January 2000

Building Parameterized Action Representations From Observation

Ramamani Bindiganavale
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-00-17.

Building Parameterized Action Representations From Observation

Abstract

Virtual worlds may be inhabited by intelligent agents who interact by performing various simple and complex actions. If the agents are human-like (embodied), their actions may be generated from motion capture or procedural animation. In this thesis, we introduce the CaPAR interactive system which combines both these approaches to generate agent-size neutral representations of actions within a framework called Parameterized Action Representation (PAR). Just as a person may learn a new complex physical task by observing another person doing it, our system observes a single trial of a human performing some complex task that involves interaction with self or other objects in the environment and automatically generates semantically rich information about the action. This information can be used to generate similar constrained motions for agents of different sizes.

Human movement is captured by electromagnetic sensors. By computing motion zerocrossings and geometric spatial proximities, we isolate significant events, abstract both spatial and visual constraints from an agent's action, and segment a given complex action into several simpler subactions. We analyze each independently and build individual PARs for them. Several PARs can be combined into one complex PAR representing the original activity. Within each motion segment, semantic and style information is extracted. The style information is used to generate the same constrained motion in other differently sized virtual agents by copying the end-effector velocity profile, by following a similar end-effector trajectory, or by scaling and mapping force interactions between the agent and an object. The semantic information is stored in a PAR. The extracted style and constraint information is stored in the corresponding agent and object models.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-00-17.

BUILDING PARAMETERIZED ACTION REPRESENTATIONS FROM OBSERVATION

Ramamani N. Bindiganavale

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

2000

Supervisor of Dissertation

Graduate Group Chairperson

To Rungun, Rochitha and my parents

Acknowledgments

During my graduate study at UPenn, I have met numerous people who have influenced and shaped my life and research in many ways. The first and foremost among them is my advisor Dr. Norman Badler to whom I am greatly indebted. It is his vision and ideas that have greatly inspired the work done in this thesis and brought it to its completion. Over the years, working both as a student and as an employee under him, I have learned a lot about research and management from him. Norm has been an ideal advisor and employer. He has always been there to guide and motivate me and also trusted me with independence and responsibility. I greatly admire his intelligence, managerial abilities, and selfless devotion to his students, and am grateful for his years of advice, guidance, and support.

I would like to express my sincere thanks to my committee chairperson Dr. Martha Palmer for taking the time to discuss the thesis in detail, and for helping me look at the results of my work from a natural language perspective. This has helped in exposing the work to several other fields too. I would also like to thank the other committee members, Dr. C.J. Taylor, Dr. Dimitris Metaxas and Dr. Jeff Siskind for their useful suggestions and comments which helped strengthen the work and its presentation.

I want to express my sincere thanks to Harold Sun. With a lot of patience and with a lot of smiles, he endured my numerous trials of getting the motion capture data. He has also helped and taught me a lot of C++ programming. Working with Harold has been a lot of fun. I would also like to express my thanks to Jan Allbeck who was another source of my motion capture data. But, most importantly, she has been a great friend who is always there to help, whether to discuss or to criticize my work, to proof-read my thesis, or to just listen to my ideas and ramblings. Others that deserve thanks include Deepak

Tolani for providing the inverse kinematics code; Liwei Zhao and Jianping Shi for working with me in developing the PAR system; Charles Erignac, Fred Azar and Suejung Huh for evaluating the results of my work; Karen Carter for being a great friend and colleague who has patiently listened to me talk on and on about my little girl; Christian Vogler for building the drivers for the motion capture system; Mark Palatucci, Ken Noble, Angela Co and Koji Ashida for helping me with video editing during the various phases of my research; Hans Oskar-Porr for teaching me all about animation; Bond-Jay Ting, Xinmin Zhao and Pei-Hwa Ho for their discussions and help during my initial years here. I would also like to thank all the members of the VET team in helping build a great product which in a way helped complete my thesis and for staying up all those nights while we debugged and met the various deadlines. I want to acknowledge EAI for providing the Jack software in which this thesis has been implemented. I would like to thank the developers at EAI, in particular John Granieri, for the prompt technical support.

