behaviors for which the postconditions are met during intervals that only temporarily overlap, and finally the **ordering** constraints enforce a topological order between behaviors, as it results from the demonstration process.

The ability to track targets within a [0, 180] degree range allows the robot to learn to execute the part of the task involving going through a gate naturally. This experience is mapped onto the robot's representation as follows: "track the yellow target until it is at 180 degrees (and 50cm) with respect to you, then track the blue target until it is at 0 degrees (and 40cm)." At execution time, since the robot is able to track both targets even after they disappeared from its visual field, the goals of the above **Track** behaviors were achieved with a smooth, natural trajectory of the robot passing through the gate. This demonstrates that the algorithm allows for learning higher-level behaviors, such as going through a door/gate using simpler behaviors already available to the robot.

Due to the increased complexity of the task demonstration, in 10% of the cases (out of more than 10 trials) the behavior network representations built by the robot were not completely accurate. The errors represented specialized versions of the correct representation, such as: **Track** the green target from a certain angle and distance, followed by the same **Track** behavior but with different parameters - when only the latter was in fact relevant.

Out of 10 trials, the robot correctly executed the task in 90% of the cases. The failures were all of the type involving exceeding the allocated amount of time for the task. This happened when the robot failed to pick up the box because it was too close to it and thus ended up pushing it without being able to perceive it. This failure results from the undesirable arrangement and range of the robot's sensors, not to

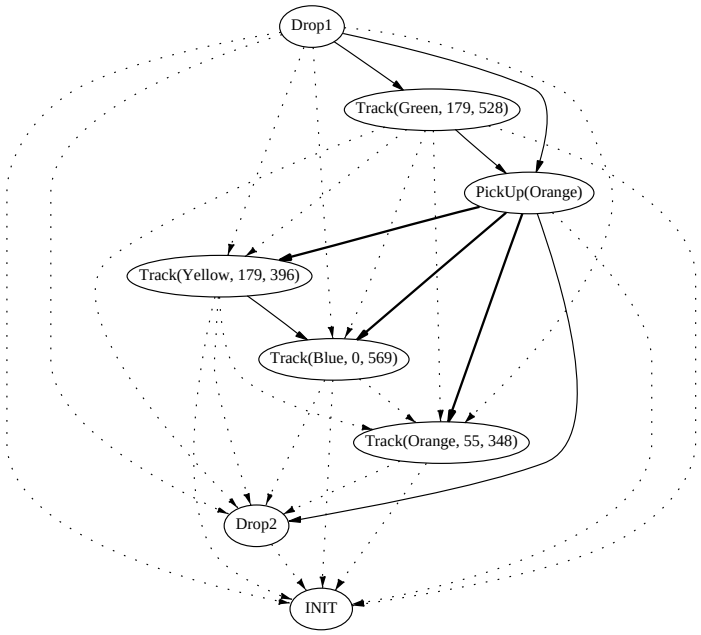


Figure 4.13: Task representation learned from the demonstration of the **Object manipulation** task

any algorithmic issues. Figure 4.14 shows the robot's progress during the execution of a successful task, specifically the intervals of time during which the postconditions of the behaviors in the network were true: the robot started by going to the green target (the source), then picked up the box, traversed the gate, and followed the orange target (the destination) where it finally dropped the box.

The results obtained from the above experiments demonstrate the effectiveness of using human demonstration combined with our behavior architecture as a mechanism for learning task representations. The

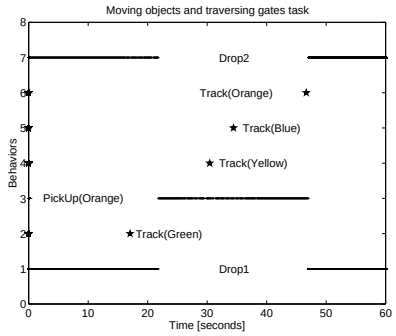


Figure 4.14: The robot's progress (achievement of behavior postconditions) while performing the **Object manipulation** task

approach we presented allows a robot to construct such representations from a single demonstration automatically. The summary of the experimental results is presented in Table 4.2. Furthermore, the tasks the robot is able to learn can embed arbitrarily long sequences of behaviors, which become encoded within the behavior network representation. Also, as is seen in the third experiment set, in the absence of a *GoThroughGate* behavior, the robot is able to represent that part of the task in a more concise manner then if the controller were to be designed by hand.

Table 4.2: Summary of the experimental results for learning in clean environments

Experiment name	Trials	Successes	
		Nr.	Percent
Six targets (learning)	5	5	100 %
Six targets (execution)	5	5	100 %
Slalom (learning)	8	8	100 %
Slalom (execution)	20	17	85 %
Object move (learning)	10	9	90 %
Object move (execution)	10	9	90 %

Analyzing the task representations the robot built during the experiments above, we observe the tendency toward over-specialization. The behavior networks the robot learned require that the execution go through all demonstrated steps of the task, even if some of them might not be relevant. Since, during the demonstration, there is no direct information from the human about what is or is not relevant, and since the robot learns the task representation from even a single demonstration, it assumes that everything that it notices about the environment is important and represents it accordingly.

As any one-shot learning system, our system learned a correct, but potentially overly specialized representation of the demonstrated task. Additional demonstrations of the same task would allow it to generalize at the level of the constructed behavior network, as presented in the next chapter. In the next section we address the problem of overspecialization by experimenting in cluttered environments and allowing the teacher to signal to the robot the saliency of particular events, or even objects. While this does not eliminate irrelevant environment state from being observed, it biases the robot to notice and (if capable) capture the key elements.

4.5.5 Learning in Environments With Distractors

The goal of the experiments presented in this section is to show the ability of the robots to learn from both human and robot teachers, in environments with distractor objects which are not relevant for the demonstrated tasks.

4.5.5.1 Learning from Human Teachers

The task to be learned by the robot is similar to the moving objects task from above (Figure 4.15(a)): pick up the orange box placed near the light green target (the source), go through the "gate" formed by the yellow and light orange target, drop the box at the dark green target (the destination) and then come back to the source target. The orange and the yellow targets at the left are distractors that should not be considered as part of the task. In order to teach the robot that it has to pick up the box, the human led the robot to it and then, when sufficiently near it, placed it between the robot's grippers. At the destination target, the teacher took the box from the robot's grippers. Moments in time signaled by the teacher as being relevant to the task are: giving the robot the box while close to the *light green* target, teacher reaching the *yellow* and *light orange* target, taking the box from the robot while at the *green* target, and teacher reaching the *light green* target in the end. Thus, although the robot observed that it had passed the *orange* and distant *yellow* targets during the demonstration, it did not include them in its task representation, since the teacher did not signal any relevance while being at them.

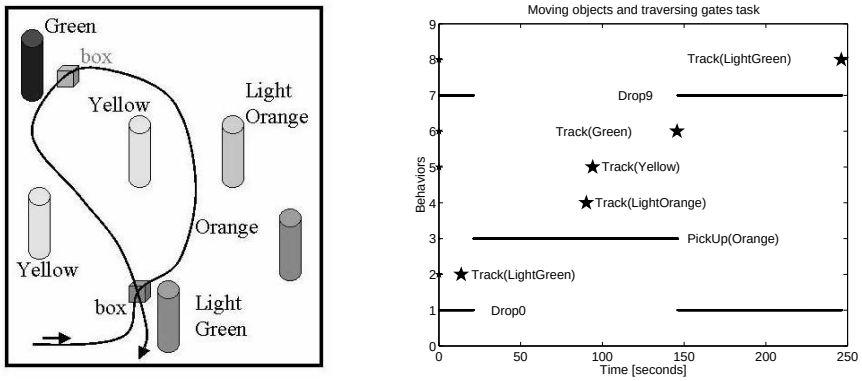
We performed 10 human-robot demonstration experiments to validate the performance of the **behavior network construction** algorithm. We then evaluated each learned representation both by inspecting it structurally and by having the robot perform it, to get physical validation that the robot learned the correct task. In 9 of the 10 experiments the robot learned a structurally correct representation (sequencing of the relevant behaviors) and also performed it correctly. In one case, although the structure of the behavior network was correct, the learned values of one of the behavior's parameters caused the robot to perform an

incorrect task (instead of going between two of the targets the robot went to them and then around). The learned behavior network representation of this task is presented in Figure 4.16.

For the 9 out of 10 successes we have recorded, the 95% confidence interval for the binomial distribution of the learning rate is [0.5552 0.9975], obtained using a Paulson-Camp-Pratt approximation (Blyth 1986) of the confidence limits.

As a base-case scenario, to demonstrate the reliability of the learned representation, we performed 10 trials, in which a robot repeatedly executed one of the learned representations of the above task. In 9 of the 10 cases the robot correctly completed the execution of the task. The only failure was due to a time-out in tracking the green target.

In Figure 4.15(b) we show the robot's progress during the execution of the task, more specifically the instants of time or the intervals during which the postconditions of the behaviors in the network were true.



(a) Environment

(b) Achievement of behavior postconditions

Figure 4.15: The **Object manipulation** task in environments with distractors

During this experiment, all three types of behavior preconditions were detected: during the demonstration the robot is carrying an object for the entire time while going through the "gate" and tracking the destination target, and thus the links between **PickUp** and the behavior corresponding to the actions above are **permanent** preconditions. **Enabling** precondition links appear between behaviors for which the postconditions are met during intervals that only temporarily overlap, and the **ordering** preconditions enforce a topological order between behaviors, as it results from the demonstration.

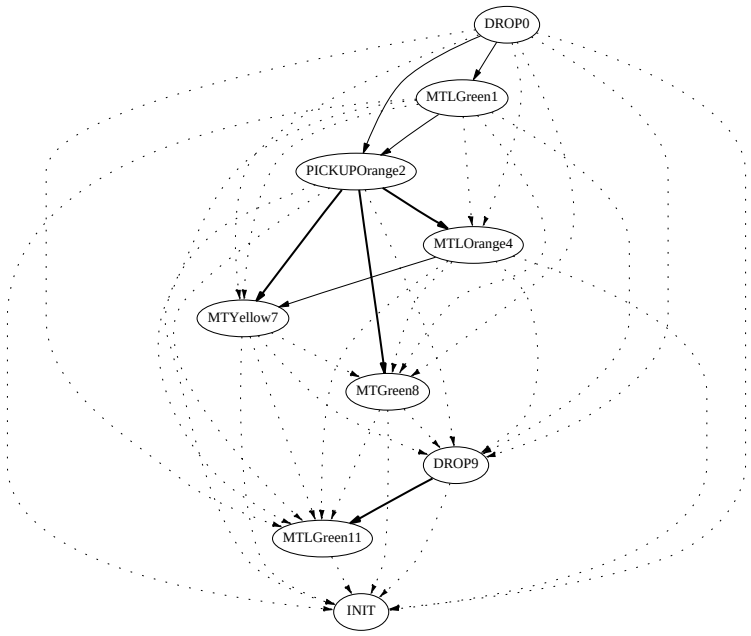


Figure 4.16: Task representation learned from human demonstration for the **Object manipulation** task

4.5.5.2 Learning from Robot Teachers

In this section we extend the problem of learning task representations to the case of learning from robot teachers. We are interested in determining the reliability of the information that is passed from robots to robots by means of teaching by demonstration.

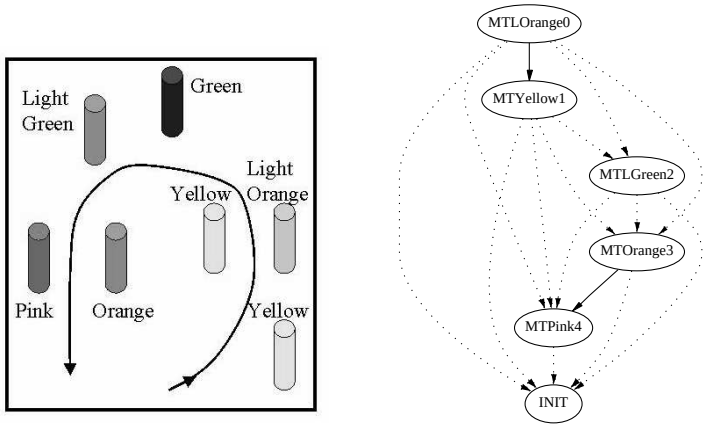
We examine the performance of learning a correct task representation transmitted from human to robot and then, in subsequent trials, from robot to another robot, in order to determine statistically (for our setup) the number of times the robots could correctly transfer the representations among themselves.

We define the task transfer limit (TTL) to be the number of successful transmissions of the same task from a teacher to a learner. A TTL of k means that the task was transmitted from the original demonstrator (usually a human) k times, until the failure point. This variable is expected to follow a geometric distribution, for which we determine the expected mean value and the confidence interval (Clemans 1959).

The task selected for the experiments is to go through a "gate" formed by the yellow and light-orange targets (Figure 4.17(a)), visit the light-green target, and come back through the pink and orange targets.

Two distractor targets (green at the top and yellow at the right bottom corner) were present in the environment, which the robots had to ignore during the learning and the execution process. Moments in time signaled by the teacher as being relevant to the task are reaching the yellow, light-orange, light-green, pink and orange targets.

We performed three human-led demonstrations, from which a learner robot correctly built the task representation each time. As a base case, to show that the performance of the robot does not degrade over time for the same task representation, we performed 10 trials in which a robot repeatedly executed the above task. In all 10 trials the robot correctly executed the task.



