Goal and Action Inference for Helpful Robots Using Self as Simulator

by

Jesse V. Gray
B.A. Math & Computer Science, Tufts University, 2002

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author .....
Program in Media Arts and Sciences
August 6, 2004

Certified by .....
Dr. Cynthia Breazeal
Assistant Professor of Media Arts and Sciences
LG Group Career Development Professor
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by .....
Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Goal and Action Inference for Helpful Robots using Self as Simulator

by

Jesse V. Gray

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 6, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

New applications are bringing robots into environments where they will have the opportunity to cooperate with humans as capable partners. A crucial element of cooperation is the ability to infer actions and goals of another by observing them. The ability to understand the intention being enacted by another is very important for anticipating the needs of and providing timely assistance to them. This thesis presents an approach to building a robot that is capable of action and goal inference that is based on the concept of Simulation Theory (a dominant theory in philosophy for how people do this). Simulation Theory argues that we exploit our own psychological responses in order to simulate others' minds to infer their mental states. With respect to action recognition and goal inference this implies that the ability to perform an action helps one to recognize when the same action is performed by others. It further implies that a robot could leverage its own action/goal representation to infer goals of others based on their actions. This implementation addresses the task of acquiring perceptual data about the physical motion of another agent and the context in which is it performed and mapping it onto the robot's own perceptual and movement repertoire. This implementation then addresses how to achieve the simulation to recognize the actions and infer the goals of the observed agent based on this movement and perceptual data. I demonstrate this skill by having the robot exhibit behaviors that address the inferred goal being enacted by the human. In principle this approach could be extended to not only infer the intentions of others, but other mental states as well (motivations, emotions, desires, beliefs, etc.). The main contribution of this work is a plausible working model of simulation theory (informed by scientific studies of autism, imitation, and the development of theory of other minds) that is able to infer the intention behind observable action and its effects. This is an important step towards building robots that can begin to understand human behavior in terms of the mental states that generate it, rather than only upon observable surface behavior. This understanding is key for cooperating with robots instead of using them as tools.

Thesis Supervisor: Cynthia Breazeal
Title: Assistant Professor of Media Arts and Sciences

Goal and Action Inference for Helpful Robots using Self as Simulator

by

Jesse V. Gray

The following people served as readers for this thesis:

Thesis Reader .....

Bruce M. Blumberg
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Massachusetts Institute of Technology

Thesis Reader .....

Deb Roy
Associate Professor of Media Arts and Sciences
AT&T Career Development Professor
Program in Media Arts and Sciences
Massachusetts Institute of Technology

Acknowledgments

To all the people that made this thesis possible, thanks!

Many thanks to the members of the Robotic Life group, without whom there would be no robot and no thesis. Thanks and apologies to anyone who has replaced a motor that I "might" have burned out. Also special thanks to my advisor Cynthia for her support and insightful guidance of my research direction!

The Synthetic Character's group has been also been very good to me; they took me on as a young urop and introduced me to life at the lab. They also provided me with an excellent codebase to work in! Thanks especially to Bill for initially taking me on and making me feel at home, Bruce for all his caring support, Marc for his continuing invincibility, and to all my officemates, past and present. In fact, everyone I've had the pleasure of working with at the lab has been pretty exceptional, and I am constantly happy about it.

Thanks especially to Matt, who has always been looking out for me, and without whom I never would have been a urop at the lab. He has continued to look out for me since, both in the lab and outside, even down to providing essential advice and support with this very thesis!

Without the exceptional help and patience of Pat and Linda I would never have made it through this program, and I am grateful for it. Also thanks to Aileen, without whom a whole different set of things would never have gotten get done.

Thanks also to my thesis readers, Deb and Bruce!

And to Stan Winston Studios, thanks for providing such a nice furry robot to work with.

And of course, without my family, I would not be anywhere today. You guys did a great job of encouraging my interests while at the same trying (successfully!) to diversify them, sending me out along my current path (which I am enjoying!).

Contents

1 Introduction 11
- 1.1 Goal Inference 11
- 1.2 Approach 11
- 1.3 Architecture Overview 12
- 1.4 c5m Platform 13
  - 1.4.1 Perception System 13
  - 1.4.2 Belief System 14
  - 1.4.3 Action System 15
  - 1.4.4 Motor System 16
  - 1.4.5 Context Tree 16
- 1.5 Hardware And Sensing 18
  - 1.5.1 Robot 18
  - 1.5.2 Vision 18
  - 1.5.3 Speech 18
  - 1.5.4 Motion Capture 18

2 Related Work 21

3 Mapping 23
- 3.1 Facial Imitation 23
  - 3.1.1 Imitation in Humans 23
  - 3.1.2 Motor Babbling 26
  - 3.1.3 Intermodal Representation 27
  - 3.1.4 Detecting Contingency 28
  - 3.1.5 Organ Identification 28
  - 3.1.6 Representation of Novel Expressions 29
- 3.2 Body Movement Classification 30
  - 3.2.1 Mapping 30
  - 3.2.2 Acquiring Training Data 31
  - 3.2.3 Following the Human's Motion 33
  - 3.2.4 Segmenting Observed Motions 33
  - 3.2.5 Matching Observed Motions 33
  - 3.2.6 Limitations with this Matching 34

4 Segmentable Actions Representation 37
- 4.1 Segmentable Action Architecture 37
- 4.2 Conditions 40
- 4.3 Action Segments 41
- 4.4 Action Segments and Conditions Working Together 42
- 4.5 Relationship to Existing Architecture 42

4.5.1 Conditions as Action Groups 43
4.5.2 Action Segments as Action Tuples 43
4.6 Passing Parameters 44
5 Using Action Segments in Reverse 49
5.1 Matching Observations to Action Segments 50
5.1.1 Determining if a Context is Valid 50
5.2 Being Helpful 51
5.2.1 Verifying Human's Goal 51
5.2.2 Help On Failure of Human 52
5.3 Advantages 55
5.4 Design Choices and Limitations 55
5.4.1 Context Generator Modules 55
5.4.2 Limitations of Segmentable Actions 57
6 Demonstration 59
6.1 The Task 59
6.2 Implementation 59
6.3 Results 60
7 Conclusions 67
7.1 A More Complicated Interaction 67
7.1.1 Recursive Structure 67
7.1.2 Variables in Action Segments 68
7.1.3 Dialogue 68
7.1.4 Unhelpful Behavior 68
7.1.5 Multiple Action Segment Recognition 69
Bibliography 71

Chapter 1

Introduction

1.1 Goal Inference

Today, robots are largely considered to be complex tools operated by highly trained specialists; some, like assembly line robots, need initial programming to carry out a repetitive task, while others, such as planetary explorers, require the constant supervision of tele-operators. While requiring a trained operator to manage each robot may be feasible for these situations, as robots enter into the human environment and are expected to interact with an untrained user, they will need to serve as helpers and teammates for humans (care for the elderly, domestic assistant, etc.).

An important element of cooperative behavior is the need to understand a participant's actions and infer their goals from their external appearance and surrounding context. Experiments have show that human infants develop the ability to understand actions according to inferred goals as early as 6 months [25]. At 18 months they are able to imitate the goal of an unsuccessful action [15]. Understanding the actions of others in terms of their goals is a natural level of representation of behavior in humans. For instance, 3 year old children have been shown to imitate actions based on inferred goals rather than perceived movements [10].

Understanding actions in intentional terms through observation will enable the robot to see through surface level observable behavior to understand what the person is trying to do rather than just what they are physically doing at that very moment. This will allow the robot to provide assistance that is relevant to the person's goal, an important skill to participate in cooperative interactions with humans.

1.2 Approach

In order to approach the problem of action recognition and goal inference, I am using ideas

drawn from the development of this capability in human children. Simulation theory holds that certain parts of the brain can have a dual use; they are used to not only generate our own behavior and mental states, but also to predict and infer the same in others. More specifically, to understand another person's mental process, we "step into their shoes" to determine the consequences. That is, we use our own similar brain structure to simulate the thoughts of the other person [5].

There is a debate in developmental psychology with respect to our "mindreading" competence between the proponents of theory-theory (that propose a separate abstract system for behavior prediction) and simulation theory, which is grounded in first hand experience by taking the perspective of others. This work does not address this debate, but it provides a working model for how a simulation theoretic system could infer from external perception the goals being enacted by others by utilizing its own action and motor production systems. I also believe that simulation theory offers a computationally efficient and elegant solution for implementing this capability; not only does it simplify certain parts of the implementation allowing for re-use of functionality between producing behavior and predicting it, but it also grounds everything the robot can understand about other people in the actions it can do itself.

This implementation describes a system for representing these actions in a structured way that could eventually be used to learn new actions of this type. Learning these new actions would be powerful, because they could be used for both accomplishing the particular task and detecting that same behavior in a human. However, learning new actions of this form is not yet addressed in this thesis, only the use and recognition of pre-existing actions is covered.

1.3 Architecture Overview

My implementation for creating a simulation theory based system breaks down into three main parts. First, when designing the action system of the creature, it must have enough

structure that it is possible to determine the relationships between the motions, actions, and goals of the robot. Next, the robot must be able to use that structure to determine which, if any, of its own actions is the same as the action the human is performing. To do this, the robot has a mechanism (see Chapter 3) to perceive the movement trajectories of the human in the same representation that its own movements are stored in, so it can determine if any of its own movements match the one the human is performing. It also has additional perceptual information to further determine what action is being performed given the situational context (see Section 5.1). Finally, it has a mechanism for using the structure of the action system to introspect over the system. This mechanism can determine the goal of the human, once their behavior has been classified as one of the robot's own actions, and determine how to best help them with their goal.(see Chapter 5).

1.4 c5m Platform

The software described in this thesis exists as part of the c5m system (based on the C4 system, [1],[4],[7]) system, an architecture created by the Synthetic Characters Group (and now shared with the Robotic Life Group) for designing virtual creatures from perception to motor production. Originally designed with animated creatures in mind, it has been adapted to also render out movement data to robots, allowing the two groups to share the system. In this section, I will give a brief introduction to a number of key parts of this system that will get mentioned throughout this thesis.

1.4.1 Perception System

The Perception System in c5m is the gateway for all external data that the creature can perceive. It is responsible for handling all incoming sensor data from many types of sensors (see Section 1.5) and converting it into a common format for the other systems to work with.

The Perception System is constructed out of many individual Percepts. Inspired by the human perception system, each Percept is designed to fire in response to one particular type of

input. For example, one percept might be responsible for detecting if incoming data refers to an object that is red. When that percept is exposed to data about a red object, it will result in a 100% match. Data of another color will result in a 0% match.

The percepts are arranged hierarchically according to their specificity so that the most general percepts are higher in the hierarchy, with the specificity increasing towards the leaves. A percept can opt not to pass data to its children if the data is irrelevant for that branch of the tree. So for example, a typical percept tree would distinguish between auditory and visual data at the top level of the tree. The red detector percept mentioned above, then, would be an ancestor of the percept for detecting visual objects, so it would never be exposed to irrelevant auditory data.

Each piece of incoming sensor data is pushed through the tree and presented to all the relevant percepts. Each of these percepts makes an evaluation, and these evaluations are collected into one structure. Once all the sensor data has been converted into these collections of percept evaluations, it is the job of the belief system to use these collections to maintain persistent information about what objects exist in the world and what properties they have.

1.4.2 Belief System

The Belief System has the responsibility of maintaining persistent knowledge about the current state of the world. Each object the robot knows about is represented by a structure called a Belief. A Belief consists of a time history of Percept evaluations about one object in the world.