I have been very lucky to have such a wonderful family. My parents have always been my role models who have taught me to face all challenges in life with a smile and to achieve my goals and dreams with a lot of hard work and determination. I thank them for their unflagging support, encouragement and guidance. I want to thank my sister, my wonderful wonderful in-laws and the rest of my family for their love and support. I want to say a special thanks to my best friend, Asha, who has been there for me since we were 5 years old. Even from a long distance, she is there to have fun with, to share my joys and to hold my hand whenever I need it and encourage me in everything I do.

I am so grateful to my daughter, Rochitha, who was born during my graduate studies and has been the light of my life ever since. With her sweet smile, her words and her laughter, she has been able to instantly erase all the tiredness and frustrations I have faced during the work and taught me to reconsider the things that are more important in life. There are no words that can describe my love and heartfelt thanks and gratitude to my dearest husband, Rungun, who has stood by me through every step of my thesis. He is a wonderful father to our daughter and is also my best friend. His love, encouragement and constant moral support gave me the strength to overcome all the difficulties and frustrations in the course of my research. I am so thankful for his willingness to always listen to my ideas, to help me realize the same and to help me in every aspect of my thesis and my life. This thesis would never have been possible without all his help. I dedicate this thesis to him.

This research is partially supported by U.S. Air Force F41624-97-D-5002, Office of Naval Research K-5- 55043/3916-1552793, NSF SBR-8900230 and IIS-9900297, and NASA NRA NAG 5-3990.

Abstract

BUILDING PARAMETERIZED ACTION REPRESENTATIONS FROM OBSERVATION

Ramamani N. Bindiganavale

Norman I. Badler

Virtual worlds may be inhabited by intelligent agents who interact by performing various simple and complex actions. If the agents are human-like (embodied), their actions may be generated from motion capture or procedural animation. In this thesis, we introduce the CaPAR interactive system which combines both these approaches to generate agent-size neutral representations of actions within a framework called Parameterized Action Representation (PAR). Just as a person may learn a new complex physical task by observing another person doing it, our system observes a single trial of a human performing some complex task that involves interaction with self or other objects in the environment and automatically generates semantically rich information about the action. This information can be used to generate similar constrained motions for agents of different sizes.

Human movement is captured by electromagnetic sensors. By computing motion zero-crossings and geometric spatial proximities, we isolate significant events, abstract both spatial and visual constraints from an agent's action, and segment a given complex action into several simpler subactions. We analyze each independently and build individual PARs for them. Several PARs can be combined into one complex PAR representing the original activity. Within each motion segment, semantic and style information is extracted. The style information is used to generate the same constrained motion in other differently sized virtual agents by copying the end-effector velocity profile, by following a similar end-effector trajectory, or by scaling and mapping force interactions between the agent and an object. The semantic information is stored in a PAR. The extracted style and constraint information is stored in the corresponding agent and object models.

Contents

List of Figures

Chapter 1

Introduction

How do you teach a new complex physical task to a group of people? As a natural solution, you first demonstrate the task by breaking it into simple subtasks. The group of people initially observe you, assimilate information about the task, and then attempt to perform it. While the people's actions may be similar, they will not be exactly the same. The dissimilarities are mainly due to the differences in sizes of the people, as well as individual performance or stylistic variations. Also, when the people are asked to repeat the same task in a different situation, they automatically adapt to the new environment and perform the task with subtle variations. The main objective of this thesis is to build a system that can observe a given action, automatically abstract motion information and build a parameterized action representation (PAR) from it. From a single PAR, we then successfully generate the task motions for virtual human models of different sizes in different situations.

The parameterized action representation (PAR) [8] gives a high level description of an action. A PAR is parameterized because an action depends on its participants (agents who execute the action and the objects involved in the action), and other attributes) for the details of how it is performed. A PAR also includes applicability conditions and preparatory specifications that have to be satisfied before the action is actually executed. The action is finished when the termination conditions are satisfied. Uninstantiated PARs (UPARs) are stored hierarchically in a database, called the Actionary. During execution, an UPAR is instantiated into an IPAR (Instantiated PAR) with specific information on the agent, physical object(s), manner, termination conditions, etc. Our system generates a UPAR with the necessary PAR attributes, the default preparatory specifications, and the termination conditions of the observed action.