(a) Environment (b) Learned task representation

Figure 4.17: The **Visiting targets** experiment

Next, we performed 10 trials in which two robots, starting from a correctly learned task, switched roles in teaching each other the same task, each time using the information acquired at the previous step. Figure 4.17(b) presents the correct learned behavior network for this task. For each of the above trials we recorded the number of successful teaching experiences until the learning broke down. The maximum and minimum number of teaching experiments before learning failed were 6 and 2 respectively. The observed mean for the TTL obtained from the experiments is 2.5, with a 98% confidence interval of [1.4 8]. As the statistical evaluation shows, any information learned from a human can be further transfered to other

robots at least one more time in the worst case, despite the naive approach we have employed for the robot teacher.

We attribute the difference between the performance obtained in the case of a human versus a robot teacher to the quality of the demonstration: the human facilitates the learner's observations, whereas the robot teacher has to wonder around searching, due to its own limited sensing capabilities.

4.5.6 Learning from Non-Expert Users

We also performed experiments with non-expert users with different levels of expertise in robotics and computer science in general. Extended studies would be needed to compute statistically the performance of the proposed learning approach among naive users. Since for this work we were mainly interested in analyzing the type and amount of information necessary to train the robots for someone not familiar with the approach, we limited the experiments to two users.

The first user had a background in literature, science, and technology, but no computer programming or robotics experience. The second user had a background in computer science and robotics, but had not previously interacted with the robot.

Appendix A presents the manual that was provided to these non-expert users prior to their experiments with training the robot. The document is written in non-technical terms, in order to make the teaching approach accessible to various categories of users. By looking at the information described in the manual, we conclude that knowledge of the following key aspects is essential to enable non-expert users to teach a robot by demonstration:

- Robot sensors
- Robot actuators
- Features the robot can detect
- Robot skills (their description, sensors and actuators used, and how the robot performs them)

- Method for teaching the robot (following the teacher, responding to spoken instructions, and a description of all possible instructions along with their applicability conditions)

- A step-by-step description of a sample task.

To test the user's ability to teach the robot by demonstration, we selected the following task (performed in the environment presented in Figure 4.18): visit the **Light Green** target, pick up the **Orange** box, visit the **Yellow** target, visit the **Pink** target, drop the box near the pink target, and then visit the **LightOrange** and **Light Green** targets.

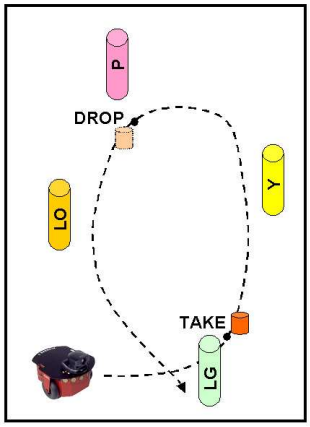


Figure 4.18: The **Object manipulation** task

With each of the users we performed 5 trials in which they demonstrated the above task to the robot. Table 4.3 presents the number of successes for each user. By examining the results obtained from these experiences, we observed that the erroneous trials were determined by the following factors:

Limitations of the vision-based, color blob detection mechanism:

- the robot perceives false positives, such as the presence of a target of a color that was not even present in the environment.

- the robot misidentifies targets, and assigns different colors to them.

Limitations of the user's demonstration:

- difficulty in having the robot continuously follow the teacher during the demonstration. The environment contained numerous distractors (in particular, a wall-sized curtain that has a color similar to the one assigned for recognizing the teacher). For an experimented teacher, already aware of these distractors, it is easier to lead the robot such that they would not interfere with the learning process. However, the naive users also learned to guide the robot away from that wall and also to capture the robot's "attention" and have it follow them again.

- inaccurate estimation of when and from what directions the robot is able to detect the colored targets from the environment. As a result, during some demonstrations the robot did not observe the visits to some of the targets.

Table 4.3: Experimental results for learning from non-expert users

Users	Trials	Successes
User 1	5	1
User 2	5	3

Although the performance of learning from non-experienced users is lower than that of a user that has been working with the robot for a longer time, the experiments showed that it is possible for these users to learn how to guide the robot through the task even in a small number of trials, and with no other prior training.

4.6 Discussion

Based on the current set of behaviors available to the robot and the current vocabulary for instruction, the robot is able to learn any task belonging to the set described in Chapter 3 (Section 3.6).

The expressiveness of the learned tasks could be further increased by introducing new words in the robot instruction vocabulary. For example, commands such as **repeat-until** could allow the robot to learn

repetitive tasks, or repetitive fragments of a more elaborate task. This capability could be easily incorporated in the learning algorithm, as the control architecture supports the "reinstantiation" of networks after their completion and thus the ability to perform repetitive tasks.

In addition, if a robot is provided with skills that measure translational and rotational velocities (e.g., **Turn(Angle)**, **MoveForward(Distance)**), the proposed framework would also allow learning of trajectories. Behaviors considering time taken during the demonstration, in conjunction with instructions such as **do-until**, would enable learning of tasks such as *Do X for T time*, discussed in Section 4.3.

4.7 Summary

This chapter presented an application of the *Hierarchical Abstract Behavior Based Architecture* described in Chapter 3 to the problem of robot teaching by demonstration. It described an on-line approach that allows a robot to learn task representations from its own experiences of interacting both with a human and a robot teacher. The robot relates the observed changes in the environment with its own internal behaviors and learns from only one trial, even in the presence of distractor objects. The chapter also demonstrated the correctness of the algorithm formally, and then validated its effectiveness experimentally.

Chapter 5

Learning through Generalization from Multiple Examples

This chapter describes a method for generalization of task representations from a small number of teacher-provided demonstrations. It starts by presenting the generalization problem in the context of graph-like task representations and then it presents the approach for constructing a generalized task representation from several demonstrations, by using a dynamic programming approach. Next it provides a discussion regarding behavior preconditions and task execution in the presence of alternate paths of execution, which are introduced through the generalization process. Finally, experiments validating the approach are presented.

One of the capabilities that allows humans to learn effectively is the ability to generalize over multiple observed demonstrations. For a teaching by demonstration approach to be efficient, it is essential that the robot learn from as few demonstrations as possible. A robot house keeper is of little use if the owner must show it hundreds of times how to bring in the mail. Therefore, statistical learning techniques, which rely on a large number of training examples, are not appropriate for our desired approach. Also, in robotics, the existing methods for generalization from demonstrated examples are largely based on function approximation (Kaiser 1997). However, in our case, the problem consists of generalizing across graph-like representations of the tasks encoded as behavior networks.

The important aspect of the task that needs to be considered across the multiple experiences is the ordering in which behaviors are present (and therefore executed) in the acyclic graph representation of a

behavior network. Therefore, we choose to solve this problem at the level of the *topological* task structure. Given the directed acyclic graph (DAG)-like structure of the behavior network representation of the robot tasks, we consider the *topological* representation of such a network to be a linked list of behaviors, obtained by applying a *topological sort* on the behavior network graph. By using the topological form of the networks as training examples for our domain, the problem of generalization from multiple demonstrations is equivalent to inferring a regular expression (Finite State Automaton (FSA) representation) from a set of given sample *words* (Figure 5.1(a)). In this analogy, each symbol in a given *word* corresponds to a behavior in a topological representation.

Unfortunately, applying standard methods for regular expression inference, such as the K-TSI Inference Algorithm (Garcia & Vidal 1990) or Morphic Generator Grammatical Inference (MGGI) (P. Garcia & Casacuberta 1987), to this generalization problem yields results that are too complex (in terms of the obtained FSA representations) even for very simple examples. This is due to the fact that these methods assume that all the training examples are correct and they try to fit them as well as possible. For our robot domain, in which the inaccuracies in the training examples (learning irrelevant steps or omission of steps that are relevant) are exactly the problem we need to solve, these methods are therefore not a good choice.

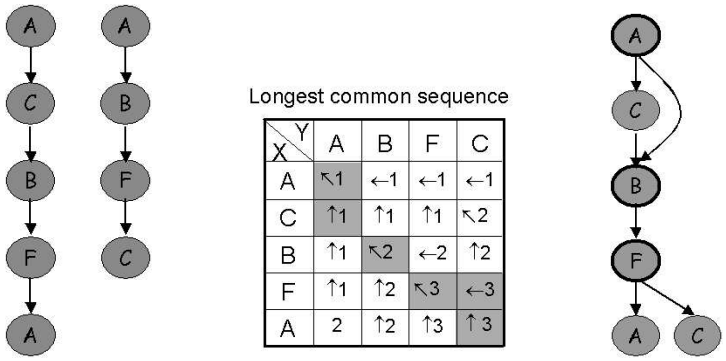
5.1 Constructing the Generalized Task Representation

5.1.1 Computing the Similarity Across Tasks

The reason we are interesting in giving a robot the ability to generalize over multiple teaching experiences is that its limited sensing capabilities, the quality of the teacher's demonstration, and the particulars of the environment generally prevent the robot from correctly learning a task from only one trial. The two main inaccuracies that occur in the learning process are *learning irrelevant steps* (false positives) and *omission of steps that are relevant* (false negatives).

Our approach for generalization is to build a task representation that encodes the specifics of each input example, but most importantly that points out the parts that are common. As a measure of similarity

we consider the longest list of common nodes between the *topological* forms of the sample tasks. Based on this information we further construct a *generalized topology* in which nodes that are common to both tasks will be merged, while the others will appear as alternate paths in the graph. For example, for the examples presented in Figure 5.1(a), behaviors A, B and F constitute the longest subsequence of common nodes. The representation resulted after "merging" the initial graphs at their common nodes is shown in Figure 5.1(c).

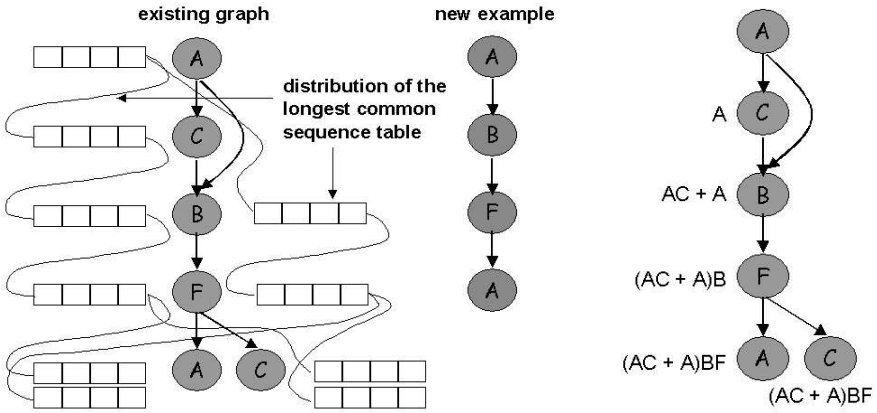


(a) (b) (c)

Figure 5.1: Generalization across multiple demonstrations. (a) Training examples; (b) Longest common sequence table; (c) Generalized topology

In order to find the similarity between the two inputs we rely on a standard dynamic programming approach for computing the *longest common subsequence (LCS)* (Cormen, Leiserson & Rivest 1990) between two sequences of symbols. If $X = < x_1, x_2, ..., x_m >$ and $Y = < y_1, y_2, ..., y_n >$ are two sequences of symbols, and the **prefix** of a sequence is defined as $X_i = < x_1, x_2, ..., x_i >$, the algorithm computes a longest common subsequence table (Figure 5.1(b)) that encodes in each element $tbl[i, j]$: i) the length of the longest common subsequence of the sequences X_i and Y_j, and ii) a pointer to the table entry corresponding to the optimal subproblem solution chosen when computing $tbl[i, j]$. The right bottom element of the table contains the length of the LCS for the entire sequences X and Y. The running time of the algorithm is $O(mn)$, with m and n being the lengths of the two sequences X and Y, typically small for our domain.

We obtain the generalized topology by traversing the LCS table starting in the right bottom corner and following the arrows: an "↖" arrow indicates nodes that are common to both training examples and that are merged, while "←" and "↑" arrows indicate nodes that belong to only one sequence. These latter cases are added as alternate paths of execution (Figure 5.1(c)).



(a) (b)

Figure 5.2: Incorporating new demonstrations: (a) Efficient computation of a generalized topology; (b) Generalized topology

The generalization process is incremental, meaning that each newly acquired experience is *incorporated* into the existing topological task representation. If this topology is already the result of a previous generalization, and has the form of a DAG with alternate paths of execution (Figure 5.1(c)), in a simple approach, incorporating a new demonstration into the existing structure would amount to running the same algorithm described above between the new example and all the possible paths of that graph. Since a LCS table encodes the common subsequences for all possible subproblems ($LCS(X_i, Y_j)$, with $i = [1, m]$ and $j = [1, n]$), we can efficiently apply this algorithm without having to compute a different table for each path of the graph. For this, we construct a structure that contains the LCS table in the form of a linked list of rows computed as the ones above. Within this structure, each node has an associated row for each different possible path from the root(s) to that node (Figure 5.2(a)). Each of these rows points to the row associated to the parent node on the corresponding path. As a result, each path in the graph has associated a linked list of rows that encodes the measure of similarity between that path and the new given example.

To compute the generalized topology from this structure, we traverse the list that embeds the longest of the possible subsequences, similarly with traversing the LCS table above. This process is efficient in terms of both computational time and space, as different paths "share" the parts of the table that are common. For our example, the obtained generalized topology is not changed by incorporating the new example, as shown in Figure 5.2(b).