As new collections of percept evaluations are produced by the Perception System, the Belief System must determine which, if any, of the existing Beliefs these collections refer to. For this it employs multiple merge metrics. A commonly useful merge metric is the physical distance between objects, augmented by their types. Two sensor readings about a human within a short spatial distance are likely about the same human. However, if one object is clas-

sified as a ball and the other as a human, they are unlikely to be both part of one object even if physically very close. If the Belief System determines that a collection of new evaluations refers to the same object as an existing Belief, that Belief is updated to contain the new evaluations. If not, a new Belief is created to represent the object specified in the new evaluations. Beliefs that have not been updated for some time can be culled; this can help to eliminate Beliefs that were constructed in error or refer to objects no longer present.

1.4.3 Action System

The core of the creature's behavior is the Action System. The Action System is responsible for using the perceptual data as cataloged through the Belief System, as well as other internal state, to generate the behavior of the creature.

The main components of the Action System are Action Tuples and Action Groups. An Action Tuple contains a number of elements used for action arbitration, such as a "Trigger" module which indicates when the Action Tuple should become active and a "Do Until" module that indicates when it should stop being active. Each Action Tuple also contains an element that specifies what the Action Tuple does while it is active. For Example, the simplest Action Tuples generally request some type of motion to be carried out while they are active.

Action Tuples are each contained within an Action Group. The Action Group has the responsibility of determining which Action Tuples should be active and for how long. It uses the information provided by the Triggers and Do Untils to aid in its decisions. A simple Action Group implementation, then, is to activate Action Tuples whose Triggers are active and deactivate those whose Do Untils indicate that they are done. A more complicated Action Group often used in c5m creatures is the Probabilistic Action Group. This group ensures that only one action runs at once, and arbitrates between the Action Tuples in a more sophisticated manner by taking into account how low each has run, and how various Trigger and Do Until values have changed recently.

Action Groups can also be embedded inside Action Tuples. When an Action Tuple with an embedded Action Group becomes active, its effect is to run its Action Group and allow it to arbitrate between its own Action Tuples. In this manner, hierarchical systems can be achieved.

1.4.4 Motor System

The Motor System in c5m is based on a pose-graph structure. This structure is a directed graph of poses, where each pose is a potential joint configuration for the creature. The possible movements of the creature, then are paths through this structure.

For many types of motions, it is desirable to have a continuous range of motion possible. For example, a character may want to perform a motion in a happy manner, a sad manner, or anywhere in between. Another example is for more precise movements, where a character might want to point to an object wherever it is without having thousands of different pointing motions. For this reason, poses in the posegraph can also be blended poses instead of regular poses. A blended pose, instead of containing just one possible joint configuration, contains multiple joint configurations. Each time a blended poses is encountered in the posegraph, these multiple configurations are blended together to form some intermediate pose based on parameters presented to the motor system.

The pose-graph has multiple possible viewing resolutions. Described above is the pose-view, where nodes in the graph are individual poses (joint configurations) of the creature. There is a higher level view available, the movement-view, where each node in the graph is one logical movement of the creature. Systems can examine the graph in either view, and the current position in the graph is available as both a pose and a movement to which it belongs.

1.4.5 Context Tree

This is not one of the major systems in c5m, but instead part of the glue that ties them together. The Context Tree is a hierarchical runtime scoping system. The structure of this

hierarchy is determined by balanced calls to begin-context and end-context. These calls can be nested, such that one scope is contained inside another. A context is entered by calling begin-context, and everything that is done after that call and before the corresponding end-context occurs in that context. Each context provides a mutable mapping from arbitrary keys to values. These mappings are inherited from parent contexts; if a key has not been set in a child context lookups of that key will return the parent's value.

The hierarchy is generally arranged such that each creature has a context, and each system with in that creature has a subcontext. Subsystems within those systems each operate in a subcontext, and so on. This allows flexibility with parameters and state of the creature. For example, information that might be globally interesting can be installed at the creature level, so it is available to all subsystems. Specific subsystem, however, may override this information for themselves or for their children.

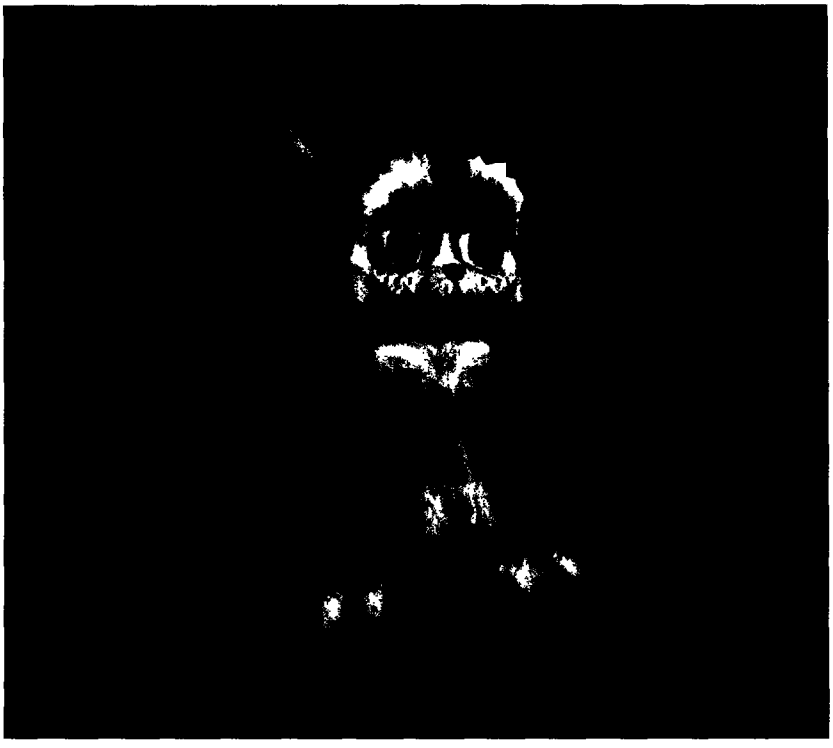

Figure 1.1: The c5m system can control either virtual Leo or real Leo

1.5 Hardware And Sensing

1.5.1 Robot

My implementation runs on the Leonardo robot (Leo), a 63 degree of freedom robot provided to the group through a collaboration with Stan Winston Studios. Every update cycle, joint angles are calculated based on the position of virtual Leo's skeleton. These angles are upsampled to a higher framerate (to provide smooth motion on the robot), then sent over the network to the motor controller computer, which in turn relays them to the motor controller boards.

1.5.2 Vision

Leonardo sees the world through 2 environmentally mounted stereo vision cameras, and 2 eye cameras. One stereo vision system is mounted behind Leo's head, and detecting humans standing near Leo. The second stereo vision system looks down from above, and is used to detect objects in Leo's space as well as human hands pointing to these objects. Leo can use his eye cameras for fine corrections to look directly at objects. Leonardo also has access to data from the Axiom ffT system. This system uses one camera to track the features of a human face. The system provides data as the 2 dimensional image coordinates of various facial features, such as the inner corner of the eyebrow or the top of the mouth.

1.5.3 Speech

For speech recognition, Leonardo is using the Sphinx system developed by Sun Microsystems. We have configured the system with the vocabulary and grammar used in our interactions, and it provides very good recognition.

1.5.4 Motion Capture

In order for Leonardo to perceive the pose and motions of a human interacting with him, we use a Gypsy motion capture suit. This suit has potentiometers mounted near the joints of the human to determine their joint angles. This suit is worn by the human interacting with Leo (For a picture see figures at end), and it constantly transmits their current joint angles to

Leonardo. In the future, we may be able to replace the use of this suit with a vision solution to allow for more natural interactions.

Figure 1.2: Leo The Robot with Fur On

Chapter 2

Related Work

The movement classification portion of this system is based on the idea of sharing a representation between the perception and production of body poses. This idea of sharing a representation between one's own motor actions and the observed behavior produced by others has been discussed in the fields of philosophy [11], developmental psychology [17], autism [9] and neuroscience [21],[23]. Mirror neurons are seen as possible evidence for this shared representation, as they have been shown to fire similarly in both cases [23]. The discovery of mirror neurons is also possible evidence for Meltzoff and Moore's [20] Active Intermodal Mapping Hypothesis (AIM), which proposes a modality independent representation common to perception and production. This shared representation is thought to be a starting point for a simulation theoretic process of inferring goals and intentions in humans [17].

Mapping demonstrated human movements into the robot's motor space has been discussed in Programming by Demonstration (PdB) applications [13],[8]. PdB is used as a programming shortcut for quick, intuitive training of robots using human demonstration of tasks. This work focuses on motor skill acquisition rather than detection of known motor skills. In [14], Lieberman explores mapping movements into the space of a robot and detecting boundaries between actions, but from the perspective of teaching new motor skills.

A number of motor learning efforts for robotic systems have looked to mirror neurons for their biological inspiration [24],[6]; some even use simulated mirror neurons to recognize movements of both the robot and the human while interacting [26]. These systems examine motor learning and motor trajectory recognition, but have not yet addressed using these abilities to infer hidden mental information about the human. They do not go so far as to infer the intended goal or other mental states that are responsible for generating the observed action.

Some similar work is going on inside the same code base; [3] describes a simulation theory based system for action inference similar to this one, also in c5m. Buchsbaum's thesis is built upon similar foundations [2], but has a slightly different focus (Understanding the actions of others in terms of your own to infer properties of unknown objects). In other related work, the task of determining what goal directed action could produce a given movement was also necessary in [1], but for the purpose of distributing a reward for that movement to the appropriate causal action.

Chapter 3

Mapping

This chapter addresses finding which motion of the robot's the human is currently performing. This problem of mapping an observed human movement into the robot's repertoire is the same problem as motion imitation, so first work in facial imitation will be covered. Later in the chapter the system developed for facial imitation will be applied to recognition of body movements.

3.1 Facial Imitation[1]

In order to solve the problem of imitation, the robot must be able translate between seeing and doing. Leonardo sees the facial features of the human using the Axiom ffT facial feature tracker (see Section 1.5.2). In order to imitate, the robot must be able to determine the correspondence between that configuration observed in the human and its own facial configuration, and adopt the configuration of the human. To solve this problem, we look to human infants.

3.1.1 Imitation in Humans

According to Meltzoff [16], "human parents are prolific imitators of their young infants." Caregivers continually shadow and mirror their infant's animated movements, facial expressions, and vocalizations. In turn, infants seem to recognize when their behavior has been matched. They preferentially attend to adults whose actions are contingent on their own, and especially to adults who are imitating them [18]. Specifically, they seem to recognize both temporal contingency (i.e., when the infant performs action x, the adult performs action y, where x and y differ in form), as well as structural congruence (i.e., when x and y have the same form).

1. This section draws heavily on [2], written earlier about this same work

This early imitative capability continues to develop over time. Meltzoff suggests a four-stage progression of imitative abilities (for a review, see [16] and [22]). The first stage is called body babbling (analogous to vocal babbling) that involves random experimentation with body movements in order to learn a set of motor primitives that allow him to achieve elementary body configurations.

Next, the infant is able to imitate body movements. Just hours and even minutes after birth, infants can imitate facial acts that they have never seen themselves perform. This suggests an innate mapping between the observation and execution of movements in humans. It has been shown that 12 to 21 day old infants can identify and imitate the movement of a specific body part and imitate differential action patterns with the same body part [20]. This is called organ identification.

At 6-weeks, infants have been shown to perform deferred imitation from long-term memory after seeing the target facial act performed 24 hours earlier [19]. They are able to correct their imitative response in a goal-directed manner from memory without requiring any feedback from the model. This presents further evidence that observation-execution pathway is mediated by a representational structure.

Meltzoff argues that this structure is represented within an intermodal space into which infants are able to map all expressions and movements that they perceive, regardless of their source. In other words, the intermodal space functions as a universal format for representing gestures and poses - those the infant feels himself doing, and those he sees the adult carrying out. The universal format is in terms of the movement primitives within his act space. Thus the perceived expression is translated into the same movement representation that the infant's motor system uses making their comparison much simpler. The imitative link between movement perception and production is forged in the intermodal space.