We are mainly interested in contact based complex actions involving interactions with objects (*e.g.*, drinking from a cup, digging with a shovel, etc.). The non-contact based actions generally fall under gesture movements and can be handled by other systems like EMOTE [18]. We motion capture the actions of a person, executing various complex actions involving interaction with objects (including self) in the environment. These actions are mapped in real-time to a same-sized virtual human model (called the *primary agent*). The motion captured data is automatically segmented, parameterized and analyzed for specific features used in generating a new UPAR or in recognizing an existing one. The UPARs are now applied to imitators, referred to as *secondary agents*, who try to execute the same action by interacting with the objects in a similar fashion. The thesis here is that *relationships between the world, the body, and the end-effectors (hands, eyes) of the primary agent have been overlooked and are of considerable importance in reconstructing correctly scaled motions.* Often the objects being held are simply wielded for effect, such as holding a shield or slashing with a sword. Keeping feet in contact with the ground plane is one frequently encountered problem, but usually only vertical displacements are moderated: the actual horizontal step position may be input to inverse kinematics procedures to keep the body from floating or sinking. The issue of changing the step locations is related to the motion mimicry problem, but we do not address it here. In this thesis, we show that the PARs generated automatically from observing motion captured data can be used to generate the correct actions while successfully maintaining spatial constraints for hands and eyes, such as grasping a cup at the correct place and bringing it to the mouth for a drink. The style, for the new motions, can be borrowed from the motion captured data or can be newly generated using various manner parameters.

In this thesis, we first introduce a technique to automatically recognize spatial and visual alignment constraints from captured motions. Maintaining these constraints is the basis of motion mapping from the primary agent to secondary agents. We then extend this technique to do automatic motion segmentation and parameterization of a given action. The problem of recognizing motion events directly from (synthetic) image sequences was first studied by Badler [7]. We update these notions to abstract information about significant events and spatial constraints from 3D motion captured data. In general, we are interested in building complex actions out of simple components (motion primitives), as we feel this approach accelerates learning of new tasks. Hence, we break up a given complex action into several simpler subactions, analyze them independently, and build individual PARs for each of them. We finally combine all the primitive PARs into one complex PAR representing the original activity. We test our PAR generation system on the following examples - drink from a mug, touch an object, slide an object and pick up the object with both hands.

We emphasize here that unlike Atkeson *et al.* [2, 55], we do not use any form of "machine learning" techniques for generating motions from observation. Instead, we show that we can abstract all required information from one sample. We prove this by testing the motion abstraction and PAR generation methods on three different sets of motion captured data.

The task of generating a parameterized action representation from observation can be divided into several subtasks:

Motion Generation for Primary Agent: Generate motions for the primary agent from motion captured data by using real-time optimization techniques [61, 67] to solve for the kinematic constraints imposed by the data itself. This process is described in Section 3.1.

Automatic Motion Segmentation Automatically segment the primary agent's actions using the concepts of *motion zero-crossing* and *co-occuring geometric spatial proximities of end-effectors with interacting objects* to recognize the spatial constraints. The occurence of a spatial constraint signals the end of a motion segment. The segmentation is done separately for each kinematic chain of the virtual human's body. This process is described in Section 4.3.

Motion Understanding and Feature Extraction: We analyze each individual motion segment and extract relevant features, and PAR parameters. We also abstract the line of attention of the primary agent during significant events. We then impose this as an additional spatial constraint to be solved during motion generation for the secondary agents. This alignment constraint forces the secondary agent to *look at* the same objects. This provides a very natural affect as, in general, people tend to look at an object while interacting with it [21]. All the extracted features are discussed in Section 4.4.

Action Recognition: We compare the extracted features with those of pre-existing action descriptions in the feature table. If no match is found, it is added to the feature table and a new UPAR is also created. The process of feature-based action recognition is described in Section 4.5 and the process of building primitive PARs is explained in Section 4.6.1.

Building Complex Action: We combine the PARs generated for all the motion primitives and create one complex UPAR corresponding to the performed action. This is explained in Section 4.6.2.

Motion Generation for Secondary Agent: In the final stage, we generate motions from newly created PARs for any secondary virtual agent in a given environment. The generated motions are shown to have different styles borrowed from the primary agent. This is explained in Section 4.7.

The thesis is organized as follows. In Chapter 2, we review the different approaches and their purposes for studying human performance data with an intention to imitate it either by a robot or by a virtual agent. In Chapter 3, we describe the process of motion abstraction and mapping the motions directly from the primary agent to a secondary agent. In Chapter 4, we describe our methodology for generating a parameterized representation for the given action. In Chapter 6, we discuss the results and some future work.