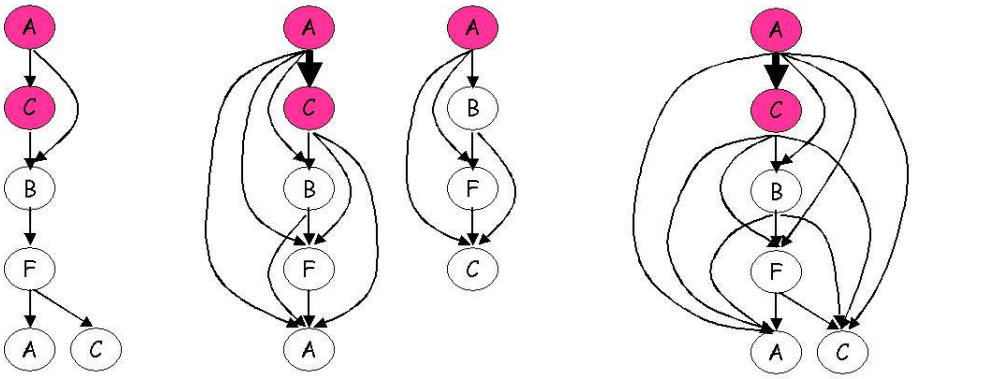
The generalization between multiple learned tasks (encoded as behavior networks) is performed at the level of their topological representations and provides a *generalized topology*. The *generalized behavior network* associated with this topology, is constructed from the underlying temporal dependencies between behaviors in the networks that participated in the generalization. This process is described in the next section.

5.1.2 Updating the Generalized Network Dependencies

In order to ensure proper behavior sequencing we need to transfer the temporal dependencies between behaviors to the generalized behavior network.

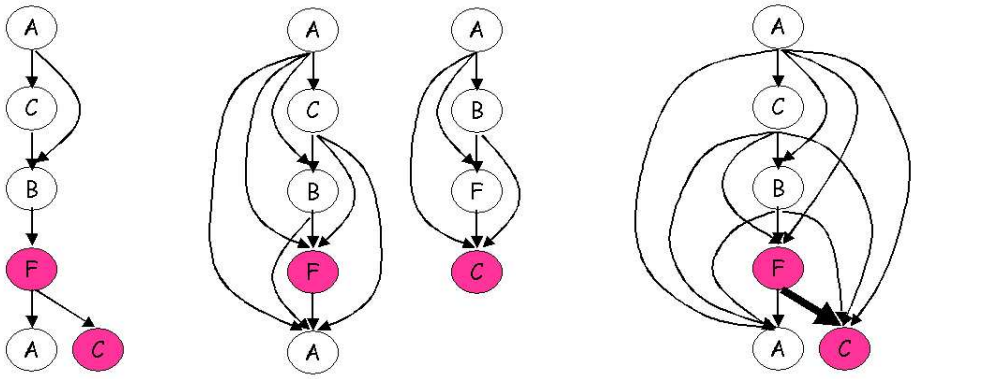
While computing the behavior network dependencies between any two behaviors X and Y belonging to the *generalized topology*, there are three possible situations that may occur:

- X and Y do not belong to the LCS, but are both part of the same task, such as behaviors A and C in the left network in Figure 5.3(b). In this case the link between behaviors X and Y is only present in one of the networks (the one on the left), and therefore the same link will be used in the generalized network.
- X and Y are each part of a different underlying task, and X is a predecessor of Y in the generalized topological representation, such as behaviors F (left network) and C (right network) in Figure 5.4(b). In this case, since there is no dependency between these behaviors in any of the existing networks, but since the precedence of X over Y needs to be enforced, in the generalized network, an **ordering** constraint from X to Y will be added.



(a) Topology (b) Component Networks (c) Generalized Behavior Network

Figure 5.3: Computing dependencies between behaviors: case 1.



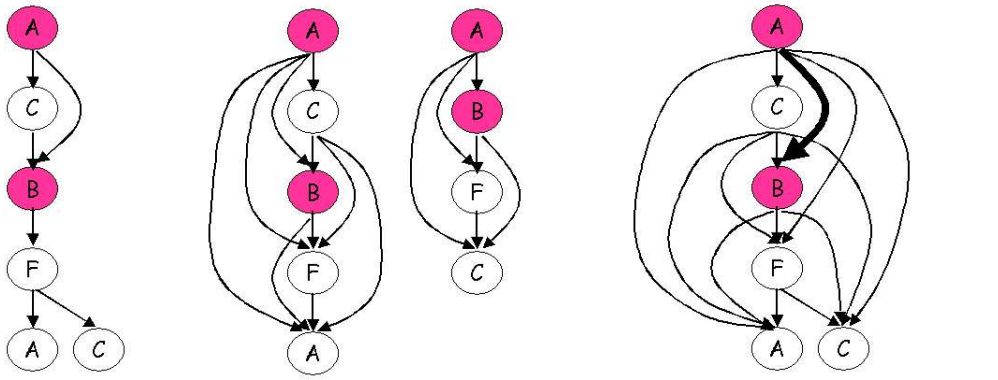
(a) Topology (b) Component Networks (c) Generalized Behavior Network

Figure 5.4: Computing dependencies between behaviors: case 2.

- Both X and Y are part of the LCS, such as behaviors A and B in Figure 5.5(b). In this case, there is a dependency between the behaviors in both behavior networks, and if these dependencies have different types, a decision needs to be made on which one to consider. The solution chosen for this case is to take the type that represent the least restrictive constraint, as shown in Figure 5.6.

5.2 Alternate Paths of Execution

During generalization, the topological representation of each new demonstration is compared with the existing topological structure, while computing their similarity in the form of their longest common sequence, as described above. To build the generalized topology, common nodes are merged while the rest



(a) Topology (b) Component Networks (c) Generalized Behavior Network

Figure 5.5: Computing dependencies between behaviors: case 3.

Net2 \ Net1	Ordering	Enabling	Permanent
Ordering	Ordering	Ordering	Ordering
Enabling	Ordering	Enabling	Enabling
Permanent	Ordering	Enabling	Permanent

Figure 5.6: Updating temporal dependencies between behaviors

appear as alternate execution paths. Due to the generalization, the following types of alternative paths can be obtained:

- both paths contain actual behavior(s). For example, Figure 5.7 encodes the fact that performing behaviors A or C after behavior F is acceptable for the task. For such alternate paths the robot chooses opportunistically between them, as induced by the state of the environment.
- one path is a direct link to the end of the other alternate sequence. In Figure 5.7, there is a direct link from A to B, bypassing the behavior C. For such paths, the robot will automatically chose the direct path, shortcutting the alternate sequence. These unattainable paths could be removed from the graph, but we are keeping them as we can envision extensions in which teacher feedback could eliminate such direct links ("marking" as wrong certain transitions from one step to another).

5.3 Computing Behavior Preconditions in Generalized Representations

In a simple behavior network (whose topology is a chain of behaviors and not a DAG), the task-dependent preconditions for a given behavior (the ones that depend on the execution of its predecessors) have the form of a conjunction of the status of all its predecessor behaviors.

In a generalized topology, since multiple alternate paths to a particular behavior can exist, the preconditions are encoded as combinations of conjunctions and disjunctions of the different paths. Thus, computing the preconditions for each behavior becomes equivalent to computing the regular expression from a FSA representation (Figure 5.7).

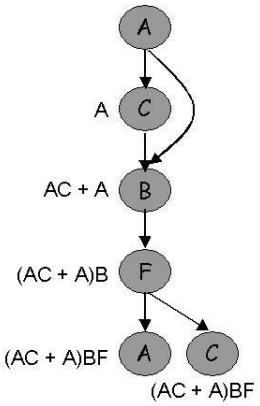


Figure 5.7: Computing behavior preconditions in a topological representation

For example, evaluating the preconditions for behavior F means checking that either the goals of A and C and B or those of just A and B are or have been true in accordance with the types of dependencies between them and behavior F, as given by the generalized behavior network computed above.

To summarize, the generalization process between two *behavior networks* is performed at the level of their *topological* representations, resulting in a *generalized topological structure*. The temporal dependencies between behaviors in the generalized task representation, which are encoded in a corresponding *behavior network*, are computed from the information contained in the underlying *behavior networks* involved in the generalization.

5.4 Experimental Validation

5.4.1 The Robot Testbed

To implement and test our concepts we used the same testbed as in the previous chapters. The robot is a Pioneer 2-DX mobile robot, equipped with two rings of sonars (8 front and 8 rear), a SICK laser range-finder, a pan-tilt-zoom color camera, a gripper, and on-board computation on a PC104 stack.

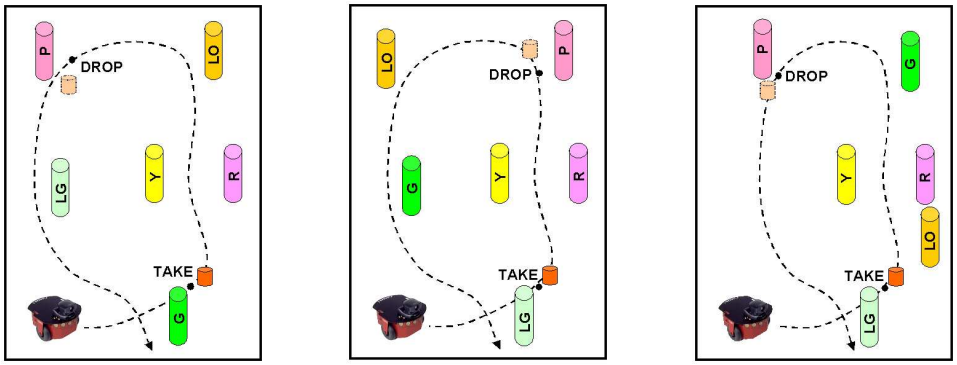
5.4.2 The Behavior Set

The robot used the behavior set described in Chapter 3 (Section 3.5.2.2), that contains the **PickUp**, **Drop** and **Track** behaviors.

5.4.3 Generalization from Three Given Examples

We validated the *generalization* abilities of the robot by teaching it an object transport task in three consecutive demonstrations, performed in different environmental setups (Figure 5.8), and purposely designed to contain incorrect steps and inconsistencies. The next chapter shows how already learned/generalized tasks can be further refined through *practice* and *feedback*. As discussed above, giving **"HERE"** cues during the demonstrations does not help the robot detect the relevant parts of the task perfectly. In these three training experiments we included all of the robot's observations into the learned task representations, solely for the purpose of demonstrating the *generalization* technique, to simulate that the robot was not able to discern the relevant aspects despite the teacher's instructions. The importance of such messages, however, will be shown in the *practice-feedback* experiments presented in the next chapter.

The environment consisted of a set of cylindrical targets, in colors that the robot is able to perceive. The teacher led the robot around these targets, while also instructing it when it had to pick up or drop a small orange box. The task to be learned was as follows: go to either the **Green (G)** or the **Light Green (LG)** targets, then pick up an **Orange (O)** box, go between the **Yellow (Y)** and **Red (R)** targets, go to the



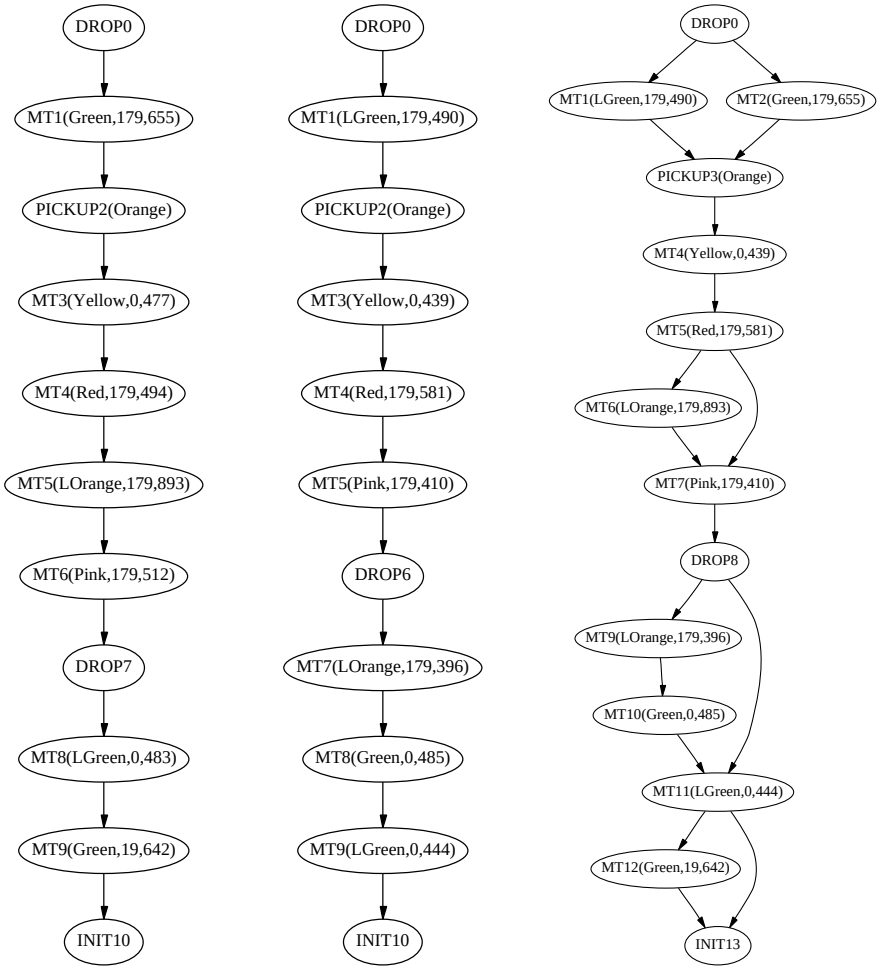
(a) First demonstration (b) Second demonstration (c) Third demonstration

Figure 5.8: Structure of the environment and course of demonstration

Pink (P) target, drop the box there, then go to the **Light Orange (LO)** target and come back to the target **Light Green**.