In order to successfully imitate, one must locate and recognize the facial features of a demonstrator, find the correspondence between the perceived features and one's own, and be able to move one's features into the desired configuration. Meltzoff and Moore [20] proposed a descriptive model for how an infant might accomplish this task known as the Active Intermodal Mapping Hypothesis (AIM). A schematic of the AIM model is presented in Figure 3.1. In general, the AIM model suggests that a combination of innate knowledge and specialized learning mechanisms allow infants' to imitate in a cross-modal, goal-directed manner. Specifically, AIM presents three key components of the imitative process: motor babbling, organ identification, and the intermodal space. Taken together, this model suggests mechanisms for identifying and attending to key perceptual features of faces, mapping the model's face onto the imitator's, generating appropriate movements, and gauging the correspondence between produced and perceived expressions. This model has been a guide for the implementation described here.

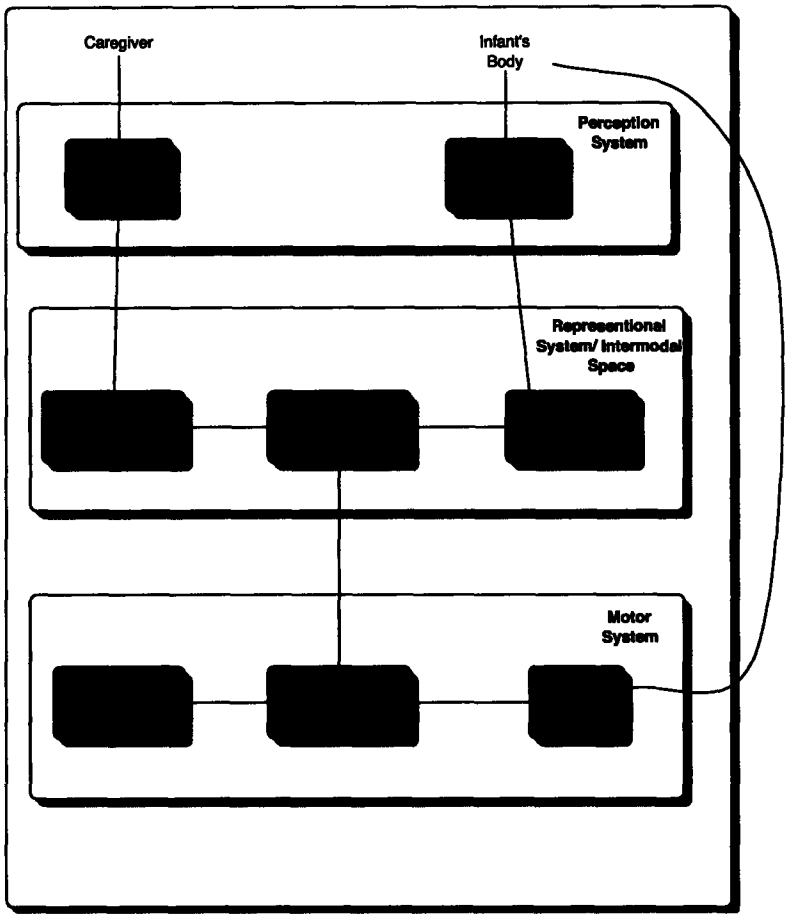

Figure 3.1: AIM Model. Meltzoff's AIM model describes a method by which infants may learn to imitate through motor babbling and feedback from caregiver

3.1.2 Motor Babbling

Similar to the motor babbling exhibited by infants in the AIM model to physically explore their motor space, Leonardo employs a motor babbling action to cause the robot to physically explore its pose space. While Leo's motor babbling action is active, it randomly selects a pose from the basis set used to create its posegraph, requests that the motor system go to that pose and hold it for a moment (approximately four seconds), and then selects a new pose. While Leonardo is motor babbling, the human participant tries to imitate Leo's facial expressions.

Leo performs this Motor Babbling action to help the robot learn to map perceived human expressions onto an intermodal space, like the one used by infants in the AIM model. By

detecting when the human participant is likely to be imitating him, Leonardo can use its own pose (generated through motor babbling) and the human's imitation of this pose to improve the robot's ability to map the human's facial expression to its own intermodal space.

3.1.3 Intermodal Representation

According to Meltzoff's model, infants use the same internal representation for their own expressions and those they see an adult perform. Furthermore, this representation is the same one used within the infant's motor system to describe how the infant must move in order to achieve a given expression. As such, this representation bears strong resemblance to the function of mirror neurons [17]. The intermodal representation allows the infant to discover correspondences between his own expressions and those of the human model, by providing a format in which they can be directly compared.

In our motor system, Leonardo's expressions are represented as poses, which are sets of joint angles (see Section 1.4.4). We chose to use these poses in Leonardo's own joint space as its intermodal representation since it is the native format for his movements. Therefore, the human expressions that Leonardo perceives must be mapped from the set of two-dimensional absolute coordinates provided by the facial feature tracking software onto the robot's joint space. This process is complicated by the fact that there is not a one to one correspondence between the tracked facial features and Leo's joints. To solve this problem, Leonardo learns the intermodal representation from experience while the human participant is imitating the robot. This is a rough analogy to learning mirror neurons for encoding and representing perceived movement in terms of motor primitives [21]. The robot models the intermodal map using a separate neural network for each facial region corresponding to the right eye, left eye, and mouth (see section 3.1.5).

3.1.4 Detecting Contingency

In order for Leo to successfully train the neural nets, the robot must provide the networks with example input-output pairs. Within the framework of the imitative interaction, one way for Leonardo to acquire this data is for the robot to identify when the human participant is imitating it, and to then store a snapshot of the current facial feature data and the robot's own current joint configuration. Unfortunately, before the neural networks are trained, Leo cannot detect an exact correspondence between the human's facial features and its own pose. Identifying when the robot is being imitated is tricky at this stage.

The literature on infant imitation indicates that infants are especially responsive to adult movements that appear to be contingent on their own. Similarly, Leonardo determines when a person is imitating Leo contingently based on the elapsed time (less than a couple of seconds) between the start of Leo's movement and the human's response. To avoid false positive detections of human movement due on sensor noise, thresholds for human movement were set per dimension relative to the standard deviation of data for that dimension. In addition, the human's movement must be surrounded by a few seconds of stillness, so as not to classify constant motion as contingent. Some error is still possible with this metric; for instance, if the human moves contingently but is not imitating Leo. Overall, however, we found that using contingent motion to detect imitative interactions produced more accurately trained neural networks.

3.1.5 Organ Identification

We found that during the training process, people often only imitate a particular region of the robot's face (e.g., the mouth, eyebrows, etc.) rather than the robot's entire expression. For instance, the human may choose to only imitate Leo's mouth, in which case the rest of their face provides irrelevant data for the training of their respective regions. To address this issue, we partition the incoming facial feature data and Leo's degrees of freedom into three indepen-

dent groups of features that are handled separately: the left eye/eyebrow area, the right eye/eyebrow area, and the mouth. The data from each facial region of the human's face is collected using three separate contingency detectors. These groupings allows Leo to start with a rough idea of which of its organs correspond to those of the human participant, an advantage the AIM model proposes infants share.

Inside each area, the exact relationship between the coordinate data from the facial feature tracking software and the joints in Leo's face is not yet known and must be learned individually using separate neural networks. For each, we used a two-layer network, with 7 hidden nodes (7 was established to be a good number after we varied it for several tests). The inputs to the networks are the relevant degrees of freedom from the Axiom ffT data (see Section 1.5.2): the x and y positions of facial features, normalized to be invariant to the scale of the face, facial translation, and rotation. The outputs are the angles for relevant joints in Leo's face. Each joint in the virtual robot is restricted to one degree of freedom of rotation, just as the motors in the actual robot are.

3.1.6 Representation of Novel Expressions

Once the separate neural networks are trained, they are able to take as input the data from visual perception of a human expression, and output the intermodal representation of that expression in terms of the robot's joint angles. The separation into facial regions has an important advantage: Leo can create an intermodal representation of the human pose separately for each group of features. This allows him to generalize and create overall expressions that may never have been in the babbling set. For example, if none of Leo's babbled poses have asymmetric eyebrows, a neural network for the entire face would never allow him to create an intermodal representation with one cocked eyebrow. With this method, however, the eyebrows each respond separately to produce a representation of the novel facial expression.

After mapping a novel expression into Leo's joint space, it is important take the next step and match that joint configuration against one of Leo's expressions. This provides higher level data to the system; it is more useful to say that Leo is imitating "happy" than to record the list of all the joint angles.

In this implementation, Leonardo could actually determine the blend of his expressions that best approximated the observed facial expressions (using the sum of joint angle errors as a distance metric). This allowed him to represent novel facial expressions as a set of weights defining the expression somewhere in the space of his existing expressions. See [2] for details.

3.2 Body Movement Classification

Currently Leo perceives data about the human's position via a motion capture suit (see Section 1.5.4). The data from the gypsy suit appears to be much more suitable for direct mapping into the robot's pose space than the data from the Axiom ffT, since it is providing action joint angles. In fact, the gypsy suit can produce accurate joint angle data; however it relies on an extensive calibration procedure (30 minutes for a full calibration, 10 for a rough one if the person doing the calibration is experienced, according to the product literature) and requires the operator to be careful to put the suit on the same way the each time. Also, the suit does not have exactly the same geometry as Leo, so even if it is producing perfect data it is not possible to use the measured joint angles accurately directly in Leo's coordinate space. Because of these issues, we elected to forego the manual calibration that accompanies the suit software, and instead use the raw suit data and have Leonardo learn the mapping from that raw data into his intermodal space in a similar manner to the facial mapping.

3.2.1 Mapping

For the work in facial imitation, we used a neural network to map the facial pose data onto Leo's facial joints. For this body mapping task, we used radial basis functions to produce the mapping. An RBF model consists of a linear fit to the data combined with a non-linear

portion which accounts for residual error by interpolating amongst the training examples. These two components correspond well with our data set, which is dominated by a linear mapping between joint angles but with occasional but localized regions off nonlinearity. We initially tried RBF's instead of neural nets because they are easier to work with (the computational overhead of training is less, and they can learn a fairly accurate linear approximation given even very sparse training data). Since they worked well at modelling the mapping, we continued to use them.The data from the Gypsy suit is fairly close to the destination format of Leo's joint space; there is a one to one correspondence between Gypsy joint and Leo's joints. For this reason, we were able to train the radial basis functions on the error of the default mapping, instead of using them to do the entire mapping, which gave us better results.

3.2.2 Acquiring Training Data

Here we built directly on the work done for training the mapping for facial imitation. Leo acquires the training data by motor babbling while the human is engaged and imitating. Certain lessons learned from the facial imitation system became useful here. First, we found that when trying to imitate the facial expression of Leonardo, it was difficult to imitate the entire face at once. Because of this, we broke the babbling of the body joints into small groups, including the two arms and the torso. In order to keep the human focussed on the important area, the robot babbles through each zone separately. Before starting a new zone, the robot wiggles the joints involved so the human can tell what joints are going to be important.

Figure 3.2: Leo imitates human showing off his right arm mapping immediately after training session of human imitating Leo

In our tests, the robot only requires one each of about 15 poses to train up the 6 degree of freedom arm mapping, and one each of about 9 poses to train the 2 degree of freedom torso mapping. These poses go by quickly, so the whole mapping can be calibrated in just a few minutes, and without any particular expertise.

Another nice feature of this method for acquiring training data is the intuitiveness of the resulting mapping. While it may not be perfectly clear to an observer which poses of the human should match which poses of the differently proportioned Leo, the trainer has implicitly made those choices during the imitative interaction. By attempting to imitate each of Leo's poses, the trainer has specified what they feel the correlation between their body and Leo's is; thus they will likely not be surprised by the resulting mapping.