Chapter 2

Background

A common way of teaching someone to perform a movement skill is to demonstrate or model the skill. Romack, in [53] describes imitation or observational learning as the process by which a performer acquires a novel behavior by observing and attempting to produce actions performed by another.

In this thesis, we are attempting to teach a virtual agent to perform a new task. To do this, we develop a system that observes a person's actions and parameterizes it in such a way that a virtual agent of a different anthropometric size can imitate the performer's actions. In this chapter, we discuss different approaches and their purposes for studying human performance data with an intention to imitate it either by a robot or a virtual agent.

There are many sources of performance data: video cameras, LCD cameras, CCD cameras, motion capture systems [41] - magnetic, optical and video, etc. The magnetic motion capture system generates data that has six degrees of freedom (three for position and three for orientation). The data from the optical motion capture system has three degrees of freedom for position. The data from the rest of the sources has to be analyzed using vision-based techniques. Human performance data in any form is a rich source of information, and is immensely valuable for a variety of applications :

- generating animations in the virtual world [1, 13, 17, 46, 56, 57, 62, 63, 64],
- motion understanding [35, 40],
- action recognition [3, 14, 23, 24, 26, 59, 66],

- building interactive virtual worlds [10, 15, 26],
- robot learning from demonstration [2, 55]

2.1 Generating Animations in the Virtual World

Motion capture is a very popular and efficient technique for generating complex and natural motions of virtual humans or any other virtual character. In most applications, sensors or optical markers are placed on an actor's body, and the motion-capture system monitors and records the actor's motions. The resulting data can then be used to animate many types of virtual characters.

There are different types of motion capture systems (surveyed in [42]) - magnetic, optical, and video. We use the magnetic system, *Motion Star*, from Ascension Technology. But, the methodologies and techniques described in this thesis are applicable to data from any source.

The motion captured data can be used in several ways. One of the main purposes is to replicate motions of the actor on different sized avatars in real-time [13, 9, 46, 56, 57, 63]. For other applications, the motion-captured data can be further edited using various techniques. Some of the techniques [1, 17, 62, 64] use signal processing methods to edit and modify the actions for the same virtual agent. The techniques described in [22, 29] use optimization methods to modify the original motions in the presence of space-time constraints. As explained in [22], space-time refers to "the set of all DOF (joint angles and figure position) over the entire animation sequence." In [54], a new set of transition motions are created between two basis motions using space-time constraints. All these techniques treat the problems of mapping motions from one virtual agent to other agents, of modifying the nature or style of the motions, and of modifying the motions while maintaining space-time or spatial constraints as separate problems and solving them individually.

2.1.1 Motion Retargetting

The focus of our thesis is on actions involving interactions with self or other objects in the environment. Given the motion captured data of a person interacting with the environment, our goal is to build a system capable of understanding, abstracting, and parameterizing the motion data such that the data can be adapted to agents of different sizes while maintaining the constraints. This problem is commonly referred to as motion-retargetting. In [31], constraint based motions are adapted to other agents, but interactions between objects and self are not considered. In [30], optimization techniques are used to retarget the motions to other agents during object interaction. But a very simple human model is used and the problem of visual constraints is not considered. In [19], the problem of real-time motion retargetting is addressed.

In [33], Hodgins *et al.*, describe a technique that adapts an example behavior to the physical characteristics of a new character. This technique works by scaling control system parameters based on a dynamic analysis of the two characters. The original motions of the first agent are computed using dynamic simulation, consisting of a dynamic model containing equations of motion for the rigid body model, constraint equations for the interaction with the ground and parameterized control algorithms for running or bicycling. During each simulation step, the control algorithm computes desired positions and velocities for each joint based on the state of the system and the requirements of the task as specified by the user. Proportional-derivative servos compute joint torques based on the desired and actual value of the joint. The equations of motion of the system are integrated forward in time, taking into account the internal joint torques and the external forces and torques from interactions with the ground plane or other objects. These control system parameters are then scaled (for both geometric and mass data) to achieve motion for a new character that has similar dynamic properties to that of the original.