The courses of the three demonstrations above show that none of them corresponds exactly to the target task. Besides containing unnecessary steps (such as a final visit to a **Green** target in the first trial), these training runs also contain inconsistencies, such as the visits to the **Light Orange** target, which happened at various stages during the demonstrations. Figures 5.9 and 5.10 present the task representations obtained after each "learning $\rightarrow$ generalization" process. For all these experiments, in order to validate the correctness of the learned/generalized representations, we had the robot execute them in the same environment in which they had been demonstrated after each teaching experience. In all cases the robot performed the task correctly for the particular stage of the generalization process. Also, in order to demonstrate the robustness of our architecture to changing environments and the advantages of learning high-level representations of tasks, we had the robot execute the last generalized network (Figure 5.10(b)) in a different environment than the three presented before (Figure 5.10(c)). The robot correctly executed that task in the new setup.

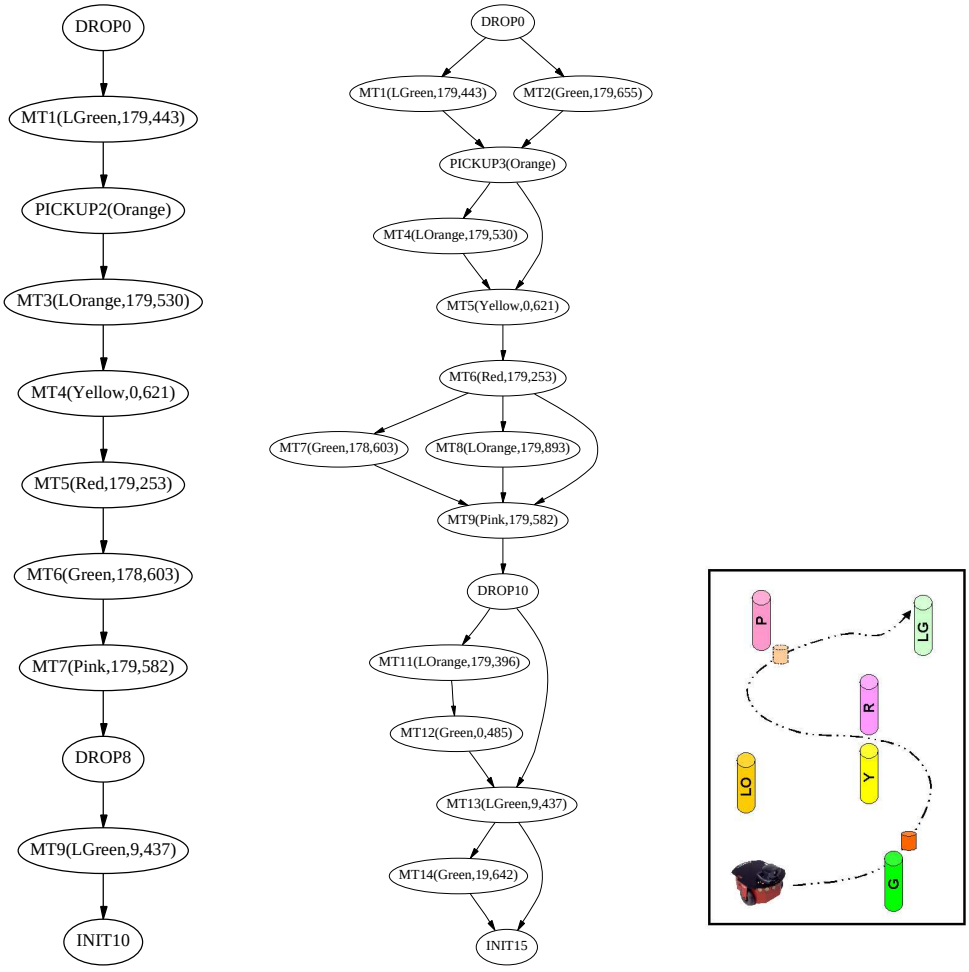
The representation that the robot has about the task at the end of these experiments contains most of the steps that would be required for a correct performance. More specifically, the robot captured all the steps that have been consistently observed throughout all three demonstrations. Considering the interpretation that the robot gives to the alternate paths of execution, from the topology obtained after the last step of



(a) First demonstration (b) Second demonstration (c) Generalized task

Figure 5.9: Evolution of the task representation over two successive demonstrations

generalization, the robot learned to perform the following task: go to either the **Green (G)** or the **Light Green (LG)** targets, then pick up an **Orange (O)** box, go between the **Yellow (Y)** and **Red (R)** targets, go to the **Pink (P)** target, drop the box there, and then come back to the **Light Green** target. The step of visiting the **Light Orange** target is missing: although captured correctly during the second demonstration, the visit to this target has been demonstrated inconsistently throughout the sequence of demonstrations.



(a) Third demonstration (b) Generalized task (c) New environment

Figure 5.10: Evolution of the task representation after the third demonstrations

5.5 Discussion

The generalization method uses an acyclic graph to compactly encode the actual "rules" from the multiple demonstrations. The resulting generalized task representation captures the main structure of the task while at the same time dealing with the irrelevant and inconsistent parts of the demonstrations: both of these situations are captured as becoming a part of bypassed alternate paths which will never be executed. While it is desired that the irrelevant actions are thus pruned, the steps demonstrated inconsistently which are still necessary would have to be included by different means. These results are to be expected: *generalization*

alone, when provided with inconsistent examples, is not enough for learning a correct representation. The next chapter shows how *practice* and *teacher feedback* can be used for solving this problem.

5.6 Summary

The ability to generalize from multiple experiences or examples is essential for any learning by demonstration approach, as the robot's limited sensing capabilities and the correctness of the teacher's performance may negatively influence the learning process. Existing generalization methods have mostly employed statistical approaches that rely on a large number of examples, and that have focused on non-symbolic representations, such as movement trajectories. This chapter presented an iterative approach to generalization that uses a small number of examples to construct a symbolic, graph-like task representation. The described experiments validate the proposed approach.

Chapter 6

Improving Learning through Practice and Teacher Feedback

This chapter presents an approach that allows a robot to improve the accuracy of a learned task, by practicing it under a teacher's supervision. The chapter describes the two main types of inaccuracies that can occur during a process of teaching by demonstration: learning of non-relevant steps, and failure to include steps that are pertinent to the task. For each of these situations it describes how are the problems be detected, and also by what means a teacher provides the robot with *corrective feedback*. Finally, the chapter describes two sets of experiments in which robots refine previously learned tasks using the feedback given by the teacher.

Generalization over several training examples helps in identifying the steps that were observed most often and are thus most likely a part of the task. However, repeated observations of irrelevant steps may inadvertently bias the learner toward including them in the representation. Also, limitations in the sensing capabilities of robots and particular structures in the environment may prevent the robot from observing steps that are relevant.

To enable a robot to learn correctly a task in these conditions, we take an approach similar to what people do when teaching each other. After one, or several, demonstrations of a particular task, the robot is allowed to *practice* the task it had learned under the teacher's supervision. During this step, the robot performs the task, while the teacher observes it and detects any errors in execution. These errors are signaled

by the user at the moment when they occur, through appropriate feedback cues, in our implementation given as spoken commands. The robot uses these messages to update and correct its representation of the task. This is a natural and accessible approach for refining the robot's learned task representation, which does not require expertise or knowledge regarding the robot's internal model of the task. As a result, the *practice* runs allow the teacher to observe the execution of the robot and to point more accurately to where problems occurred.

6.1 Removing Unnecessary Learned Steps

One of the problems that may occur during a teaching by demonstration process is that the robot may gather irrelevant observations, and include them as pertinent aspects for the task at hand. These are false positives. For example, while teaching the robot to visit a particular target the robot may pass close to another one and infer that it should also be visited. The robot considers these irrelevant steps as valid and therefore performs them during its practice experience. Since the teacher is aware of the robot's capabilities and also of how it performs them, having the robot practice what it had learned allows for easy identification of the steps that are not relevant for the task.

There are two types of irrelevant observations:

1. task steps that are not a part of the task at all. An example is the learned sequence A, B, C, D, A for an intended task A, B, D, A. In this case C is irrelevant, and there is no other occurrence of C in the correct task.

2. task steps that are not a part of the task, but for which there exist identical steps which are a part of the task. An example is the task sequence $A, B, C, B, A,$ for an intended task A, C, B, A. In this case the first occurrence of B is irrelevant, and the second is a relevant step.

To correct the first type of error, the teacher communicates to the robot within a short time after detecting the problem, by giving a **"BAD"** command. The teacher may also signal before allowing the robot

to finish the execution of the unnecessary step, if from the robot's exhibited behavior it is clear what the robot's intention is. **"BAD"** indicates that the behavior that the robot is currently executing, or the one that has just been finished (assuming a response time of 10 seconds) is irrelevant for the task. If the cue from the teacher comes within 10 seconds after the completion of a behavior, the message is considered to relate to this behavior, even if, at the moment of receiving the feedback, another behavior was active. This is based on the assumption that the teacher could not have reacted so quickly to the activation and execution of the second behavior. The robot labels the indicated behavior as irrelevant and removes it from the task representation using standard graph manipulation techniques. This process is represented in Figure 6.1 below.

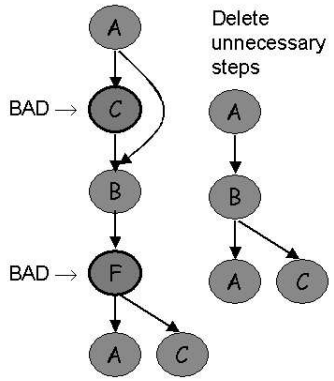


Figure 6.1: Using feedback to eliminate unnecessary steps

To detect and correct the second type of error, the teacher should allow the robot to complete the task, and provide feedback in a subsequent practice run. By looking at the robot's performance only as far as the occurrence of the incorrect step, the cause of the problem cannot be decided precisely. This is due to the fact that the same sequence of observed events can be generated by two different types of errors in the robot's task representation: either the robot learned an irrelevant step (the case analyzed here), or the robot failed to learn a sequence of steps preceding a relevant step similar in type to the irrelevant one (Section 6.2).

For example, if a correct task is A, C, B, A, and the robot performs B after A, this problem may have been generated by two causes: either the robot learned the sequence A, B, C, B, A, where the first

occurrence of B is not relevant, or the robot learned A, B, A, where the relevant step C is missing. To clear this ambiguity and to identify the correct cause of the problem, it is good for the teacher to wait until the end of the robot's task performance and to provide feedback in a subsequent practice run, using the same command (**"BAD"**) and approach as described above.

However, the teacher may still provide feedback during the first performance of the task. In this case, making the wrong assumption on the cause of the observed error results in a more complex process for refining a robot's learned task representation through feedback.

6.2 Incorporating Missing Steps

Another error that may occur during teaching a robot by demonstration is that, due to limited sensing capabilities, the robot may miss steps that are relevant for the task. These are false negatives. These steps are detected during a robot's practice experience, by observing that the robot skips performing them. As discussed in the previous section, this problem is detected by allowing the robot to perform a complete practice run, to eliminate ambiguities related to irrelevant task steps.

To correct this error, the teacher can intervene using a **"COME"** and a **"GO"** command. "COME" makes the robot enter into the *learning* mode described in Chapter 4; it starts following the teacher who demonstrates again the missing part of the task. When these parts have been demonstrated, the teacher ends the demonstration with "GO," after which the robot continues executing the remaining part of the task. The robot incorporates the newly learned steps into the task representation and includes the newly demonstrated steps in its next execution (Figure 6.2). The arrow next to behavior B means that the "NEW" message was received while the behavior was active, or shortly after the behavior finished its execution (in our case this time interval is of 10 seconds). By providing feedback at this particular time, the teacher implies that the steps to be added should have happened before B's execution, as represented in the final task structure.

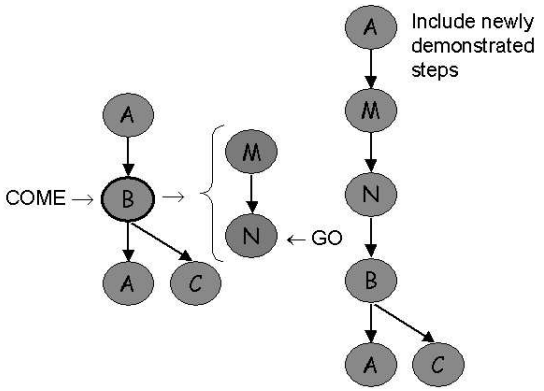


Figure 6.2: Using feedback to incorporate missing observations

Both types of instructions (for correcting missing or unnecessary steps) can be applied at any time, during the same practice runs, and as many times as the teacher considers it necessary. The ability to use these cues to refine previously learned tasks by a robot is validated experimentally in the next section.

6.3 Experimental Validation

To validate the key features of our proposed approach we performed two sets of robot experiments, which are described next.

6.3.1 The Robot Testbed

To implement and test our concepts we used the same testbed as in the previous chapters. The robot is a Pioneer 2-DX mobile robot, equipped with two rings of sonars (8 front and 8 rear), a SICK laser range-finder, a pan-tilt-zoom color camera, a gripper, and on-board computation on a PC104 stack.

6.3.2 The Behavior Set

The robot uses the behavior set described in Chapter 3 (Section 3.5.2.2), that contains the **PickUp**, **Drop** and **Track** behaviors.

6.3.3 Task Refinement after Demonstration and Generalization

In the first experiment the experienced user employed the practice-feedback approach after having given multiple demonstrations (in our case 3) of the same task. This example is based on the same task presented in the previous chapter, in Section 5.4 and follows on its results. The task to be learned was as follows: go to either the **Green (G)** or the **Light Green (LG)** targets, then pick up an **Orange (O)** box, go between the **Yellow (Y)** and **Red (R)** targets, go to the **Pink (P)** target, drop the box there, then go to the **Light Orange (LO)** target and come back to the target **Light Green**. The teacher performed three demonstrations of this task in three different environments consisting of a set of cylindrical targets, in colors that the robot is able to perceive.