In part because imitating arm motions turned out to be relatively easy compared to prolonged facial imitation, and in part due to the increased difficulty of applying the contingency detector (Section 3.1.4) in this more complicated context, this training process doesn't use the

contingency detector strategy from the facial imitation (However, I would like to see this happen in the future). Instead, after the robot displays which joints are currently important with a wiggling animation, it assumes that the human is moving all of those joints.

3.2.3 Following the Human's Motion

Once Leonardo has successfully trained a mapping from the incoming motion capture data into his own joint space, he can keep track of the motions of the human. He does this by keeping another copy of his skeleton that follows along with the human's movements. This allows him to either imitate the motions of the human, or just follow along and observe their movements.

3.2.4 Segmenting Observed Motions

The motion stream of the human is segmented based on pauses in motion of the end effector. End effector position is obtained using geometry of the secondary skeleton that Leo is using to represent the position of the human. As the end effector moves, the segmentor builds up a statistical model of its tick-to-tick displacement. Pauses are detected when the movement of the end effector drops below a threshold derived from this statistical model for a small number of ticks. The segmentor creates movement chunks consisting of all of the end effector movement data between pauses, as well as between the most recent pause and the current time. These chunks are then handed to the matcher to be matched against movements in Leo's repertoire.

3.2.5 Matching Observed Motions

Because we do not expect the movements of the human to closely match those of the robot, we chose a very coarse matching metric. The matcher operates by using a representation of end effector movement based on Movement Axis Model [12]. For any given sequence of end effector positions, a Movement Axis Model can be created, consisting of the average position of the end effector, as well as the direction and length of the "major axis" of move-

ment of the end effector. The "major axis" of the movement was computed simply by finding the two end effector positions in the movement that were furthest away from each other; the vector between these two positions was said to be the axis of the movement. The Movement Axis Model for each movement in Leo's repertoire can be computed just once and then stored for later use.

When a new observed chunk of movement was handed to the matcher by the segmentor, the matcher first computed the Movement Axis Model for the observed movement. Once computed, this new Movement Axis Model was compared to each of the Movement Axis Models for the animations in the high-level posegraph view (see Section 1.4.4). The distance between any two Movement Axis Models was computed as the sum of three quantities: (1) the distance between the average end effector positions, (2) the difference between the lengths of the major axes, and (3) the angular distance between the two major axes, multiplied by the average length of the two major axes. These three quantities were chosen because they are comparable: all three represent distances in space, and for typical arm movements, all three vary over similar size scales. Every time a new observed motion segment is tested (every update cycle while a motion is occurring), the animation of Leonardo's that is closest to the motion based on the Movement Axis Model test receives one vote. If one animation has a 2/3 majority after 20 or more observations are made, it is declared an actual match for the observed motion. Otherwise it is discarded and considered to be a motion that is not in Leonardo's repertoire. All votes are reset when the human stops moving according to the segmentor.

3.2.6 Limitations with this Matching

The metric used here is very simple, but sufficient for distinguishing between the limited number of movements necessary to create a simple example of the rest of this system operating (see Chapter 6). It is not intended to be a general purpose motion matching algorithm, and in order to extend this system to more animations, this metric will likely need to be revis-

ited. There are many other mechanisms used for comparing two motions for similarity (for example, HMM's) that could potentially be employed here for greater accuracy in the future.

The segmenting method described here also limits what types of motions can be recognized. The motions we have used have all started and ended with pauses, but that is not true about motions in general.

Chapter 4

Segmentable Actions Representation

This chapter introduces an architecture for specifying simple goal directed actions that the robot can perform. The next chapter describes how this architecture can also be used, in conjunction with the motion data computed in the previous chapter, to interpret the actions of an observed human in terms of the robot's own actions in order to provide helpful behavior. Classifying the observed behavior of the human as one of the robot's goal directed actions instead of as a movement trajectory can provide the robot with a higher level understanding of the human's activities, including a guess at their current goal. The architecture described in this chapter is designed to maximize the usefulness of classifying the human's behavior as one of the robot's goal directed actions, while also being flexible and convenient to use to create the robot's action system. This chapter focuses on the architecture from the perspective of creating and performing the robot's own actions. The next covers how it is used to classify human's actions and then using the result to assist the human in accomplishing the goal.

4.1 Segmentable Action Architecture

Segmentable actions are composed of a hierarchy of Action Segments and Conditions. Action Segment modules perform a task (intended to be atomic, on the level of a single motion), while Condition modules provide an evaluation about some status of the world or the creature's internal state. A Condition that follows an Action Segment is a goal of that segment - it is expected that after the completion of that segment, the Condition will evaluate to true. A Condition that precedes an Action Segment is its precondition - it is necessary that that Condition evaluate to true before the action is attempted (See Figure 4.1).

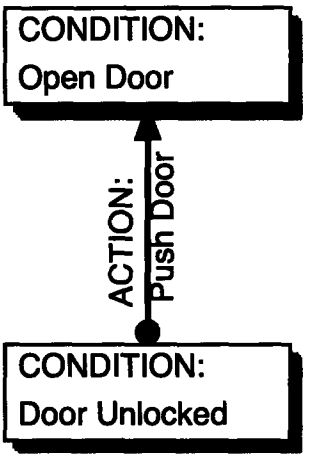

Figure 4.1: Simple Action Segment with precondition and goal

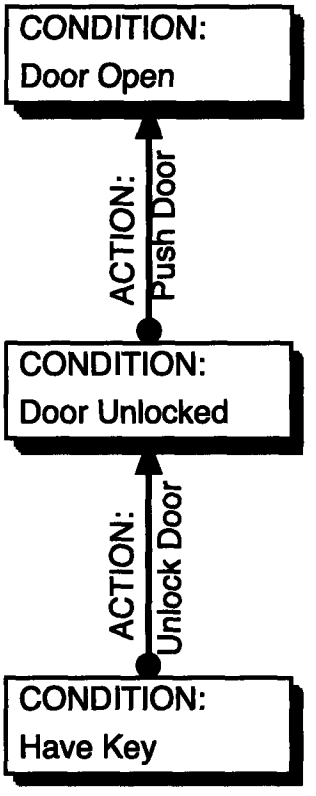

Figure 4.2: Compound Action Segment. A Condition can be the goal of one action and the precondition of another.

This relationship between Action Segments and Conditions allows larger structures to be created where the goal of one Action Segment becomes the precondition of a later Action Seg-

ment (See Figure 4.2). This allows a complicated task to be specified as a hierarchy of very simple Actions Segments and Conditions (see Figure 4.3). To achieve some desired state, only the desired Condition need be activated, and the system will traverse the hierarchy completing preconditions and Action Segments as necessary until the Condition is fulfilled (or it becomes known that the Condition cannot be fulfilled). It is the responsibility of each Action Segment to ensure its preconditions are achieved, and of each Condition to activate the appropriate Action Segments that cause it in turn to become valid. While there is a standard mechanism for each to accomplish this by traversing the hierarchy (described in the following sections), this delegation of responsibility means that a special kind of Action Segment or Condition that must use a different strategy can easily be accommodated. The following sections address more specifically the two main elements that make up these action hierarchies: the Action Segments themselves and the Conditions that connect them. Section 4.5 describes the relationship between these concepts and the c5m Action System.

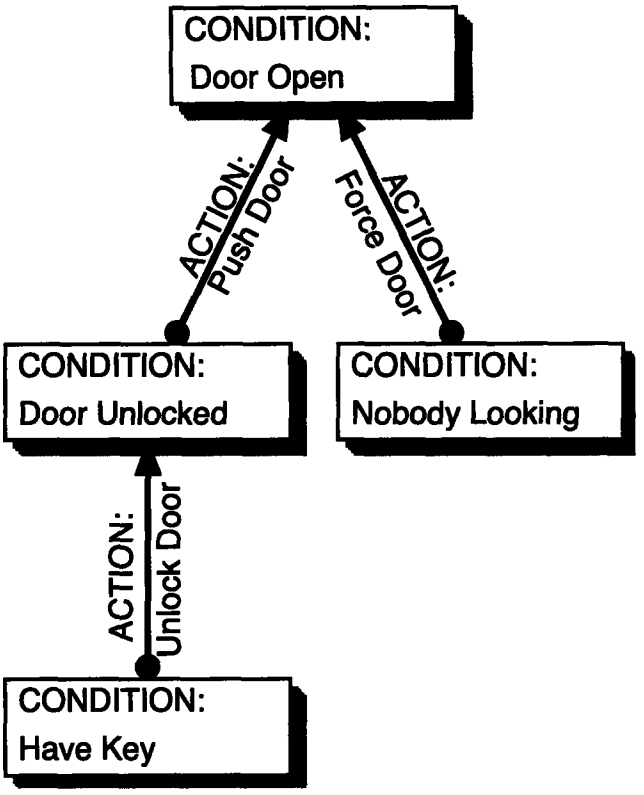

Figure 4.3: Hierarchy of Action Segments for achieving "Door Open"

4.2 Conditions

A Condition consists of two parts. The first is a wrapper common to most Conditions, and the second is the actual evaluation mechanism. The conditions have two modes of operation. They can be queried, in which case they simply respond with whether or not they are currently met. They can also be activated, in which case they try to achieve themselves.

The wrapper of a standard Condition provides it with a list of Action Segments that can be used to bring it about. When the condition is queried about its state, it simply delegates to the evaluation mechanism to report whether or not it is currently met. However, when it receives a request to become valid, it uses these Action Segments to attempt to become so. The default mechanism is to try the Action Segments one by one, checking the evaluation mechanism each time and stopping once the evaluation mechanism reports that it is achieved. This

process can also terminate in failure if the action list is exhausted without achieving the condition.

The evaluation mechanism can vary; however the most common type of evaluator relies on the Belief System described in Section 1.4.2. Since the Action Segments are often physical motions, the preconditions and goal Conditions can often be expressed in terms of objects in the environment. For example, turning a button on could have a precondition that the button is off (expressed as a percept evaluation stored in the belief which represents that button), and a goal Condition that the same button is on. The Conditions can also receive optional parameters. In the case of achieving the button-on Condition, the process that activates this Condition must provide the information about which button should be on. (see Section 4.6 for argument passing)

4.3 Action Segments

Like the Condition, the Action Segment consists of a wrapper, common to most Action Segments, and an embedded action.

The wrapper of the Action Segment is very much like the wrapper of the Condition. This wrapper contains a list of preconditions for the action. Whenever it is requested that this Action Segment be performed, it first verifies that all its preconditions are met. Any Conditions that are not met are activated. If the Action Segment can achieve all of its preconditions by activating them, it will do so and then activate itself. If any of its preconditions fail to complete, the Action Segment will terminate in failure and it will not activate its wrapped action.

The wrapped action within the Action Segment structure is often a single motion performed by the robot, although it could be something more sophisticated provided by the creator of the Action Segment. Oftentimes these motions have additional parameters, such as where or on what object the robot should perform the motion. The Action Segment is respon-

sible for providing these extra motion parameters to the Motor System, and it in turn must be provided with any data it needs to compute these motor parameters. (see Section 4.6 for argument passing)

4.4 Action Segments and Conditions Working Together

Using the properties of Conditions and Action Segments, it is possible to create a multi-stage hierarchy out of simple elements. Whenever a Condition is activated, that condition will activate one of its Action Segments to bring itself about. That Action Segment, however, may have its own preconditions that will need to be activated, and each can initiate their own Action Segments to cause themselves to be brought about. The flow of control will continue to move down the hierarchy until an Action Segment with no (or already true) preconditions is reached, or until a false precondition with no Actions Segments to help it is reached. In the successful case, flow of control will slowly move back up the hierarchy as each Action Segment achieves its goal Condition, enabling further Action Segments whose preconditions were that goal. Eventually the final Action Segment, the one which should enable the originally activated Condition, will have its preconditions met and it will be able to run.