2.2 Motion Understanding

Motion understanding is the key to action recognition. Bobick [14] distinguishes between a motion (movement) and an action as follows: a movement is the most atomic primitive, requiring no contextual or sequential knowledge to be recognized; an action is a larger scale event which typically includes interaction with the environment and causal relationships. Also, understanding an action implies producing a semantically rich description of the various action primitives and the relations between them. In this section, we describe various techniques that use qualitative methods to understand and describe an action.

2.2.1 Computational Scene Dynamics

In [40], Mann *et al.* reason about qualitative scene dynamics to understand observations of interacting objects. They derive symbolic force-dynamic descriptions directly from camera input. An action performed by a human is observed through a camera, focusing on interaction between a hand and the objects. All other movements are ignored. The motions are interpreted in terms of explicit Newtonian physics-based representations. In the very first frame, the positions of all objects are assumed to be known. From this, the motions are easily tracked over time. Based on the contact relationships and the types of forces between the objects, various interpretations are made at each frame. Each interpretation is checked to see if it is dynamically feasible and the most preferred one is selected. These interpretations are used to identify the active and passive objects in an action.

In [39], the above technique was modified to make interpretations over a period of time thus removing the per-frame ambiguity. The interpretations are of the form CONTACT(object), ATTACH(object), FLYER(object), GRASPER(object), etc. These interpretations can be strung together to give a description of an action. But, no further information is obtained which can help in reproducing similar actions.

2.2.2 Task understanding of polyhedral objects

In [35], Ikeuchi *et al.* observe a human performing an assembly task, understand the task and generate a robot program to achieve the same task. A video camera is used for observing the task. In this paper, the actions of a person manipulating a polyhedral object are abstracted as a series of robot commands. The actions are initially captured on camera. The start and end of actions are determined by the appearance and disappearance of the person in the series of images.

An object recognition module first extracts the features of all objects (not the human) in the environment and then sends them to a geometric modeler which builds the corresponding geometric object models. These models are used to determine all the face contact relationships. It is assumed that at any given time, there is only one manipulated object and the rest are stationary objects. Only the objects in the environment are of interest and not the human. So, the main focus is on what manipulative actions were done on the objects and not on how they were done by the person.

The different configurations (assembly relations) for polyhedral objects in contact are first pre-defined based on the number of faces (sides) in contact. A contacting face pair is formed by a face from the manipulated object and a face from the environmental object, which have the same face equations and whose surface normals are opposite in direction to each other. These configurations also determine the possible motion directions - in-contact direction (object moves but remains in contact with the environment objects) and detaching direction (object moves breaking the face contact). Abstract task models are defined for all the possible transitions between these assembly relations. Each task model consists of an assembly relation transition, a motion macro (like move, insert-into actions) and the necessary parameters required to expand the motion macro into a sequence of manipulator commands.

In the instantiated environment, a geometric reasoner is used to recognize the start and end assembly relations from the pre-action and post-action world models respectively and to recognize the correct abstract task models. There may be several paths to transition from the start to end assembly relation. The correct path is determined by first using dis-assembly relations from the goal configuration (i.e. going in the reverse direction). Five motion parameters (starting configuration, approaching configuration, goal configuration, grasping configuration and approach direction) are determined for each instantiated action. The approach direction is determined by the contact normal of the object in the post-world model. The approaching configuration is determined by translating the goal configuration along the contact normal. Each abstract object model has several candidate configurations for grasping. The correct one is calculated based on the current body configurations and collisions with the environment objects.

Finally, the instantiated tasks are converted to robot text commands. The robot is assumed to be capable of doing all the actions with no constraints.

2.2.3 Qualitative Recognition

Kuniyoshi *et al.* [37] developed a qualitative visual recognition system for block assembly tasks. Here, a multi-processor vision hardware system is used for detecting various visual features in parallel. Qualitative changes in the movement of the tracked object together with a directed visual search trigger attention switching and temporal segmentation. Movements and their effected events are used in classifying the actions.

2.3 Action Recognition

The problem of recognizing actions has been attempted both in the robotics and graphics domains. For effective action recognition, observing a performer's bodily motion is necessary but not sufficient. A recognizer must also look for causally linked effects in the performer's surroundings and relate movements and effects. We need information not only of the movement of the human but also of the effect, which must be causally linked. For example, *reach, touch, pickup*, etc, may have some similar features but they cannot be classified to be the same using only body movement information.