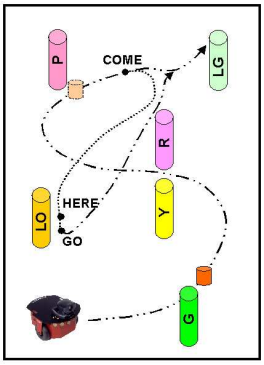


Figure 6.3: Environment for feedback given after three demonstrations

Generalization over the several given examples, allowed the robot to build an improved representation of the learned task. However, as discussed in the previous chapter, the generalized network did not represent the target task intended by the user. The missing part was a visit to the **Light Orange** target, which should have happened right after dropping the box and before going to the **Light Green** target. Since the generalization process already built the remaining of the task structure, simple feedback during a robot practice run was enough for refining it to the desired structure. We performed the practice run in the environment presented in Figure 6.3. The figure also shows the robot's trajectory and (dotted) the teacher's intervention. After dropping the box at the destination **Pink** target, the robot started going toward

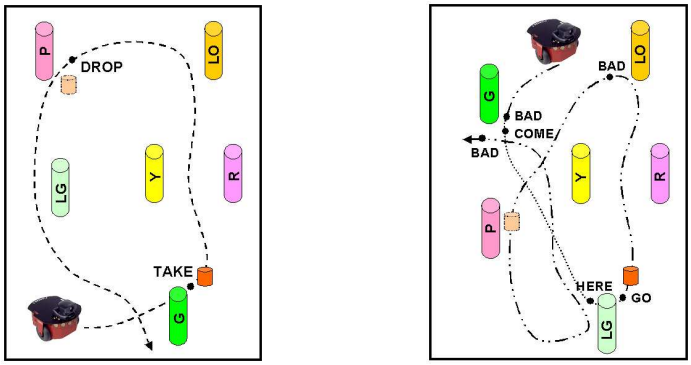
the **Light Green** target, which was a sign that the robot skipped the visit to the **Light Orange** target. Observing this tendency, the teacher intervened by using the **"COME"** command: the robot switched to the learning mode, and followed the teacher, who lead it to the **Light Orange** target it had previously failed to observe. The use of the **"HERE"** feedback cues during this learning stage was essential, as the robot also passed by and detected other targets (**Pink** and **Yellow**) while following the teacher, and which were thus ignored. After demonstrating the missing step, the teacher signaled the end of the "learning" intervention using the **"GO"** command, and the robot continued and finished the task by going to the remaining **Light Green** target. Figure 6.5(a) shows the structure of the task after this practice run. The newly added steps are marked on the graph: they also include a **Drop** behavior, as the robot had nothing in the gripper at the point of the demonstration and therefore the goals of this behavior were also detected as true. At the time of the execution, the existence of this behavior had no influence, since the robot had already dropped the object at that point.

6.3.4 Task Refinement after Demonstration Only

The second experiment demonstrates how a similar transport task can be learned and refined by providing feedback directly after the initial demonstration. In this case, since the robot does not have the opportunity to improve its task representation through additional examples, to correct the potential inaccuracies during practice involves more complex teacher feedback.

This example is based on the task and the results obtained after the first demonstration presented in the previous chapter. The experiment was performed in the environment presented in Figure 6.4(a). At the end of the demonstration, the robot learned the representation shown in Figure 5.10(c) (in Chapter 5), which can be summarized as: visit the **Green** target, pick up the **Orange** box, visit the **Yellow**, **Red** and **Light Orange** targets, visit the **Pink** target, drop the box there, then visit the **Light Green** and the **Green** targets.

To show how the approach for giving feedback during demonstrations can refine previously learned, incorrect task representations, let us assume that the actual task the robot should learn is as follows: visit the **Light Green** target, pick up the **Orange** box, visit the **Yellow** and **Red** targets, visit the **Pink** target,



(a) Demonstration environment (b) Practice environment

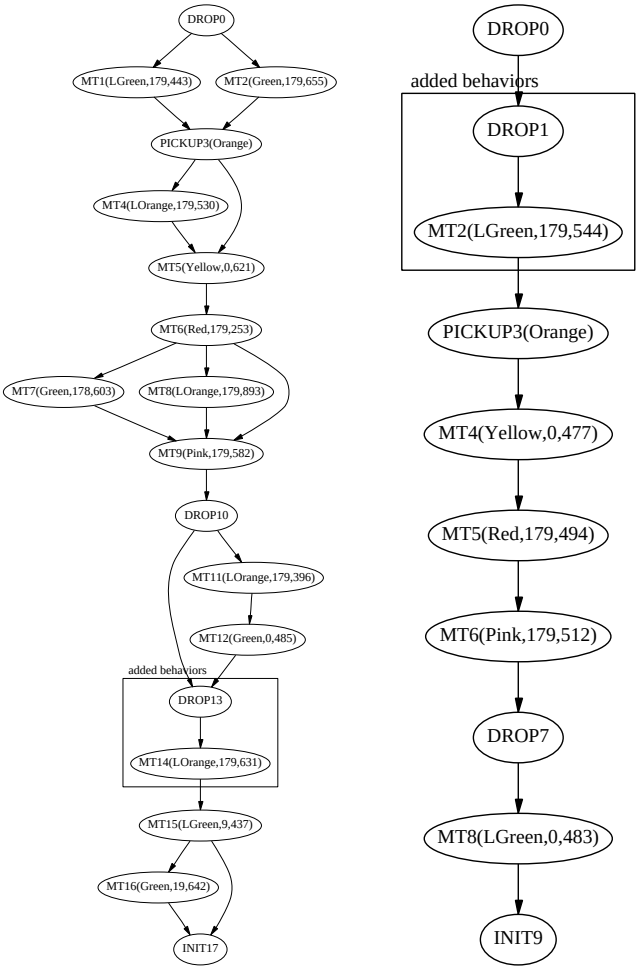
Figure 6.4: Environment for feedback after one demonstration

drop the box there, then visit the **Light Green** target. During the practice run, the teacher observed the following errors in the robot's performance:

- The initial visit to a **Green** target is wrong; a **Light Green** target should be visited instead.
- The visit to the **Light Orange** target is wrong, and not a part of the task.
- The end visit to the **Green** target is wrong, and not a part of the task as well.

Figure 6.4(b) shows the trajectory of the robot and (dotted) the intervention and messages of the teacher during the robot's practice run. The effects of this feedback are that: the visit to the **Green** target was replaced by a visit to the **Light Green** target, and the visits to the **Light Orange** and **Green** have been removed. Figure 6.5(b) presents the structure of the task after this practice run.

To validate the correctness of the learned representations, we had the robot execute the tasks learned after both of the practice runs described above: in each case the execution proved that the robot correctly adapted its task representations according to the teacher's feedback, which matched the target tasks intended by the user.



(a) After the 3rd demonstration

(b) After the first demonstration

Figure 6.5: Topologies obtained after practice and feedback

6.4 Discussion

We observe that the *practice-feedback* runs are a more precise method for refining previously learned tasks. Since feedback can be given at any step during the robot's practice, incorrect steps can be marked either directly upon observation during execution, or, as in the case of correcting missing steps, in a subsequent practice run.

Another important feature of the practice-feedback approach that we need to stress is the naturalness of this process. In order to give the robot appropriate feedback, the teacher does not need to know the structure

of the task being learned, and thus is shielded from having to know any details about the robot's control architecture. Instead, he simply relies on observing the actions performed by the robot: if they comply with the desired representation, no feedback is given, and if they do not, the corresponding situations are treated with appropriate feedback as described in our experiments.

6.5 Summary

This chapter presented an interactive approach for providing a robot with information necessary to refine previously learned, but incorrect tasks. It showed how giving direct feedback during a robot's task performance helps solving the two main problems that can occur during teaching by demonstration that may also persist after generalization. The experiments demonstrate that both *learning of irrelevant steps* and *failing to capture task-relevant aspects* can be solved through teacher intervention.

Chapter 7

Conclusions

This dissertation presented a framework that incorporates novel approaches to robot learning, autonomous control and human-robot interaction. The framework extends the capabilities of autonomous robots and their ability to function in dynamic, unpredictable environments. Robots that can effectively operate in such environments require robust and flexible real-time control and the ability to perform complex tasks. In addition, for efficient interaction with humans or other robots in their environment, robots need to be endowed with capabilities for learning such tasks from other agents.

The framework presented in this dissertation provides an approach for learning and refining high-level task representations of robotic tasks from *instructive demonstrations*, *generalization* over multiple demonstrations, and *practice* under a teacher's supervision, based on a set of underlying capabilities (behaviors) available to the robot. For robot control, the method uses a flexible *Hierarchical Abstract Behavior-Based Architecture*, developed to extend the capabilities of standard behavior-based systems with AI-level concepts.

The developed task learning technique is based on experienced demonstrations, during which the robot actively participates in the demonstration along with the teacher and experiences the task through its own sensors. The robot learns by creating a mapping between the observations and its own skills that achieve the effects perceived during the teacher's demonstration.

Due to inherent challenges of the learning process, it is important that robots be able to improve their capabilities by receiving additional training and feedback, similarly with the approach humans use when teaching each other by demonstration. The solution developed in this dissertation allows a teacher to use relevant *cues* during training, thus providing the robot with essential information regarding the relevant and irrelevant aspects of a demonstration. In addition, concise *instructions* allow for a richer demonstration, by actively involving the robot in the process. Through *generalization*, the robot can incorporate several demonstrations of the same task into a single graph-like representation. *Feedback cues* provided by the teacher allow the robot to further refine this representation during *practice* experiences.

The *Hierarchical Abstract Behavior-Based Architecture* presented in this dissertation uses an *action-embedded* approach for task representation and has the following key features: 1) ability to encode and execute sequential, hierarchically structured tasks within a behavior-based framework; 2) behavior reusability across different tasks; 3) means for sequential and opportunistic task execution; and 4) support for automatic generation of a behavior-based system. With this architecture robot tasks are encoded as hierarchical behavior networks, which represent the sequential and hierarchical task dependencies.

The proposed concepts were implemented and validated on a Pioneer 2DX mobile robot. The key features of the control architecture were demonstrated in numerous examples that showed the use of hierarchical design, sequential and opportunistic task execution. The experimental results also validated the ability of our approach to incorporate multiple means for instruction and learning in order to teach robots various tasks through demonstration, generalization, and practice. In addition, the experiments performed show that *generalization* and *feedback* can be used interchangeably in various combinations, providing the teacher the flexibility to instruct the robot in the manner considered most suited for each case.

Reference List

Agre, P. A. & Chapman, D. (1990), 'What Are Plans For?', *Journal of Robotics and Autonomous Systems* **6**(1-2), 17–34.

Aha, D. W. & Salzberg, S. L. (1993), Learning to Catch: Applying Nearest Neighbor Algorithms to Dynamic Control Tasks, *in* 'Fourth International Workshop on Artificial Intelligence and Statistics', pp. 363–368.

Allen, J. F. (1983), 'Maintaining Knowledge about Temporal Intervals', *Communications of the ACM* **26**(11), 832–843.

Angros, R. H. (2000), Learning What to Instruct: Acquiring Knowledge from Demonstrations and focused experimentation, PhD thesis, University of Southern California.

Arbib, M. (1992), Schema Theory, *in* S.Shapiro, ed., 'The Encyclopedia of Artificial Intelligence', Wiley-Interscience, pp. 1427–1443.

Arkin, R. C. (1987), Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior, pp. 264–271.

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

Arkin, R. C. & Balch, T. (1997), 'AuRA: Principles and Practice in Review', *Journal of Experimental and Theoretical AI* **2-3**, 175–189.

Atkeson, C. G., Moore, A. W. & Schaal, S. (1997), 'Locally Weighted Learning for Control', *Artificial Intelligence Review* **11**(1-5), 75–113.

Atkin, M. S., King, G. W., Westbrook, D. L., Heeringa, B., Hannon, A. & Cohen, P. (2001), SPT: Hierarchical Agent Control: a Framework for Defining Agent Behavior, *in* 'Proc., Intl. Conf. on Autonomous Agents', pp. 425–432.

Benson, S. & Nilsson, N. J. (1994), Reacting, Planning and Learning in an Autonomous Agent, *in* K. Furukawa, S. Muggleton & D. Michie, eds, 'Machine Intelligence', Vol. 14, Oxford University Press,.

Billard, A. & Dautenhahn, K. (1998), 'Grounding Communication in Autonomous Robots: an Experimental Study', *Robotics and Autonomous Systems, Special Issue on Scientific methods in mobile robotics* **24:1-2**, 71–79.

Billard, A. & Hayes, G. (1998), 'DRAMA, a Connectionist Architecture for Control and Learning in Autonomous Robots', *Adaptive Behaviour Journal* **7:2**, 35–64.

Bindiganavale, R., Schuler, W., Allbeck, J. M., Badler, N. I., Joshi, A. K. & Palmer, M. (2000), Dynamically Altering Agent Behaviors Using Natural Language Instructions, *in* 'Proc., Intl. Conf. on Autonomous Agents', pp. 293–300.

Blyth, C. R. (1986), 'Approximate Binomial Confidence Limits', *Journal of the American Statistical Association* **81**(395), 843–855.

Bonasso, R. P., Firby, R. J., Gat, E., Miller, D. K. D. & Slack, M. (1997), 'Experiences with an Architecture for Intelligent, Reactive Systems', *Journal of Experimental and Theoretical Artificial Intelligence* **9**(2–3), 237–256.