4.5 Relationship to Existing Architecture

The system described here is designed to work as a component of the c5m Action System, introduced in Section1.4.3. The c5m Action System is already good at representing hierarchical action structures through using Action Groups that contain Action Tuples, and Action Tuples that can in turn be Action Groups. The additions described in this chapter are designed to act as specialized Action Tuples and Action Groups that continue to allow for powerful hierarchical action structures, but are constrained enough that they will support the programmatic introspection required by the next chapter.

4.5.1 Conditions as Action Groups

Conditions are designed as a specific type of Action Group. In addition to the normal Action Tuple arbitration duties of an Action Group, the Condition also encapsulates a mechanism for evaluating some kind of state (either in the world, or an internal state of the creature). The Condition is then both a detector of some state, and, through its included Action Tuples, a mechanism for bringing about that state. As the Condition is designed to use its included actions specifically for the purpose of bringing itself about, its default arbitration scheme is different from other types of Action Groups. It also expects that all of its included actions are actually Action Segments and depends on features of Action Segments (their ability to end themselves, and relay their success or failure). Another difference between Conditions and standard Action Groups is that the Condition is responsible for remapping any current parameters of the ongoing compound action to be relevant to its children, as described in Section 4.6.

4.5.2 Action Segments as Action Tuples

Action Segments act as a specific kind of Action Tuple, but they present a slightly different interface. Instead of a Trigger condition which governs when an Action Tuple should begin, the Action Segments have a "Cannot Finish" mechanism. This mechanism is used like the Trigger, in that an Action Segment that cannot finish will never start. It is also used, however, throughout the life of the action to indicate failure. The Action Segments also have a mechanism similar to the Do Until of regular Action Tuples. This mechanism, however, is always controlled by the Action Segment itself (a normal Action Tuple's Do Until might depend on some other environmental factors, rather than being determined internally); the Action Segment uses this and the Cannot Finish mechanism to differentiate between success and failed attempts at the action; this information can be used by the Condition when it is using its Action Segments to try to bring itself about. Finally, Action Segments also can have a set of

Conditions which represent preconditions necessary to achieve before the Action Segment is attempted. Because they can contain a set of Conditions to run, they are share some properties with Action Groups as well.

4.6 Passing Parameters

Frequently, Action Segments and Conditions need some kind of parameters in order to determine their function. Although other parameters may be necessary, the most common parameter specifies which object the Action Segment or Condition is operating on. Action Segments and Conditions also sometime need to modify that set of parameters along the way in order to have their children operate properly (see Figure 4.4 for an example of a compound action where each segment needs modified parameters). These arguments are provided through a mechanism called the Context Tree (see Section 1.4.5).

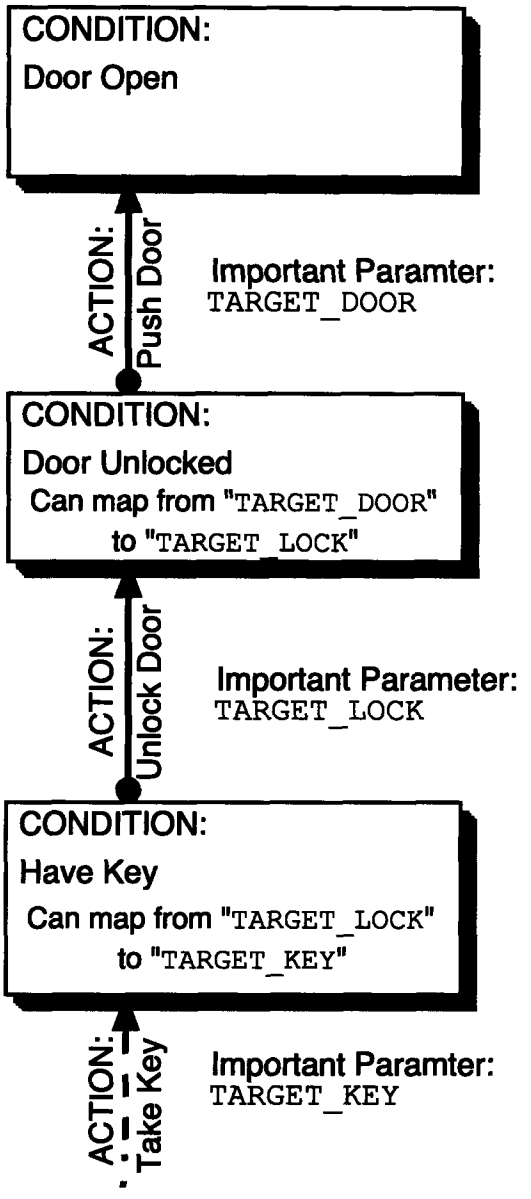

Figure 4.4: An example of why Conditions may need to modify existing parameters. Here, each Action Segment needs different parameters than are relevant to parent actions. Allowing Conditions to incrementally change current parameters to those relevant to their children allows for more modular design. Here an invoker of "Door Open" need not necessarily know anything about keys.

This Context Tree mechanism is well suited for this hierarchical system of Action Segments and Conditions, and allows for an efficient design for handling parameters. I have designed the Action Segments and Conditions so that they look in the Context Tree for any

arguments they need. This allows each Condition to add and modify parameters for its children, without altering data necessary for its parents to function.

For an example of a situation where this isolation of child actions is necessary, consider this compound action that contains two "move block" Action Segments. The first "move block" Action Segment might look under a predefined key for the block it should move, and the direction it should move it. It's precondition, "block clear", might look at the values under those same keys to make sure the target destination is free. If the area is not clear, the condition should employ its child Action Segment (also "move block") to make it clear. However, this "move block" Action Segment will be looking at the same keys as the original "move block" Action Segment, as they are the same, but the block it should be moving is different from the original target block.

Thus, each Condition, in addition to having the option of writing new data to the context, may elect to enter a new context before writing the data. This way, the execution contexts will reflect the same hierarchical structure as the Actions Segments and Conditions. This means that the original data will be available to all, unless it is overridden by more specific data relevant to some precondition. Also, preconditions overriding the original data will not harm the original data, as they will only do so in their own local sub-context, so as execution comes back to the top of the hierarchy the local sub-context is exited and all the original data will be intact (see Figure 4.5).

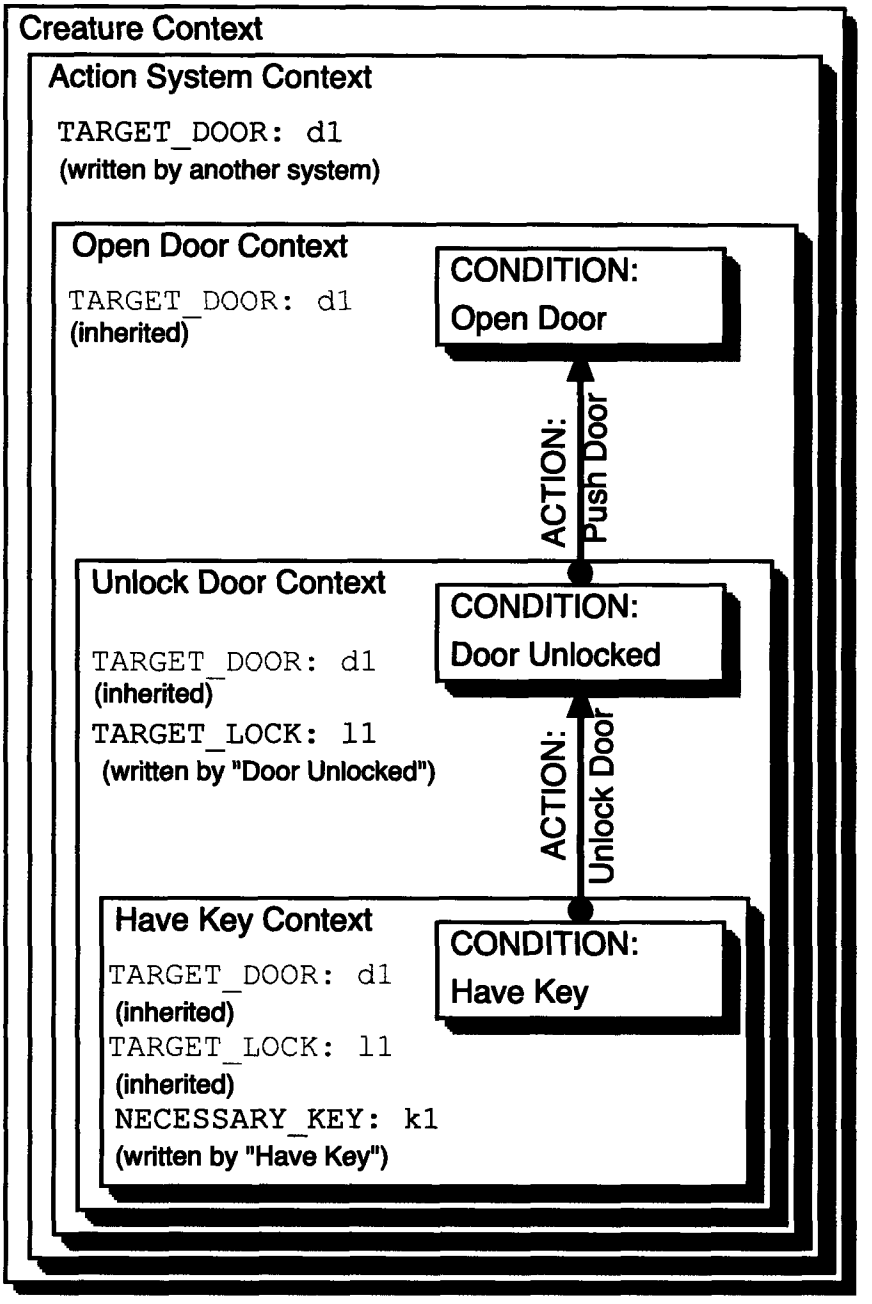

Figure 4.5: Action Segments receive their parameters through the context tree. Conditions convert parameters from the parent scope to those relative to their children.

Chapter 5

Using Action Segments in Reverse

Understanding human behavior in terms of actions instead of raw movements is thought to begin very early in children. By 1 to 1.5 years old, children are adept at imitating body movements and actions on objects (such as toys) in a variety of contexts. At 18 months, infants are able to read beyond perceived behavior to infer the underlying goals and intensions of the actor [15]. This is demonstrated by their ability to imitate the goal of an attempt that was enacted unsuccessfully. For instance, the adult may try to perform a manipulation on an object where her hand slips several times so the goal remained unachieved. The infant does not imitate the literal action, but rather performs the action correctly (or even performs novel means) to achieve the intended goal. This brings the infant to the threshold of understanding the behavior of others in terms of their underlying mental states.

Inspired by this, This chapter presents a method for a robot to attempt to understand human behavior in terms of action in a simulation theoretic manner. The previous chapter described a system for specifying the actions of the robot, and chapter 3 described a method for finding the robot's motion that corresponds most closely to the observed motion of the human. This chapter describes a way to use the motion information, in conjunction with other contextual clues, to find the Action Segment of the robot's that matches the action being performed by the human. Identifying the human's activity on the action level instead of the motion level gives the robot more information to work with. It can now infer the goal of the human's action based on its own Action Segment goals, as well as use the structure of its own Action Segments to determine what it might do to help a human who's struggling with a task.

5.1 Matching Observations to Action Segments

Every time the robot detects the human performing a known motion, it triggers a search through its Action Segments to try to find a match. An Action Segment is considered a match for the human's action if both the motion matches and the appropriate contextual information for the robot's Action Segment is present in the human's coordinate frame. Finding Action Segments that perform the motion in question is simple - each Action Segment that performs a motion registers its possible motions with a central repository when it is created, so finding all the Action Segments that could perform a motion is as simple as looking them up. Determining if the context is appropriate, however, is more difficult.