There are two distinct approaches to action recognition - the knowledge based or context sensitive approach and the machine learning approach. In this section, we discuss various techniques using the two approaches. In all these methods, newly generated actions are matched against previously stored definitions of actions.

2.3.1 Knowledge Based Approach

In [3], computer vision techniques are used for security applications to recognize a few very specific actions (like *picking up a phone, stand/sit*, and *use a computer terminal*). For each action, specific regions of the image are tracked and the results completely depend on the prior knowledge of the environment. The output of this system is both a textual and a key frame description of the recognized actions.

In [26], Emering *et al.* address the problem of recognizing full body human actions in real-time. A set of actions is maintained in a database from which candidate sets are selected based on the closest match to the given new action. Each action in the database is defined as a combination of action primitives - positions/velocities of the center of mass, end effectors and final postures of the virtual skeleton. The action description process involves 2 hierarchical levels - gesture and posture. At the gesture level, the action is described in terms of (CoM,velocity direction) or (endeffector, velocity direction). Example of velocity directions are front, back, left, right, etc. which are all defined relative to the body. At the posture level, the action is defined in terms of the final posture (joint angles).

In the action recognition process, the candidate data set is initialized with the database. (i.e. all actions inside the database are initially considered as candidate actions). Then, for each new body posture sample, the CoM velocity is first compared with the CoM velocities of the actions in the database. If there is no match, the process is aborted. Otherwise, only the few actions in the candidate set with which there is a match are retained. Next, the EndEffector (EE) velocities are compared. The filtering process continues with the CoM, EEs and the joint values. But now, the comparison is with the final posture's joint angles and CoM and EE positions. Only those actions in the candidate set which differ by a very small amount (prespecified) in value from the given new action are retained. The joint angles are considered in the final stage of the recognition process.

The actions stored in the database are prototype actions - derived from a single person's actions. Most of the actions considered are free actions and do not involve any interaction with objects. If there is any object, the presence of the object is determined by the constant distance between the hands. All the measurements are normalized to the agent's height. Hence, actions of different sized virtual humans can be recognized. But, if the same task is done with different styles, then it cannot be recognized. This is because a close match to postures, joint angles and velocities is sought at each frame. The recognized actions of the avatar can be used to activate motion generators to produce a similar or reactive action in another agent.

2.3.2 Machine Learning Approach

In [59], Siskind *et al.*, use a maximum likelihood approach for training models to recognize simple spatial motion events. The event recognition task is partitioned into two subtasks - tracking and classification. In the lower level task of tracking, colored and moving objects are tracked separately. Using the techniques of proximity clustering and region-growing, each of the objects is fitted with a parameterized ellipse that abstractly characterizes the position, orientation, shape and size of the object. A large feature vector, extracted from each ellipse, is used for the upper level task of event recognition. Supervised learning techniques are used to train a Hidden Markov Model from a set of examples for each event class. Then given any new sample, the HMM is used to identify the class (or event) that has the closest correspondence with the sample. In this method, the start and end of events is specified manually. Also, only spatial motion features are recognized and not force-dynamic ones.

In [23], Davis *et al.* use temporal templates for action recognition. The two components of the temporal templates are MEI (motion-energy image) indicating the presence of motion and MHI (motion-history image) indicating the recency of motion. Action recognition is done by comparing the templates against a table of known actions using Mahalanobis distance. In [15], Bobick *et al.* use this action recognition process along with knowledge-based recognition to build a perceptually based interactive narrative virtual space for children to play in.

2.3.3 Building Virtual Worlds

In [10], Balcisoy *et al.* describe an augmented virtual reality system in which acts of a real person and a virtual human are portrayed in the virtual world. The virtual human's actions are completely triggered by user input and not by action recognition.

2.3.4 Robot Learning from Demonstration

In [2], Atkeson *et al.* describe learning a pendulum swing up task for a Sarcos robot arm based on a human demonstration of the same task. The human motion is measured using a stereo vision system. The robot tries to follow the human hand trajectory, and learns a task model and a reward function (intention model) by watching its own performance over a few trials, using the same camera used to watch the human performance. A planner is then used to find a swing up trajectory that works for the robot based on the model learned. A parametric model is built using a knowledge-based approach and a non-parametric model is built using locally weighted learning. In [55], Schaal extended this technique to use reinforcement learning. In this case, the robot was able to learn pole-balancing from a demonstration in a single trial with great reliability.