Booker, L. B. (1988), 'Classifier Systems that Learn Internal World Models', *Machine Learning* **3**, 161–192.

Brand, M. (1996), Understanding Manipulation in Video, *in* 'Proc., the 2nd Intl. Conf. on Face and Gesture Recognition', Killington, VT, pp. 94–99.

Brand, M. (1997), The "Inverse Hollywood Problem": From Video to Scripts and Storyboards via Causal Analysis, *in* 'Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)', AAAI Press, Menlo Park, pp. 132–137.

Brooks, R. A. (1986), 'A Robust Layered Control System for a Mobile Robot', *IEEE Journal of Robotics and Automation* **2**(1), 14–23.

Brooks, R. A. (1990*a*), The Behavior Language: User's Guide, Technical Report AIM 1227, MIT AI Lab, Cambridge, MA.

Brooks, R. A. (1990*b*), 'Elephants Don't Play Chess', *Journal of robotics and autonomous systems* **6**(1-2), 3–15.

Brooks, R. A., Connell, J. H. & Ning, P. (1988), HERBERT: A Second Generation Mobile Robot, Technical Report AIM 1016, MIT AI Lab, Cambridge, MA.

Carbonell, J. G. (1983), Learning by Analogy: Formulating and Generalizing Plans from Past Experience, *in* J. C. R.S. Michalsky & T. Mitchell, eds, 'Machine Learning: An Artificial Intelligence Approach', Tioga Publ., pp. 371–392.

Carbonell, J. G. & Gil, Y. (1990), Learning by Experimentation: the Operator Refinement Method, *in* Y. Kodratoff & R. S. Michalski, eds, 'Machine Learning: An Artificial Intelligence Approach', Vol. 3, Morgan Kaufmann, pp. 191–213.

Chapman, D. (1987), 'Planning for Conjunctive Goals', *Aritifical Intelligence* **32**, 333–377.

Clemans, K. G. (1959), 'Confidence Limits in the Case of the Geometric Distribution', *Biometrika* **46**(1/2), 260–264.

Colombetti, M. & Dorigo, M. (1994), 'Training Agents to Perform Sequential Behavior', *Adaptive Behavior* **2**(3), 247–275.

Connell, J. H. (1990), *Minimalist Mobile Robotics: A Colony-style Architecture for a Mobile Robot*, Academic Press.

Connell, J. H. (1992), SSS: A Hybrid Architecture Applied to Robot Navigation, *in* 'Proc. of the IEEE Int. Conf. on Robotics and Automation', Nice, France, pp. 2719–2724.

Cormen, T. H., Leiserson, C. E. & Rivest, R. L. (1990), *Introduction to Algorithms*, MIT Press.

Dautenhahn, K. (1995), 'Getting to Know Each Other - Artificial Social Intelligence for Autonomous Robots', *Robotics and Autonomous systems* **16**, 333–356.

Dauthenhahn, K. (1994), Trying to Imitate - a Step Towards Releasing Robots From Social Isolation, *in* 'Proc., From Perception to Actions Conference', IEEE Computer Society Press, Lausanne, Switzerland, pp. 290–301.

Dayan, P. (1992), 'The Convergence of TD(λ) for General λ', *Machine Learning* **8**, 341–362.

DeJong, G. & Mooney, R. J. (1986), 'Explanation-Based Learning: An Alternate View', *Machine Learning* **1**(2), 145–176.

del R. Millan, J. (1996), 'Rapid, Safe, and iIcremental Learning of Navigation Strategies', *IEEE Transactions on Systems, Man and Cybernetics. Special issue on Learning Approaches to Autonomous Robots Control* **26**(6), 408–420.

del R. Millan, J. & Torras, C. (1992), 'A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze Like Environments', *Machine Learning* **8**(3–4), 363–395.

Delson, N. & West, H. (1996), Robot Programming by Human Demonstration: Adaptation and Inconsistency in Constrained Motion, *in* 'Proc., IEEE Intl. Conf. on Robotics and Automation', Minneapolis, MN, pp. 30–36.

Demiris, J. & Hayes, G. (1999), Active and Passive Routes to Imitation, *in* 'Proc., the AISB Symposium on Imitation in Animals and Artifacts', Edinburgh.

Demiris, Y. & Hayes, G. (2002), Imitation as a Dual-Route Process Featuring Predictive and Learning Components: a Biologically-Plausible Computational Model, *in* K. Dautenhahn & C. Nehaniv, eds, 'Imitation in Animals and Artifacts', MIT Press, pp. 321–361.

Dorigo, M. & Colombetti, M. (1994), 'Robot Shaping: Developing Autonomous Agents Through Learning', *Artificial Intelligence* **2**, 321–370.

Dorigo, M. & Colombetti, M. (1997), *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press, Cambridge.

Farrell, S., Maglio, P. P. & Campbell, C. S. (2001), How to Teach a Fish to Swim, *in* 'Proc., IEEE Symp. on Human-Centric Computing Languages and Environments', IEEE Computer Society, Stresa, Italy, pp. 158–164.

Fikes, R. E. & Nilsson, N. J. (1971), 'STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving', *Artificial Intelligence* **2**, 189–208.

Firby, J. R. (1989), Adaptive Execution in Complex Dynamic Worlds, PhD thesis, Yale University, Computer Science Department.

Friedrich, H. & Dillmann, R. (1995), Robot Programming Based On A Single Demonstration And User Intentions, *in* 'Proc., 3rd European Workshop on Learning Robots', Crete, Grece.

Friedrich, H. & Kaiser, M. (1995), What Can Robots Learn from Humans?, *in* 'Proc., IFAC Workshop on Human-Oriented Design of Advanced Robotic Systems', Vienna, Austria.

Friedrich, H., Hofmann, H. & Dillmann, R. (1997), 3D-icon Based User Interaction for Robot Programming by Demonstration, *in* 'Proc., IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation', Monterey, CA, pp. 240–245.

Fuellen, G. (1997), 'Multiple Alignment', *Complexity International* **4**, http://www.csu.edu.au/ci/vol04/mulali/mulali.htm.

Garcia, P. & Vidal, E. (1990), 'Inference of K-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(9), 920–925.

Gat, E. (1998), On Three-Layer Architectures, *in* D. Kortenkamp, R. P. Bonnasso & R. Murphy, eds, 'Artificial Intelligence and Mobile Robotics', AAAI Press, pp. 195–210.

Gaussier, P., Moga, S., Banquet, J. & Quoy, M. (1998), 'From Perception-Action Loops to Imitation Processes: A Bottom-up Approach of Learning by Imitation', *Applied Artificial Intelligence Journal* **12(78)**, 701–729.

Georgeff, M. P. & Lansky, A. L. (1987), Reactive Reasoning and Planning, *in* 'Proc., Intl. Conf. of the American Association of Artificial Intelligence', Seattle, WA, pp. 677–682.

Goldberg, D. (2001), Evaluating the Dynamics of Agent-Environment Interaction, PhD thesis, University of Southern California.

Goldberg, D. & Matarić, M. J. (1999), Coordinating Mobile Robot Group Behavior Using a Model of Interaction Dynamics, *in* 'Proc., the Third Intl. Conf. on Autonomous Agents', ACM Press, Seattle, Washington, pp. 100–107.

Goldberg, D. & Matarić, M. J. (2000*a*), Learning Multiple Models for Reward Maximization, *in* 'Proc., the Seventh Intl. Conf. on Machine Learning', Stanford University, pp. 319–326.

Goldberg, D. & Matarić, M. J. (2000*b*), Reward Maximization in a Non-Stationary Mobile Robot Environment, *in* 'Proceedings of the Fourth International Conference on Autonomous Agents', ACM Press, Barcelona, Catalonia, Spain, pp. 92–99.

Hayes, G. & Demiris, J. (1994), A Robot Controller Using Learning by Imitation, *in* 'Proc. of the Intl. Symp. on Intelligent Robotic Systems', Grenoble, France, pp. 198–204.

Horswill, I. (1997), Real-Time Control of Attention and Behavior in a Logical Framework, *in* 'Proc., the First Intl. Conf. on Autonomous Agents', ACM Press, New York, pp. 130–137.

Hovland, G. E., Sikka, P. & McCarragher, B. J. (1996), Skill Acquisition from Human Demonstration Using a Hidden Markov Model, *in* 'Proc., Intl. Conf. on Robotics and Automation', Minneapolis, MN, pp. 2706–2711.

Huber, M. & Grupen, R. (1997), 'A Feedback Control Structure for On-Line Learning Tasks', *Robotics and autonomous systems* **22**(3–4), 303–315.

Hugues, L. & Drogoul, A. (2002), Synthesis of Robot's Behaviors from Few Examples, *in* 'Proc., IEEE Intl. Conf. on Intelligent Robots and Systems', Lausanne, Switzerland, pp. 909–914.

Ikeuchi, K. & Suehiro, T. (1992), Towards an Assembly Plan from Observation, *in* 'Proc. of Intl. Conf. on Robotics and Automation', Nice, France, pp. 2171–2177.

Ikeuchi, K., Kawade, M. & Suehiro, T. (1993), Assembly Task Recognition with Planar, Curved and Mechanical Contacts, *in* 'Proc., IEEE Intl. Conf. on Robotics and Automation', Atlanta, Georgia, USA, pp. 688–694.

Jenkins, O. C. & Matarić, M. J. (2002), Deriving Action and Behavior Primitives from Human Motion Data, *in* 'Proc., IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems', pp. 2551–2556.

Jenkins, O. C. & Matarić, M. J. (2003), Automated Derivation of Behavior Vocabularies for Autonomous Humanoid Motion, *in* 'to appear in Proc., Second Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems', Melbourne, Australia.

Jenkins, O. C., Matarić, M. J. & Weber, S. (2000), Primitive-Based Movement Classification for Humanoid Imitation, *in* 'Proc., First IEEE-RAS Intl. Conf. on Humanoid Robotics', Cambridge, MA, MIT.

Kaelbling, L. P. (1990*a*), Learning Functions in k-DNF from Reinforcement, *in* 'Machine Learning: Proceedings of the Seventh International Conference', Morgan Kaufmann, Austin, Texas, pp. 162–169.

Kaelbling, L. P. (1990*b*), Learning in Embedded Systems, PhD thesis, Department of Computer Science, Stanford University.

Kaelbling, L. P. & Rosenschein, S. J. (1990), 'Action and Planning in Embedded Agents', *Robotics and Autonomous Systems* **6**(1-2), 35–48.

Kaelbling, L. P., Littman, M. L. & Moore, A. P. (1996), 'Reinforcement Learning: A Survey', *Journal of Artificial Intelligence Research* **4**, 237–285.

Kaiser, M. (1997), 'Transfer of Elementary Skills via Human-Robot Interaction', *Adaptive Behavior* **5**(3-4), 249–280.

Kaiser, M. & Dillmann, R. (1996), Building Elementary Robot Skills from Human Demonstration, *in* 'Proc., IEEE Intl. Conf. on Robotics and Automation', Minneapolis, Minnesota, pp. 2700–2705.

Kaiser, M. & Dillmann, R. (1997), 'Hierarchical Refinement of Skills and Skill Application for Autonomous Robots', *Robotics and Autonomous Systems* **19**, 259–271.

Kaplan, F., Oudeyer, P.-Y., Kubinyi, E. & Miklûsi, A. (2002), 'Robotic Clicker Training', *Robotics and Autonomous Systems* **38**, 197–206.

Kosecká, J. & Bogoni, L. (1994), Application of Discrete Events Systems for Modeling and Controlling Robotic Agents, *in* E. Straub & R. S. Sipple, eds, 'Proc., Intl. Conf. on Robotics and Automation', Vol. 3, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 2557–2562.

Kuniyoshi, Y. & Inoue, H. (1993), Qualitative Recognition of Ongoing Human Action Sequences, *in* 'Proc., Intl. Joint Conf. on Artificial Intelligence', Washington, DC, pp. 1600–1609.

Kuniyoshi, Y., Inaba, M. & Inoue, H. (1994), 'Learning By Watching: Extracting Reusable Task Knowledge From Visual Observation Of Human Performance', *IEEE Transaction on Robotics and Automation* **10**(6), 799–822.

Larson, A. & Voyles, R. (2001), Automatic Training Data Selection for Sensory Motor Primitives, *in* 'Proc., IEEE Intl. Conf. on Intelligent Robots and Systems', Wailea, Hawaii, pp. 871–876.

Lin, L. J. (1991), Programming Robots Using Reinforcement Learning and Teaching, *in* 'Proc., Natl. Conf. on Artificial Intelligence', Anaheim, California, pp. 781–786.

Maeda, Y., Ishido, N., Kikuchi, H. & Arai, T. (2002), Teaching of Grasp/Graspless Manipulation for Industrial Robots by Human Demonstration, *in* 'Proc., Intl. Conf. on Intelligent Robots and Systems', pp. 1523–1528.

Maes, P. (1990*a*), 'How to do the Right Thing', *Connection Science Journal, Special Issue on Hybrid Systems* **1**(3), 291–323.

Maes, P. (1990*b*), 'Situated Agents Can Have Goals', *Journal for Robotics and Autonomous Systems* **6**(3), 49–70.

Maes, P. & Brooks, R. A. (1990), Learning to Coordinate Behaviors, *in* 'Proc., Intl. Conf. of American Association of Artificial Intelligence', Boston, MA, pp. 796–802.

Mahadevan, S. & Connell, J. (1991), Scaling Reinforcement Learning to Robotics by Exploiting the Subsumption Architecture, *in* 'Eighth Intl. Workshop on Machine Learning', Morgan Kaufmann, pp. 328–337.