5.1.1 Determining if a Context is Valid

The "context" of the Action Segment refers to some features of the world state (and perhaps the relationship between those features and the doer of the action) at the time the Action Segment is performed. For example, the robot will only use its "push button" Action Segment when there is a button present and close by. In order to determine whether an Action Segment's context is currently relevant for the human, the context must be evaluated differently: it must be evaluated from the human's perspective.

Each Action Segment has its own context generation module (design decisions discussed in Section 5.4.1). The job of these modules is to discern the elements of the current context relevant to their Action Segment (from the perspective of the human), and then determine if these elements form a context that allows the action to be performed. For example, a button pushing Action Segment would have a module that must detect what object the human has touched, and determine if that object is a button. While these context generator modules could be arbitrarily complicated, most Actions Segments use a module from the small set of general modules, such as those that make some evaluation about the class of object the human is touching or pointing to. These are implemented using the robot's Belief System (see Section

4.2). For the "hand touching object" module, for example, first the position of the human's hand must be determined, and then the Beliefs close enough to the human hand can be selected and evaluated. The c5m Belief System has some nice features that are specifically useful for this type of search, including optional time histories of percept evaluations. This allows the robot to look back in time and determine the state of objects during different parts of the human's movements to better evaluate their relevance.

To determine which Action Segment a human is performing using context and motion, we must simply eliminate Action Segments whose contexts are not applicable to the current contextual state of the human. There is no need to use the context information other than to see whether it is valid. The next section, however, discusses a further use for this contextual information.

5.2 Being Helpful

5.2.1 Verifying Human's Goal

Once the robot has determined which Action Segment a human is attempting, it can infer that the human's goal is the same as its own goal would be in that situation. We know which goal this is - it is the goal Condition associated with the observed Action Segment. However, in order to evaluate that goal, the system needs the correct parameters to be present in the Context Tree (e.g. if the human is attempting to activate a button, which button is it?). In the normal operation of the robot, the initial parameters must be generated outside the Segmentable Action system. A higher level behavior chooses what button to press, and simply tells the Condition which is the target button by writing to a key in the Context Tree. Thus, we need the Context Tree to be populated with the appropriate data for the Action Segment and Condition in question, based on what parameters the human is using as they carry out the action. Here the mechanism from Section 5.1.1 comes to our aide. This mechanism is designed to discover the parameters the human is using to achieve the action. In its use in Section 5.1.1 it

is just queried as to whether or not it can find such parameters (i.e., whether or not the Action Segment could possibly match the human's current activity). Here, however, we wish it to find those parameters and populate a simulation context with them in order to examine the goal using the same objects the human is working with. For example, for a button pushing animation, the included mechanism will find the button that is close to the human's hand, and write that button to the target button key (but in the simulation context) normally used by the robot's "push button" Action Segment. The goal Condition of the Action Segment then, has enough information to evaluate the success of the human. The simulation context of Context Tree is now populated just as the Action System context would be in the forward operation of the robot's Action System, at least for data relevant to the "push button" segment. The goal Condition can then be queried just as it normally would, but if queried in this simulation context it will be applied to the data relevant to the human's action, so its success value will indicate the success or failure of the recently observed human's action.

5.2.2 Help On Failure of Human

If the robot classifies an action of the human as one of its own Action Segments and notices that the human failed to achieve the associated goal Condition, the robot can use this system to attempt to help out. Based on verifying the goal (Section 5.2.1), the robot has created a simulation context that contains parameters relevant to performing its own action in the same manner as the human (i.e., with the same parameters, on the same objects). The easiest thing for the robot to do, then, would be to perform the action. In some situations this could be helpful. If a human attempts a task and fails, the robot takes note and performs the task for the human.

Another way the robot can help is by performing a different action from the human, but one that helps the human accomplish their action. This is possible if the robot has some achievable preconditions to the action it observed. The robot can use the same mechanism it

would normally use to accomplish this Action Segment. It can first run through all the pre-conditions, making each one valid (Section 4.3). Instead of going on then doing the actual Action Segment, it can hold off and allow the human to complete the action now that the preconditions are met. When the robot is done, the human should have more success attempting the action, since the preconditions that may have been causing the failure have been eliminated. Only the first Action Segment (the one that was detected in the human's behavior) is required to perform the special step of populating the simulation context with contextual data calculated about the action from human's perspective. Once that Action Segment has populated the simulation context with that information, each of the preconditions and all recursive elements further on can run normally - drawing data as needed from this root simulation context which contains the appropriate data to allow the robot to perform its action in the same way, or on the same object as the human is attempting (just as if it had decided to do the action itself).

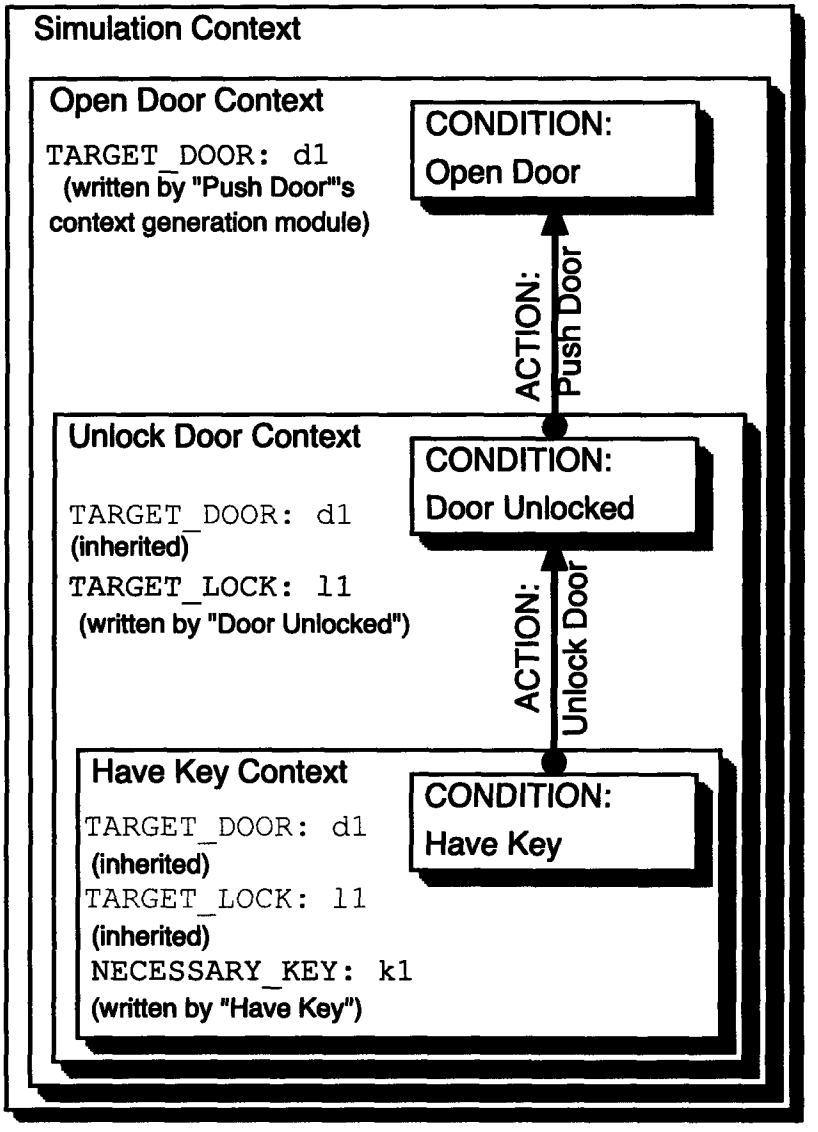

Figure 5.1: Context Tree For "Open Door" performed as a helping action. Compare with Figure 4.5, action execution in this scenario is intended to be similar to regular action execution, the only difference is where the top level parameters come from (Here "TARGET_DOOR")

The architecture described here can also provide support for other methods of helping the human. Since the robot knows which objects are salient to the necessary precondition action that the human neglected to perform, it can draw the human's attention to these objects even if it cannot reach them itself. It could do this by pointing at the important object or otherwise drawing the human's attention that way (especially relevant if the robot can detect that the

human cannot or has not seen these objects). It could possibly also use the fact that the human neglected these preconditions to make inferences about the human's knowledge of the task or the objects involved.

5.3 Advantages

Implementing goal inference and helpful behavior using this type of simulation theoretic technique has a number of advantages. Once this framework is established, it is easy to add new actions. Each new action not only adds functionality to the robot's own behavior, but can automatically be detected in humans. The new actions are also automatically available to help the human. Thus, from purely an ease of implementation standpoint, a simulation theory based system can be efficient and scalable. This will become particularly important when the robot has functionality to learn new combinations of Action Segments and Conditions - it will immediately be able to apply what it learned about itself towards recognizing that behavior in humans. Or taken the other way around, it will be able to use what it learned about human actions and goals to accomplish tasks for itself.

5.4 Design Choices and Limitations

5.4.1 Context Generator Modules

The object/parameter selection section of this implementation stands out for using a totally separate implementation for detecting as for producing (it is thus not very simulation theoretic, and requires more code than strictly necessary). It would be possible to do everything here without this special mechanism used only for detection. I initially avoided it, but included it as a compromise for robustness in the face of imperfect sensor data and different bodily proportions. It also has a certain biological plausibility. It seems most likely that, even if humans do make extensive use of simulation theory, they also rely on quick tricks to determine the target of an action. When a human observes another human push a button, they

probably do not go through an elaborate process of putting themselves in the other's shoes, recreating the motion, and determining which button they would push with that motion - they probably just look where the other human's hand is.

A different solution to this problem, instead of using a special module to detect what parameters the human is using or which objects the human is interacting with, is to exhaustively run the possible actions to determine the parameters. To do this, the robot must first put itself in the place of the human (alternatively, it may reposition the objects in the world so correspond to how they would appear from the human's perspective). Then, it must exhaustively try performing the Action Segment in question, trying to produce a motion that is most close to the observed human's motion. For a button pushing action segment, this could be as simple as trying the Action Segment on all the different buttons, and choosing the button that makes the push motion match the human's motion most closely. Action Segments that have continuous inputs are more difficult; some type of hill-climbing solution could be necessary. This solution will work for many types of Action Segments without the implementor needing to worry about the reverse usage of the segment for detection. However, it has problems.

Using this solution is appealing from an implementation standpoint, as it allows the implementor of new actions to skip the step of providing a context generation module. However, it is not practical in this domain. If the robot was able to determine the position of the human and detect the motion of the human's hand relative to that position accurately enough to reproduce the hand location in the world with enough precision (which is likely to be a difficult perceptual problem), there is still the problem of the difference in body sizes or shapes. Even with perfect sensor data, the difference in body could cause this method to fail. Consider the case of a human reaching for an object. If the robot is smaller than the human, and places itself in the human's shoes to try to determine the target of the action, its reach will be less when it recreates the motion, favoring selecting a closer object than the one the human actu-

ally selected. A good mapping between human position and robot position that takes this problem into account could potentially overcome this in certain cases, but the combination of needing extremely accurate sensor data with this possible inaccuracy makes this method unappealing.

Another solution to this problem, one that is more likely to be feasible, is to change the resolution of the Action Segments until they are small and specific enough that the preconditions of the Action Segments become relevant to determining the context of the human's action. For example, if "Push Button" was made of sub-segments, one might be "Push Down" with a precondition of "Hand Over Button". That type of condition is more relevant for determining the target object of the human. However, the Conditions must then implement some mechanism for running in reverse (answering "which button is the hand over"), which will likely look similar to the implementation of the context generator module for the same task. This solution also puts additional constraints on the way the actions must be written, which may be inconvenient for certain actions.