2.4 Summary

In this chapter, we reviewed the different approaches to and their purposes for studying human performance data with an intention to imitate it either by a robot or by a virtual agent. Some of the techniques only addressed the problem of deriving motions from motion capture or motion retargetting. Some of the other techniques addressed the problems of understanding motions for generating descriptive text, or of recognizing previously generated motion types (action recognition) or teaching a robot using various learning techniques to cause similar object motion. Also, the focus of all these techniques was on replicating either the object motion or the agent motion but not both. In this thesis, we address the problems of understanding motions involving interactions with other objects or self, to abstract style and semantic features from them and to generate parameterized conceptual representations of the actions, to efficiently recognize previously generated actions, to adapt extracted motion information from one agent to another, and to use a single trial of reference motion to teach virtual models of different sizes to perform the same action while adapting to a new environment. We also replicate the motions of both the agent and the object.

Chapter 3

Motion Abstraction and Mapping

The first task in generating a parameterized action representation from observation is to abstract all necessary semantic and motion parameters from the action. In this chapter, we describe the initial process of abstracting motion from a primary agent's actions and mapping it directly to a secondary agent without building a UPAR. This process is the basis of our PAR generation system. In the next stage of generating a PAR (Chapter 4), we deal with each motion segment separately and build a PAR which can be reused later.

This chapter is organized as follows. Section 3.1 describes the human body model and the technique used to derive the motions for the primary agent from the motion capture data. Section 3.2 describes the technique to recognize the spatial constraints and Section 3.3 explains the method to compute the locations of the constraints. Section 3.4 describes the techniques for mapping the motions to other agents and Section 3.5 describes a simple technique to recognize the visual attention of the primary agent. Section 3.6 presents the results of motion abstraction and mapping to another differently sized agent.

3.1 Deriving Motions from Performance Data

The human performance data used in this thesis is obtained by the motion capture process. But, our technique can be applied to data from any source - motion capture, key-frame or procedural. We use the MotionStar system, from Ascension Technology. It consists of one Extended Range Controller (ERC), one Extended Range Transmitter, and 12 Bird units, each controlling a single receiver (referred to as a sensor in the remainder of this paper). Although this system is cost-effective, productive and efficient in generating data, it has one main drawback. It is an electro-magnetic tracker and is hence susceptible to interference from neighboring external sources of fields [50]. The interference of the power supply frequency and its harmonics with the sampling frequency introduces noise. To minimize this, we use a sampling frequency of 103.3 Hz.

As a preliminary, off-line step in deriving motions from motion capture data for the primary agent, an avatar is built to the size of the subject and is calibrated by placing one of the sensors of the MotionStar system on the lower back of the subject roughly corresponding to the *L5* segment of the spine (sacro-iliac). In the human model, a corresponding site[1] (FOBpelvic) is created in the *L5* segment of the spine. We have implemented this technique completely within the EAI *Jack*®[5] software. The human model we use is highly articulated and has 68 joints and 135 degrees of freedom. The transformations between the MotionStar reference frame and the Jack reference frame are calculated by positioning the pelvic sensor at the FOBpelvic site. All the sensors are then mapped correctly onto the human model in the Jack environment. Next, using the data from all the sensors for the first frame, the human model is postured correctly to match the initial posture of the subject. Finally, sites are automatically created within the human model at the locations where the sensors lie on the body. We refer to these sites as EE sites.

To generate the motions, kinematic constraints are established between the EE sites and the sensors. Subject to these kinematic constraints and reach-space constraints, our IK routines [61] cause the EE sites to accurately follow the goal sensors. This is shown in Fig 3.1. So, as the sensors move, the human model moves with them along newly computed trajectories. This process easily creates motions in real-time while interacting with an object for a similarly-sized avatar. To recreate the same motions for a different-sized agent while maintaining the spatial constraints, we first need to post-process the data to recognize the spatial constraints and map the newly derived data to the secondary agents.

[1]Sites are oriented co-ordinate triples.