Matarić, M. J. (1992), 'Integration of Representation Into Goal-Driven Behavior-Based Robots', *IEEE Transactions on Robotics and Automation* **8**(3), 304–312.

Matarić, M. J. (1994), Reward Functions for Accelerated Learning, *in* 'Proc., Eleventh Intl. Conf. on Machine Learning', Morgan Kaufmann, San Francisco, CA, pp. 181–189.

Matarić, M. J. (1997*a*), 'Behavior-Based Control: Examples from Navigaton, Learning, and Group Behavior', *Journal of Experimental and Theoretical Artificial Intelligence* **9**(2–3), 323–336.

Matarić, M. J. (1997*b*), 'Reinforcement Learning in the Multi-Robot Domain', *Autonomous Robots* **4**(1), 73–83.

McCallum, A. K. (1996), Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks, *in* P. Maes, M. J. Matarić, J.-A. Meyer, J. Pollack & S. W. Wilson, eds, 'From Animals to Animats IV, Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior', Cape Cod, Massachusetts, pp. 315–324.

Meltzoff, A. N. & Moore, M. K. (1997), 'Explaining Facial Imitation: a Theoretical Model', *Early Development and Parenting* **6**, 179–192.

Michael Kasper, Gernot Fricke, K. S. & von Puttkamer, E. (2001), 'A Behavior-Based Mobile Robot Architecture for Learning from Demonstration', *Robotics and Autonomous Systems* **34**, 153–164.

Michaud, F. & Matarić, M. J. (1998*a*), 'Learning from History for Behavior-Based Mobile Robots in Non-Stationary Conditions', *Machine Learning* **31**, 141–167.

Michaud, F. & Matarić, M. J. (1998*b*), 'Representation of Behavioral History for Learning in Nonstationary Conditions', *Robotics and Autonomous Systems* **29**, 187–200.

Mitchell, R. W. (1987), 'A Comparative-Developmental Approach to Understanding Imitation', *Perspectives in Ethology* **7**, 183–215.

Mitchell, T. M. & Thrun, S. B. (1993), Explanation-Based Neural Network Learning for Robot Control, *in* R. P. L. J. E. Moody, S. J. Hanson, ed., 'Advances in Neural Information Processing Systems', Vol. 5, Morgan Kaufmann, San Mateo, CA, pp. 287–294.

Mitchell, T. M., Utgoff, P. E. & Banerji, R. (1993), Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics, *in* J. C. R.S. Michalski & T. Mitchell, eds, 'Machine Learning: An Artificial Intelligence Approach', Vol. 1, Springer, chapter 6, pp. 163–190.

Miyamoto, H. & Kawato, M. (1998), 'A Tennis Serve and Upswing Learning Robot Based on Bi-Directional Theory', *Neural Networks* **11**, 1331–1344.

Miyamoto, H., Schaal, S., Gandolfo, F., Gomi, H., Koike, Y., Osu, R., Nakano, E., Wada, Y. & Kawato, M. (1996), 'A Kendama Learning Robot Based on Bi-Directional Theory', *Neural Networks* **9**, 1281–1302.

Moore, A. W., Atkeson, C. G. & Schaal, S. (1995), Memory-Based Learning for Control, Technical Report CMU-RI-TR-95-18, CMU Robotics Institute.

Moravec, H. P. (1977), Towards Automatic Visual Obstacle Avoidance, *in* 'Proc., Intl. Joint Conf. on Artificial Intelligence', Morgan Kaufman, Cambridge, MA, pp. 584–589.

Moravec, H. P. (1990), The Stanford Cart and the CMU Rover, *in* I. J. Cox & G. T. Wilfong, eds, 'Autonomous Robot Vehicles', Springer-Verlag, pp. 407–412.

Munro, A., Johnson, M. C., Pizzini, Q. A., Surmon, D. S., Towne, D. M. & Wogulis, J. L. (1997), 'Authoring Simulation-Centered Tutors with RIDES', *International Journal of Artificial Intelligence in Education* **8**, 284–316.

Nicolescu, M. N. & Matarić, M. J. (2001*a*), Experience-Based Representation Construction: Learning from Human and Robot Teachers, *in* 'Proc., IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems', Maui, Hawaii, USA, pp. 740–745.

Nicolescu, M. N. & Matarić, M. J. (2001*b*), Learning and Interacting in Human-Robot Domains, *in* C. C. White & K. Dautenhahn, eds, 'IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, Special Issue on Socially Intelligent Agents - The Human in the Loop', Vol. 31(5), IEEE Press, pp. 419–430.

Nicolescu, M. N. & Matarić, M. J. (2002), A Hierarchical Architecture for Behavior-Based Robots, *in* 'Proc., First Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems', Bologna, ITALY, pp. 227–233.

Nicolescu, M. N. & Matarić, M. J. (2003*a*), Linking Perception and Action in a Control Architecture for Human-Robot Domains, *in* 'Proc., Thirty-Sixth Hawaii Intl. Conf. on System Sciences', IEEE Computer Society, Hawaii, USA.

Nicolescu, M. N. & Matarić, M. J. (2003*b*), Natural Methods for Robot Task Learning: Instructive Demonstration, Generalization and Practice, *in* 'Proc., Second Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems', Melbourne, AUSTRALIA.

Nilsson, N. J. (1984), Shakey the Robot, Technical Report 323, Artificial Intelligence Center, Computer Science and Technology Division, SRI International, Menlo Park, CA.

Nilsson, N. J. (1994), 'Teleo-Reactive Programs for Agent Control', *Journal of Artificial Intelligence Research* **1**, 139–158.

Ogata, H. & Takahashi, T. (1994), 'Robotic Assembly Operation Teaching in a Virtual Environment', *IEEE Transactions on Robotics and Automation* **10**(3), 391–399.

Ogawara, K., Takamatsu, J., Kimura, H. & Ikeuchi, K. (2002*a*), Generation of a Task Model by Integrating Multiple Observations of Human Demonstrations, *in* 'Proc., Intl. Conf. on Robotics and Automation', Washington, DC, pp. 1545–1550.

Ogawara, K., Takamatsu, J., Kimura, H. & Ikeuchi, K. (2002*b*), Modeling Manipulation Interactions by Hidden Markov Models, *in* 'Proc., Intl. Conf. on Intelligent Robots and Systems', Washington, DC, pp. 1096–1101.

Onda, H., Suehiro, T. & Kitagaki, K. (2002), Teaching by Demonstration of Assembly Motion in VR - Non-Deterministic Search-Type Motion in the Teaching Stage, *in* 'Proc., IEEE Intl. Conf. on Intelligent Robots and Systems', Lausanne, Switzerland, pp. 3066–3072.

O'Sullivan, J., Mitchell, T. & Thrun, S. (1997), Explanation Based Learning for Mobile Robot Perception, *in* K. Ikeuchi & M. Veloso, eds, 'Symbolic Visual Learning', Oxford University Press.

P. Garcia, E. V. & Casacuberta, F. (1987), 'Local Languages, the Successor Method and a Step Towards a General Methodology for the Inference of Regular Grammars', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **9**(6), 841–845.

Pearson, D. J. (1996), Learning Procedural Planning Knowledge In Complex Environments, PhD thesis, Computer Science and Engineering Dept., University of Mitchigan.

Pearson, D. J., Huffman, S. B., Willis, M. B., Laird, J. E. & Jones, R. M. (1993), 'A Symbolic Solution to Intelligent Real-Time Control', *Robotics and Autonomous Systems* **11**, 279–291.

Pirjanian, P. (1999), Behavior Coordination Mechanisms - State-of-the-art, Tech Report IRIS-99-375, Institute for Robotics and Intelligent Systems, University of Southern California, Los Angeles, California.

Pomerleau, D. A. (1991), Rapidly Adapting Artificial Neural Networks for Autonomous Navigation, *in* R. P. Lippmann, J. E. Moody & D. S. Touretzky, eds, 'Advances in Neural Information Processing Systems', Vol. 3, Morgan Kaufmann Publishers, Inc., pp. 429–435.

Pook, P. K. & Ballard, D. H. (1993), Recognizing Teleoperated Manipulations, *in* 'Proc., Intl. Conf. on Robotics and Automation', Atlanta, Georgia, pp. 578–585.

Price, B. & Boutilier, C. (1999), Implicit Imitation in Multiagent Reinforcement Learning, *in* 'Proc. 16th International Conf. on Machine Learning', Morgan Kaufmann, San Francisco, CA, pp. 325–334.

Price, B. & Boutilier, C. (2001), Imitation and Reinforcement Learning in Agents with Heterogeneous Actions, *in* 'Proc., 4th Biennal Conf. of the Canadian Society for Computational Studies of Intelligence', Ottawa, Canada, pp. 111–120.

Ramesh, A. & Matarić, M. J. (2002), Learning Movement Sequences from Demonstration, *in* 'Proc., IEEE Intl. Conf. on Development and Learning', Cambridge, MA, pp. 203–208.

Rosenblatt, J. (1997), 'DAMN: A Distributed Architecture for Mobile Navigation', *Journal of Experimental and Theoretical Artificial Intelligence* **9**(2/3), 339–360.

Russell, S. & Norvig, P. (1995), *AI: A Modern Approach*, Prentice Hall, NJ.

Saffiotti, A., Ruspini, E. H. & Konolige, K. (1993), Blending Reactivity and Goal-Directedness in a Fuzzy Controller, *in* 'Proc., IEEE Intl. Conf. on Neural Nets and Fuzzy Control', San Francisco, California, pp. 134–139.

Sammut, C., Hurst, S., Kedzier, D. & Michie, D. (1992), Learning to Fly, *in* 'Proc., the Ninth Intl. Conf. on Machine Learning', Morgan Kaufmann, Aberdeen, Scotland, UK, pp. 385–393.

Schaal, S. (1997), Learning from demonstration, *in* M. Mozer, M. Jordan & T. Petsche, eds, 'Advances in Neural Information Processing Systems 9', MIT Press, Cambridge, pp. 1040–1046.

Schaal, S. (1999), 'Is Imitation Learning the Route to Humanoid Robots?', *Trends in Cognitive Sciences* **6**(3), 323–242.

Schaal, S. & Atkeson, C. G. (1994), 'Robot Juggling: an Implementation of Memory-Based Learning', *Control Systems Magazine* **14**(1), 57–71.

Schoppers, M. J. (1987), Universal Plans for Reactive Robots in Unpredictable Environments, *in* J. McDermott, ed., 'Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)', Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, Milan, Italy, pp. 1039–1046.

Segre, A. M. & DeJong, G. (1985), Explanation-Based Manipulator Learning: Acquisition of Planning Ability Through Observation, *in* 'IEEE Conference on Robotics and Automation', Nagoya, Japan, pp. 555–560.

Simmons, R. G. (1994), 'Structured Control for Autonomous Robots', *IEEE Transactions on Robotics and Automation* **10**(1), 34–43.

Singh, S. P. (1991), Transfer of Learning Across Compositions of Sequential Tasks, *in* 'Proceedings, Eighth International Conference on Machine Learning', Morgan Kaufmann, Evanston, Illinois, pp. 348–352.

Steels, L. (1994), A Case Study in the Behavior-Oriented Design of Autonomous Agents, *in* 'Proc., Intl. Conf. on Simulation of Adaptive Behavior', Bringhton, UK, pp. 445–452.

Sun, R. & Peterson, T. (1998), 'Autonomous Learning of Sequential Tasks: Experiments and Analyses', *IEEE Transactions on Neural Networks* **9**(6), 1217–1234.

Sun, R. & Sessions, C. (2000), 'Self-Segmentation of Sequences: Automatic Formation of Hierarchies of Sequential Behaviors', *IEEE Transactions on Systems, Man, and Cybernetics: Part B Cybernetics* **30**(3), 403–418.

Sun, R., Peterson, T. & Merrill, E. (1998), 'A Hybrid Architecture for Situated Learning of Reactive Sequential Decision Making', *Applied Intelligence* **11**, 109–127.

Sutton, R. S. (1988), 'Learning to Predict by the Methods of Temporal Differences', *Machine Learning* **3**, 9–44.

Sycara, K., Williamson, M. & Decker, K. (1996), Unified Information and Control Flow in Hierarchical Task Networks, *in* 'Working Notes of the AAAI-96 workshop Theories of Action, Planning, and Control'.

Tambe, M., Johnson, L. W., Jones, R. M., Koss, F. V., Laird, J. E., Rosenbloom, P. S. & Schwamb, K. (1995), 'Intelligent Agents for Interactive Simulation Environments', *AI Magazine* **16**(1), 15–39.

Thorpe, W. H. (1963), *Learning and Instinct in Animals*, London: Methuen.

Thrun, S. & O'Sullivan, J. (1996), Discovering Structure in Multiple Learning Tasks: the TC Algorithm, *in* L. Saitta, ed., 'Proc., The 13th Intl. Conf. on Machine Learning', Morgan Kaufmann, San Mateo, CA.

Todd, D. J. (1986), *Fundamentals of Robot Technology*, John Wiley and Sons.

Tominaga, H., Takamatsu, J., Ogawara, K., Kimura, H. & Ikeuchi, K. (2000), Symbolic Representation of Trajectories for Skill Generation, *in* 'Proc., Intl. Conf. on Robotics and Automation', San Francisco, California, pp. 4077–4082.

van der Smagt, P., Groen, F. & van het Groenewoud, F. (1994), The Locally Linear Nested Network for Robot Manipulation, *in* 'Proc., IEEE Intl. Conf. on Neural Networks', Florida, pp. 2787–2792.

van Lent, M. & Laird, J. (1999), Learning Hierarchical Performance Knowledge by Observation, *in* 'Proc. 16th International Conf. on Machine Learning', Morgan Kaufmann, San Francisco, CA, pp. 229–238.