5.4.2 Limitations of Segmentable Actions

While this system is useful for recognizing certain types of actions and helping humans complete them, there are many types of tasks that they cannot yet handle. The most serious limitation is in the kind of action for which helpful behavior can be provided. These actions cannot yet be relative to the human. For example, if the human fails to push a button, the robot can help out. However, if the human fails to pick up an object (in order to hold it), there is no easy way for the robot to know how to help out; it does not help the human for the robot to perform the action that the human was attempting, because then the robot will be holding the object instead of the human (which may be even less helpful than doing nothing). A possible solution to this problem is discussed in the Section 7.1.2.

Another limitation is with the hierarchical structure of the Action Segments. Oftentimes a low level action can be used for multiple purposes; in that case, instead of a hierarchy a lattice might be more appropriate. These situations can be accommodated partially with only the type of hierarchies described here by creating multiple instances of the re-used Action Segments. Creating copies of the segments is not ideal, however. Also, if two Action Segments are completely identical, there will be no way for the simple matching mechanism described here to determine which is the one being performed (future work Section 7.1.5 describes a future matcher that takes into account multiple Action Segments that could work here)

Chapter 6

Demonstration

6.1 The Task

As a proof of concept of the architecture described here, I created a small action system for Leonardo using this system. Because Leonardo already has the perceptual ability to detect his buttons and the hand of a human in that area, I decided to design the task based on using these buttons. To make the interaction a bit more complicated, one of the buttons has been designated as lockable, such that it cannot change state if the lock is engaged. Another of the buttons serves as the control for that lock. The interaction with the human will be about pressing these buttons; Leo will observe the human pressing buttons, and step in to help whenever he detects that the human has made an error. This is a simple task domain, but it is complex enough to show both major modes for helpful behavior: performing an action that the human failed to perform, and performing an action that helps the human perform the action that they failed.

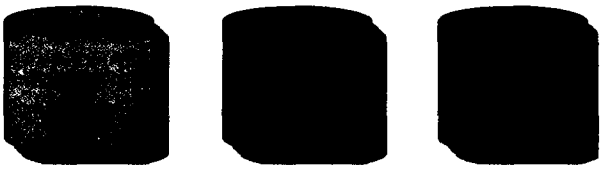

Figure 6.1: For this task, the blue button is locked by the red button. When the red button is on, the blue button is not pressable.

6.2 Implementation

Leonardo's button pressing action for this task was implemented as one compound action (see Figure 6.2). This compound action has two action segments, each of which is a simple segment which plays out a push motion with the correct parameters. The final condition is "Button On", which can be achieved by "Push Button". "Push Button" has the precondition

"Button Unlocked", which evaluates to true for unlocked or non-locking buttons. The condition "Button Unlocked" has another action "Unlock Button" to achieve it. This action also uses "Push Button" motion, but this time on the lock.

Each of the actions has a standard context generation module. These actions use the module that looks for a specific kind of object that is near to the hand. The module for "Push Button" looks for buttons that are close to the human's hand, while the module for "Unlock Button" looks for locks that are close to the human's hand.

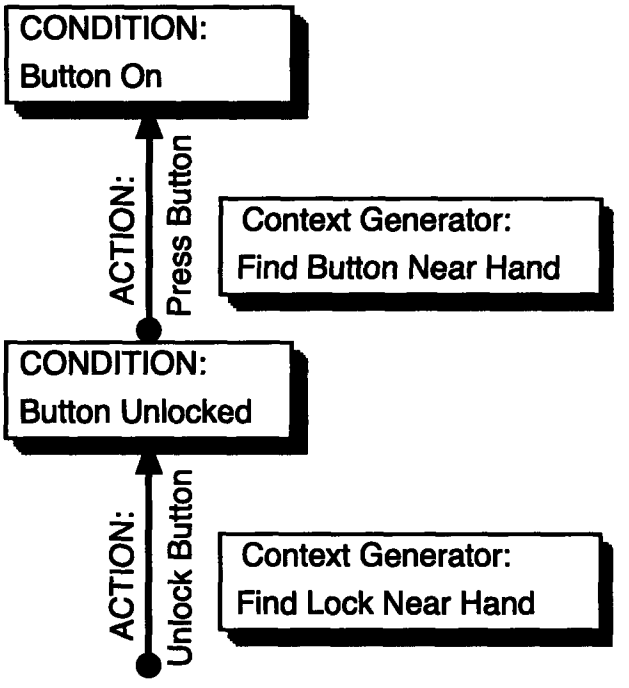

Figure 6.2: Implementation of "Press Button" compound action.

6.3 Results

The system performed as expected. After a person went through the calibration interaction to train the mapping from the gypsy suit to Leo's joint space, Leo was able to detect the motions known to him. Combining this data with perceptual information about the human's hand and the buttons, Leo was able to ignore uncontextual motions, acknowledge successful presses, and help with failed presses. Leo helped by repeating the action in the simple, no pre-

conditions case when the human failed to press a normal button. When the human failed to press a button because it was locked, Leo noticed that the human's action had an unsatisfied preconditions and unlocked the button for the human. The following figures depict the different types of interactions.

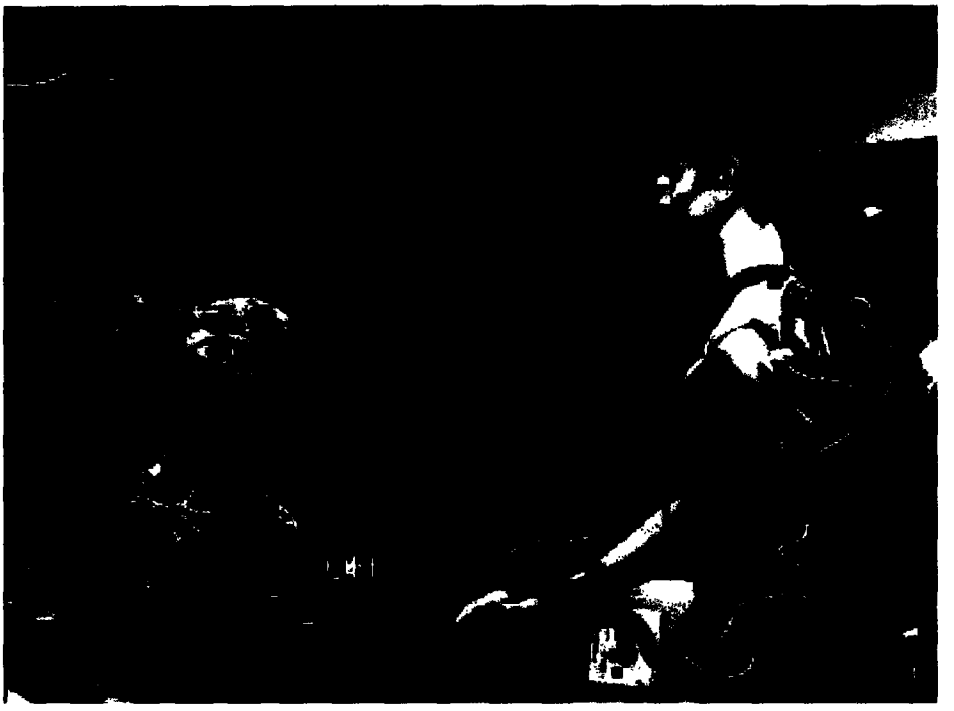

Figure 6.3: Motion in Wrong Context Trial - VIDEO: Human makes pressing motion in air

Figure 6.4: Motion in Wrong Context Trial - LEO's Internal State: Leo recognizes motion but has no action that matches the context

Figure 6.5: Successful Action Trial - VIDEO: Human successfully presses button

Figure 6.6: Successful Action Trial- Leo's Internal State: Leo recognizes human is performing a press action due to observed motion and context

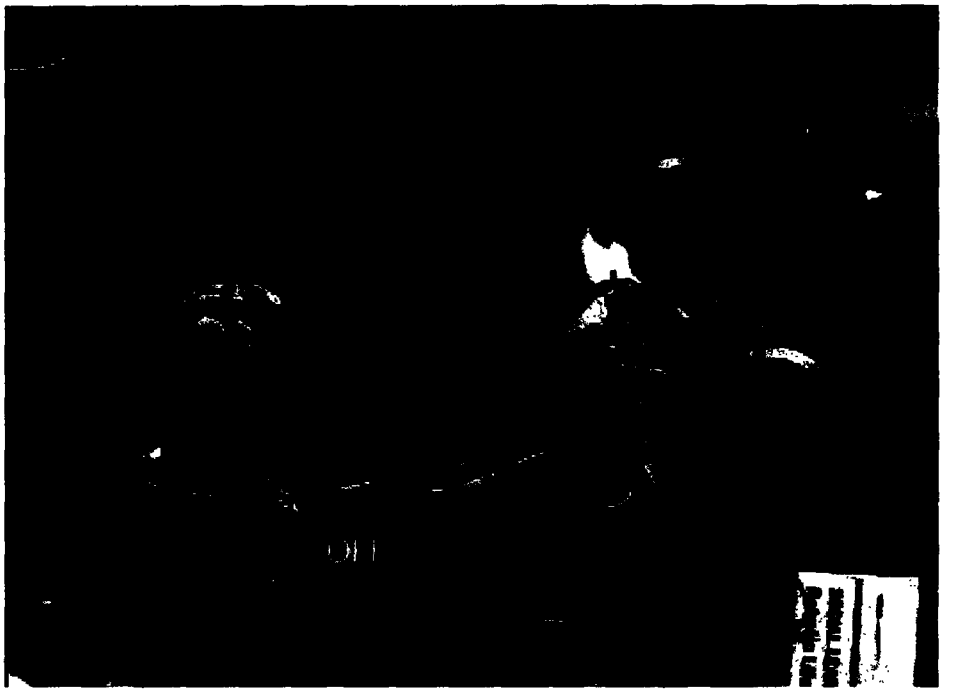

Figure 6.7: Failed Simple Action Trial - VIDEO: Human fails to press button - hand slips off

Figure 6.8: Failed Simple Action Trial - Leo's Internal State: Leo recognizes action and notices goal has not been accomplished. Since the action has no unfulfilled preconditions, Leo helps by redoing action on the same button.

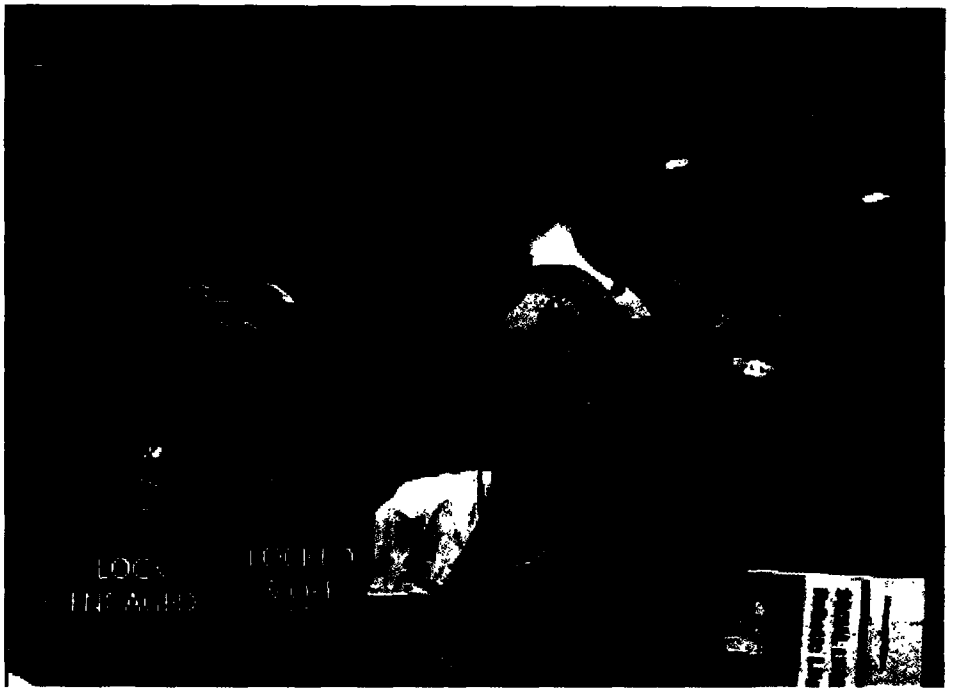

Figure 6.9: Failed Compound Action Trial - VIDEO: Human fails to press locked button because it is locked

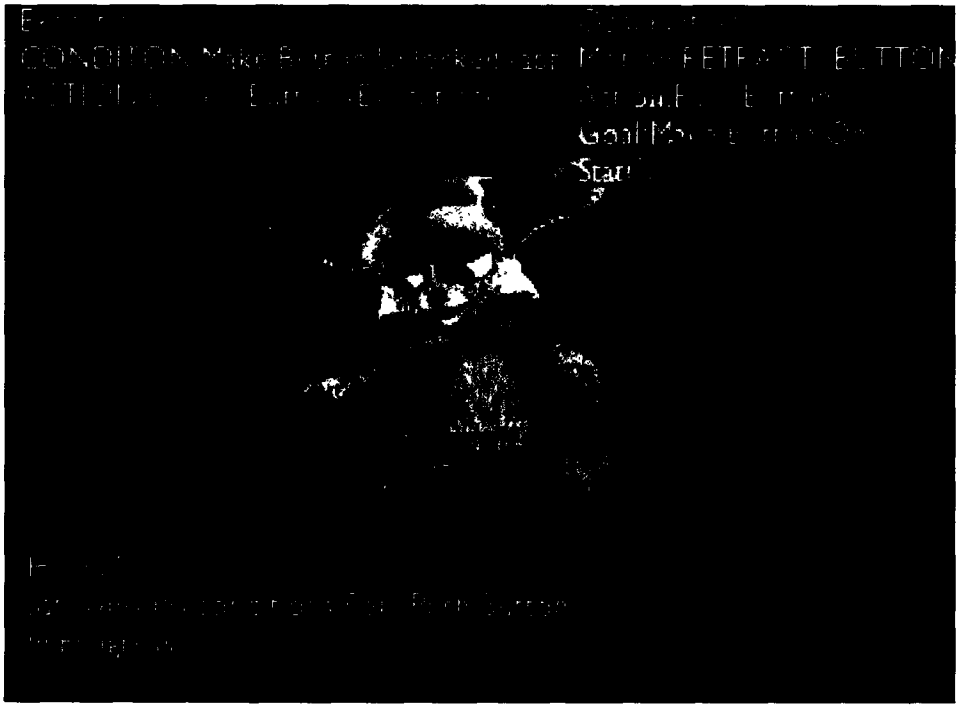

Figure 6.10: Failed Compound Action Trial - Leo's Internal State: Leo recognizes the action and notices the goal has not been accomplished. He helps out by achieving the action's precondition, unlocking the button.

Chapter 7

Conclusions

This architecture allows the robot to infer goals of simple goal directed actions observed in the human. It also allows the robot to help the human achieve the action in case the human fails. Both of these skills are possible using the creature's own actions system, so that little additional overhead is needed to utilize a new action in this manner when it is added to the system. This allows an implementor to add new actions easily while still leveraging their potential for recognition and inference, or in the future it may allow the creature to use newly learned actions in this manner. It also provides a measure of grounding to the observation abilities of the robot, since it classifies human behavior in terms of its own.

While I have only had a chance to implement a small proof of concept of this system, this architecture was easy to use to string together a simple compound action and everything needed for it to be used to detect the action in humans and help out. The way the system stands now, we should be able to explore designing new actions that Leo will be able to also detect and help with as they are performed by humans (within the limitations of the system, see Sections 5.4.2).

7.1 A More Complicated Interaction

The system as implemented allows for the interactions demonstrated here in Chapter 6 (and some that are more complicated), but new tools will be needed to scale the interaction to a lifelike level. The following sections describe some additions to support more realistic interactions.

7.1.1 Recursive Structure

One feature that the architecture described here is lacking is a formalized way nest compound actions within other compound actions. Having the option of creating recursive struc-

tures of these actions will allow complicated actions to be designed more clearly; it will also allow extending this architecture to higher level tasks, instead of only individual actions.

7.1.2 Variables in Action Segments

I think that variables for "self" and "other" that could be used in the design of the actions would enable new, useful types of actions. The action segments that I have created so far have avoided this issue by only dealing with global objects in the world, where the robot doing the action has the same effect as the human doing the action. However, these variables would enable the creation of relative actions (such as "I will hold the ball") and would provide hooks for the robot to be able to potentially help with those types of actions by examining the activity after switching the "self" and "other" roles.

7.1.3 Dialogue

The main type of Segmentable Action I implemented is one that performs a motion. Another kind whose addition would make the system more powerful is one that performs some kind of information exchange with the human, such as a query and response. For example, one dialogue action could contain the query "Which Button" and the ability to interpret a response. If these were implemented in such a way that they could be turned around just as the motion Action Segments can be (perhaps using some kind of variables for the roles, as above), it would provide a lot of flexibility; giving the robot the ability to ask questions to resolve confusions could automatically allow it to resolve similar confusion for the human.

7.1.4 Unhelpful Behavior

While this may not always be useful in a cooperative setting, unhelpful behavior is often as obvious to a human as helpful behavior. In order to play games or compete with humans, the robot must understand the opposite of helpful behavior. The simulation theoretic system would be helpful in achieving this. Ideally, without re-implementing any of the actions, the robot could try to foil the human instead of helping them. This would require the robot hav-

ing some notion of the opposites of the goal Conditions (it would need to know, for example, that "Button On" is the opposite of "Button Off"); however, the actions themselves and most of the system could remain unchanged.

7.1.5 Multiple Action Segment Recognition

In order to recognize Action Segments performed by the human, I implemented a method by which the human's motion and context are mapped onto one of the robot's Actions Segments. This mechanism could be significantly more powerful if it attempted to match recent observations against multiple Action Segments based on how they are arranged in the compound actions. Instead of choosing a definite winner it could score the probabilities of match to each of the Action Segments, and kept a short history of those scores. It could use this information to help resolve ambiguous observations by searching for compound actions that most closely match the recently observed series of Action Segments.

Bibliography

[1] Blumberg, Bruce, Marc Downie, Yuri Ivanov, Matt Berlin, Michael Patrick Johnson, & Bill Tomlinson. "Integrated Learning for Interactive Synthetic Characters." ACM Transactions on Graphics 21, no. 3: Proceedings of ACM SIGGRAPH 2002 (2002).

[2] Breazeal, C., Bushbaum, D., Gray, J., & Blumberg, B. (2004). " Learning from and about others: towards using imitation to bootstrap the social competence of robots ." (In review Artificial Life)

[3] Buchsbaum, D. (2004) Imitation and Social Learning for Animated Charcters. MIT Program in Media Arts and Sciences Master's thesis, Cambridge, MA.

[4] Burke, R., Isla, D., Downie, M., Ivanov, Y., & Blumberg, B. (2001). Creature Smarts: The Art and Architecture of a Virtual Brain. In Proceedings of the Computer Game Developers Conference. San Jose CA.

[5] Davies, M. & T. Stone. (1995). "Introduction." Folk Psychology: The Theory of Mind Debate . Ed. Martin Davies and Tony Stone. Cambridge: Blackwell.

[6] Demiris, J. & Hayes G.M, (2002). Imitation as a dual-route process featuring predictive and learning components: A biologically plausible computational model. In K. Dautenhahn & C.L. Nehaniv (Eds.), Imitation in Animals and Artifacts. (pp.231-361). Cambridge, MA: MIT Press.

[7] Downie, M. (2000). Behavior, animation, and music: the music and movement of synthetic characters. MIT Program in Media Arts and Sciences Master's thesis, Cambridge, MA.

[8] Friedrich, H., Holle J., & Dillmann, R. (1998). Interactive generation of flexible robot programs. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'98) , Leuven, Belgium, May 16-21.

[9] Gallese, V. and Goldman, A. (1998) Mirror neurons and the simulation theory of mind-reading. Trends in Cognitive Sciences. 2:12, 493-501

[10] Gleissner, B., Meltzoff, A. N., & Bekkering H. (2000). Children's coding of human action: Cognitive factors influencing imitation in 3-year-olds. Developmental Science , 3, 405-414.

[11] Gordon, R. (1986). Folk psychology as simulation. Mind and Language, 1, pp. 158-171.

[12] Gray, J. & Berlin, M. (2003) Motor Mapping and the Long Road to Goal Inference. (Class Paper, Cooperative Machines with Breazeal Fall '03)

[13] Kang, S.B., & Ikeuchi, K. Toward automatic robot instruction from perception - mapping human grasps to manipulator grasps . IEEE Transactions on Robotics and Automation, 13(1), February 1997.

[14] Lieberman, J. & Breazeal, C. (2004)Improvement on Action Parsing and Action Interpolation for Learning through Demonstration. (Submitted to International Journal of Humanoid Robotics)

[15] Meltzoff, A. N. 1995 Understanding the intentions of others: re-enactment of intended acts by 18-month-old children. Dev. Psychol. 31, 838-850.

[16] Meltzoff, A.N. (1996). The Human Infant as Imitative Generalist: A 20-Year progress report on infant imitation with implications for comparative psychology. In B. Galef & C. Heyes (Eds.), Social Learning in Animals: The Roots of Culture (pp.347-370). New York: Academic Press.

[17] Meltzoff, A. N., & Decety, J. (2003). What imitation tells us about social cognition: a rapprochement between developmental psychology and cognitive neuroscience. Philosophical Transactions of the Royal Society: Biological Sciences , 358, 491-500.

[18] Meltzoff, A.N. & Gopnik, A. (1993). The role of imitation in understanding persons and developing a theory of mind. In Baron-Cohen S., Tager-Flusberg, H., & Cohen, D.J., (Eds.), Understanding Other Minds, perspectives from autism (pp. 335 – 366). Oxford: Oxford University Press.

[19] Meltzoff, A.N. & Moore, M. K.(1994). Imitation, memory and the representation of persons. Infant Behavior and Development, 17, pp. 83-99.

[20] Meltzoff, A.N. & Moore, M.K. (1997) Explaining facial imitation: A theoretical model. Early Development and Parenting. 6, 179-192.

[21] Oztop, E.& Arbib, M. (2002). Schema design and implementation of the grasp-related mirror neuron system. Biological cybernetics, in press.

[22] Rao, R. & Meltzoff, A.N. (2003). Imitation learning in infants and robots: Towards probabilistic computational models. Proceedings of Artificial Intelligence and Simulation of Behavior (AISB): Cognition in Machines and Animals. UK.

[23] Rizzolatti, G., Fadiga, L., Gallese, V., Fogassi, L. (1996). Premotor cortex and the recognition of motor actions. Cognitive Brain Research, 3, pp. 131-141.

[24] Schaal, S. (1999) Is Imitation learning the route to humanoid robots? Trends in Cognitive Sciences. 3, 233-242

[25] Woodward, A. L., Sommerville, J. A., and Guajardo, J. J. (2001). How infants make sense of intentional action. In B. Malle, L. Moses, & D. Baldwin (Eds.). Intentions and Intentionality: Foundations of Social Cognition . (pp. 149-169). Cambridge, MA: MIT Press.

[26] Zukow-Goldring, P., Arbib, M.A., and Oztop, E., (2002). Language and the Mirror System: A Perception/Action Based Approach to Cognitive Development (in preparation)