Voyles, R. & Khosla, P. (2001), 'A Multi-Agent System for Programming Robots by Human Demonstration', *Integrated Computer-Aided Engineering* **8**(1), 59–67.

Voyles, R. M., Morrow, J. D. & Khosla, P. K. (1997), 'Towards Gesture-Based Programming', *Robotics and Autonomous Systems* **22**, 361–375.

Wang, X. (1996), Learning Planning Operators by Observation and Practice, PhD thesis, School of Computer Science, Carnegie Mellon University.

Wang, X. & Carbonell, J. (1994), Learning by Observation and Practice: Towards Real Applications of Planning Systems, *in* 'Planning and Learning: On to Real Applications: Papers from the 1994 AAAI Fall Symposium', AAAI Press, Menlo Park, California, pp. 156–161.

Watkins, C. J. C. H. (1989), Models of Delayed Reinforcement Learning, PhD thesis, Psychology Department, Cambridge University, Cambridge, United Kingdom.

Watkins, C. J. C. H. & Dayan, P. (1992), 'Q-Learning', *Machine Learning* **8**, 279–292.

Werger, B. B. (2000), Ayllu: Distributed Port-Arbitrated Behavior-Based Control, *in* 'Proc., The 5th Intl. Symp. on Distributed Autonomous Robotic Systems', Springer, Knoxville, TN, pp. 25–34.

Yang, J., Xu, Y. & Chen, C. S. (1993), Hidden Markov Model Approach to Skill Learning and its Application in Telerobotics, *in* 'Proc., Intl. Conf. on Intelligent Robots and Systems', Yokohama, Japan, pp. 396–402.

Yang, J., Xu, Y. & Chen, C. S. (1997), 'Human Action Learning via Hidden Markov Model', *IEEE Trans. on Systems Man and Cybernetics* **27**(1), 34–44.

Zollner, R., Rogalla, O., Dillmann, R. & Zollner, M. (2002), Understanding Users Intention: Programming Fine Manipulation Tasks by Demonstration, *in* 'Proc., IEEE Intl. Conf. on Intelligent Robots and Systems', Lausanne, Switzerland, pp. 1114–1119.

Appendix A

Teaching Your Robot

The role of this appendix is to describe the information necessary to teach a robot by demonstration, using the approach presented in this dissertation. The goal of this work is to allow non-specialist users to program a robot, through demonstration, to perform a particular task, constructed from a set of underlying capabilities already available to the robot.

This appendix describes the essential information regarding the sensors, actuators, perceptions and skills of the robot and the mechanism for teaching by demonstration, that a non-specialist user would need to know.

A.1 How Does the Robot Perceive the Environment?

The robot perceives its environment using its *sensors*. The Pioneer 2-DX mobile robot used in these experiments is equipped with the following sensors:

- A laser range-finder that gives the robot information about its distance from objects in front of it. The rangefinder can only detect objects placed in a plane at the level of the laser. The field of view of the laser is 180 degrees, meaning that it detects all obstacles placed within 90 degrees to the left and 90 degrees to the right of the center of its field of detection.

- A pan-tilt-zoom color camera that allows the robot to detect up to 8 different, previously trained colors.
- Infrared sensors (IR), located on the inside of the gripper's paddles that allow the robot to detect the presence of an object within the gripper.
- A sound-detection ability that enables the robot to receive spoken commands from the teacher. The commands are given through the microphone of a cordless headset.

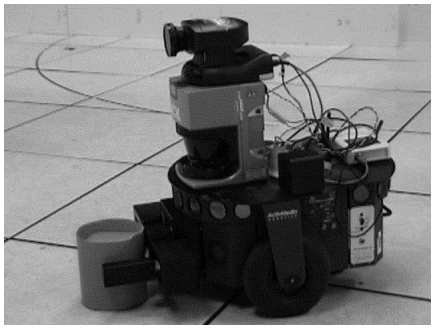


Figure A.1: A Pioneer 2DX robot

A.2 How Does the Robot Act On Its Environment?

The robot acts upon its environment using its actuators, which are mechanical devices for moving or controlling the robot or some of its parts. Following are the Pioneer's actuators:

- left and right wheels, which allow the robot to *move forward* and to *rotate*;
- a small mechanical hand called a *gripper*, allowing *grasping* and *releasing* of small objects placed within its reach.

A.3 What Can the Robot Perceive?

Using its available sensors the robot can detect the following features:

- **the presence of small objects** in any of the eight previously trained colors. Using its camera the robot detects objects as colored blobs in its visual field. The robot knows the size and position of each blob in the visual field.

- **the presence of tall cylindrical objects** in any of the eight previously trained colors. As with the features above, by using its camera, the robot detects the objects as colored blobs in its visual field. For these objects, the robot also measures the distance and angle to the objects using its laser range-finder.

- **the teacher**. The robot perceives the teacher as a tall, blue colored, cylindrical object. The robot has been previously trained to identify tall, blue targets as the teacher, and can detect the teacher up to a distance of approximately 2 meters.

- **gripper status**. If the robot's gripper is closed and the state of the IR sensors indicate the presence of an object, the robot detects that it is *carrying an object*. If the IR sensors of the robot do not indicate the presence of an object, the robot infers that its *gripper is empty and available*.

A.4 Robot Skills

The robot is equipped with the following three capabilities, which it can use to perform more complex tasks:

- **PickUp**: Using this skill the robot picks up a small object of a particular color from the pre-programmed color set. The color of the object indicates which type of object has to be picked up. In order to perform this skill, the robot wanders around in search for an object of the specified color. After detecting the object using its camera, the robot moves toward it and opens its gripper in order to pick the object up. When the IR sensors in the gripper detect the presence of the tracked object the gripper closes and lifts, grasping the object.

- **Drop**: The robot uses this skill to drop whatever it has in the gripper. To perform this skill, the robot opens the gripper and backs off while turning, in order to release the object, and avoid grabbing it again. When the IR sensors detect that there is no object present, the gripper closes and lifts.

- **Track**: Using this skill the robot can visit a tall object in one of the pre-programmed colors. To perform this skill, the robot wanders around in search of a tall object of a particular color. The robot first detects the target using the camera, then the robot combines the information from the camera and the laser range-finder to track the object and to position itself at a specific angle to and distance from it. Merging the information from the laser range-finder and the camera allows the robot to track targets within a 180 degree range, even after they go out of the visual field. The robot beeps when it reaches the target.

The robot also has two implicit skills that it uses in the following conditions:

- **Wander**: If the robot does not have a particular task to perform, or if none of the skills of a task are currently in execution, the robot wanders around.

- **Avoidance**: If the robot detects obstacles at distances smaller than a safe range for navigation, it automatically avoids those obstacles, even if other robot skills were being executed until that point.

A.5 Teaching the Robot

The robot is able to learn any particular task that uses its skills by performing the task along with the teacher. In order to train the robot, the teacher relies on two main methods: leading the robot and giving spoken instructions.

Following the teacher: during the entire demonstration the robot follows the teacher, maintaining a 50cm distance from him/her, using its camera and laser range-finder. The robot uses the known color assigned to the teacher, and uses its **Track** skill for following. During the demonstration, the teacher should stay

in the 180degree visual field of the robot's camera and laser-rangefinder, and should not go farther than 150cm away from the robot. The teacher should try to walk with a smooth, continuous movement and should try to walk with the legs close together to make a large and uniform target for the robot. If the robot gets distracted the teacher should move in front of it and take small steps to turn the robot away from the distraction.

Instructions: to give verbal instructions, the teacher speaks the commands into the microphone of the cordless headset. It is good practice to keep the headset turned off when not speaking to the robot, in order to eliminate undesired noise from the environment. The headset can be easily turned ON/OFF at the touch of a button. The teacher may give the following spoken commands (more details regarding the use of these instruction are given later in this and the next sections):

- **"START"** - signals the beginning of a demonstration; the robot starts moving toward the teacher.
- **"DONE"** - signals the end of a demonstration; robot stops moving: this is the end of the demonstration.
- **"TAKE"** - tells the robot to pick up a small object shown by the teacher and placed in its sensory range.
- **"DROP"** - tells the robot to drop anything that it might have in the gripper.
- **"HERE"** - tells the robot that it should pay attention to what is present in the environment, and to consider these observations as being relevant for the demonstrated task.
- **"BAD"** - signals to the robot that the currently executing skill, or the skill that was just finished, is not relevant for the task.
- **"COME"** - signals to the robot that it missed important steps of the task and that it should start following the teacher in order to learn these steps again.

- **"GO"** - signals to the robot that the demonstration started by the command "COME" has ended.

Here is how the teacher can use the above spoken commands to instruct the robot to perform a task:

- **PickUp:** To teach a robot that it has to pick up a small object of a particular color, the teacher: 1) shows the object to the robot, making sure that the object is in the robot's visual field, no more than one meter away, 2) says the command **"TAKE"**, and then 3) puts the object down. The robot will pick up the object on its own. (Note: when teaching the robot to perform this behavior, make sure to hold the box above the robot's laser rangefinder, so that the robot will not confuse the box with a tall cylinder.)

- **Drop:** To teach a robot that it has to drop what it is carrying, the teacher only has to say the command **"DROP."** The robot will drop any object it is carrying at that time.

- **Track:** To teach a robot to visit a particular target (such as a tall colored cylinder), the teacher walks toward the target, making sure that the object is in the robot's visual field and laser range. For this behavior, to minimize the possibility that the robot will perceive targets other than the one that is relevant, the teacher should say **"HERE"** when it knows that the robot has moved within 1.5m of the correct target. More specifically, the teacher should give this command when it approximates that the target is visible for the robot.

A.6 What if Something Goes Wrong?

If, after teaching, the robot does not perform the task correctly, it means that some problems may have occurred during the demonstrations. There are two possible errors that could cause the robot to perform the task incorrectly:

- The robot learned some steps that are not relevant for the demonstrated task. For example, while the user is teaching the robot to visit a particular target, the robot may pass close to another target and erroneously infer that this target should also be visited.

- The robot missed relevant steps of the task. The limitations of the sensory capabilities of the robot could make it fail to detect particular targets.

The teacher can detect these situations by observing the robot performing the learned task. If errors occur, the user may correct the robot by giving it appropriate feedback. Here is how the teacher *detects* and *corrects* the problems mentioned above:

- **Irrelevant steps:** These are steps that the robot should not be performing as a part of the task. The teacher can detect these incorrect steps by observing the performance of the robot. For example, if the robot gets close to a colored object and it beeps, it means that it went there on purpose. If that visit was not supposed to happen, this is a sign that the robot is performing an irrelevant step.

  To correct this problem, the teacher needs to signal the error to the robot within 10 seconds of detecting the problem. The teacher may also signal *before* allowing the robot to finish the execution of the wrong step, if from its behavior it is clear enough that this is in fact the robot's intention. The feedback signal is given by speaking the command **"BAD."** The robot will not execute this step in future performances.

- **Missing steps:** These are steps that the robot failed to perceive during the demonstration, and so it skips them while performing the task. The teacher detects this situation by observing that the robot skips steps and that it is instead trying to execute a step that should have been performed at a later stage.

  For example, let's assume that a correct task would be to visit a green, yellow, and then an orange target. If the teacher sees that after visiting the green target the robots starts going toward the orange target, this is an indication that during demonstration the robot failed to detect the visit to the green target. This is an example of a missing step.

  To correct this error, when seeing that the robot is trying to perform a step that should have been performed later in the task, the teacher may intervene by calling the robot with a **"COME"** command.

At that point, the robot starts searching for and following the teacher (similarly as during a normal demonstration, so the teacher should go in the front of the robot). The teacher can now demonstrate again the parts of the task that the robot had missed, as during a regular demonstration. The teacher ends the demonstration by speaking a **"GO"** command. The robot will continue executing the task from the point where it left off before the interruption. At the next task performance, the robot will include the newly demonstrated steps in its execution.

A.7 Putting it All Together

This section presents an example of teaching the robot a task, performed in the environment such as in Figure A.2. The task that the robot should learn is as follows: visit the **Light Green** target, pick up an **Orange** box, visit the **Light Orange** and **Yellow** targets, visit the **Pink** target, and then drop the box near the **Pink** target.

Here is the sequence of steps that the user should carry out to teach the robot this task (assuming that the robot is already turned on and ready):

1. Put on the headset (turn it ON only when giving commands).
2. Stand in front of the robot, so that you are within its visual field. Try to stay in the robot's visual field at all times during training.
3. Say "START". The robot will start coming toward you.
4. Move toward the the light green target. When you perceive that the robot can see the target (i.e, the target is within the robot's visual field) say "HERE".
5. When you get close to the box, pick it up, show it to the robot, say "TAKE", then put the box back down. The robot will pick it up on its own.
6. Move toward the light orange and yellow targets. The robot will keep following you. When you perceive that the robot can see the targets say "HERE".

7. Move toward the pink target. When you perceive that the robot can see the target say “HERE”.

8. After you reach the pink target say “DROP”. The robot will drop the object.

9. Say “DONE”. The robot will stop. This is the end of the demonstration.

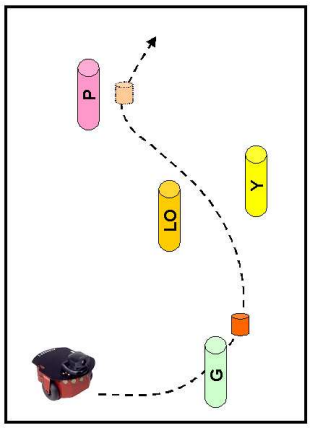


Figure A.2: The **Object transport** task