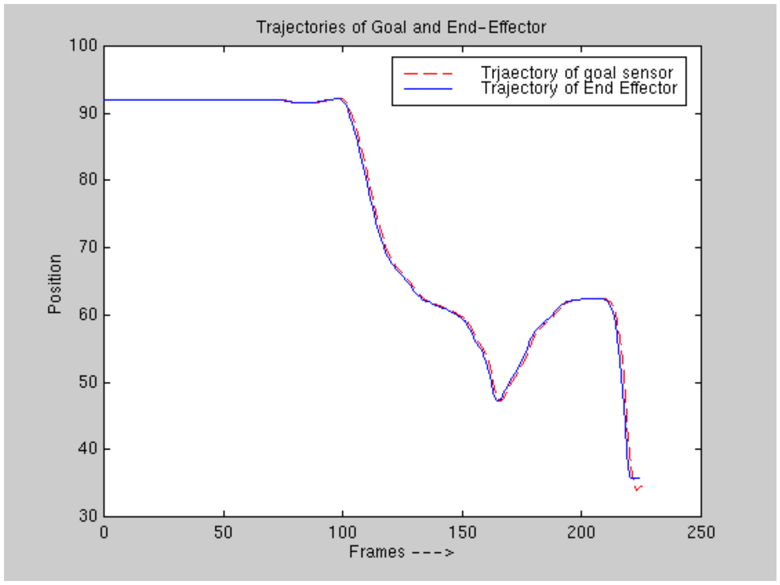


Figure 3.1: End-effector closely following the goal

3.2 Recognition of Spatial Constraints

Spatial constraints between an agent and the objects it interacts with are in general associated with geometric spatial proximities between the end-effectors and the objects. In our system, end-effectors (EE) correspond to the objects (segments in the human model) containing the E sites. For example, *right_palm* is an EE containing an EE site. As the EE sites follow the foal sensors very closely, the sensors themselves can be used to keep track of end-effector locations.

One method for recognizing spatial constraints is to use fast collision detection methods [28, 34] between different objects in the environment to compute the exact time of initial contact. We use the V-COLLIDE ([34] method to compute the spatial proximities of each of the end-effectors with the different objects in the environment. A spatial constraint is recognized when the objects first come in contact with each other. It would be possible, though computationally inefficient, to compute these collisions or proximities at every frame of the animation. We describe the various computational simplifications that we use while still being able to derive all the necessary information for recognizimg the spatial constraints.

3.2.1 Zero-Crossing

In computer vision, zero-crossings of the second derivative are commonly used for edge detection [47] in static images. For example, the Marr-Hildreth operator uses the zero-crossings of the Laplacian of the Gaussian and the Canny operator uses the zero-crossings of the second directional derivative.

In motion analysis, we can use the zero-crossings of the second derivative of the motion data to detect significant changes in the motion. The zero-crossings in acceleration data correspond to the local extrema of the velocity. In motion trajectories, this implies changes in motion such as starting from rest, coming to a stop, or changing the velocity direction. These events were noted to have descriptive significance in [7]. When the zero-crossing point also coincides with an end-effector's contact with another object, it implies a plausible causal relationship between the EE and the object. In motion studies, this further implies that the primary agent came in contact with the object and suggests creating a spatial constraint to mark this occurrence. We record the corresponding global location of the sensor and mark it as a constraint point for the corresponding end-effector of a secondary agent. The zero-crossings enable us to compute the proximities only at *possibly relevant* frames. In Section 4.3, we describe how these zero-crossings can be used effectively to segment an action.

3.2.2 Tracking Sensors

For an action, it is not necessary to track the zero-crossings of all the sensors on the human model. So, for each action, the user can specify the few specific sensors that are active in an action and need to be tracked. For example, in *drink from a mug*, only the sensor on the hand needs to be tracked for zero-crossings. In all actions, the sensor on the head is used as a tracking sensor for capturing the primary agent's attention.

3.2.3 Tag Objects

For an action, it is again not necessary to compute proximities of the tracking sensors with all the objects in the environment. As this entire technique is done as a post-process of the motion-capture session, the specific objects that are involved in the action are already known. Using this knowledge, the user can specify the few objects in the environment that need to be used for computing the proximities. To use these, we introduce *tag objects*.

A *tag object* is associated with an object, type and status. A tag object refers to a 3D object or a part of the 3D object. We can define *tag objects* on parts of the human model, thus allowing us to track body/self interactions. Within the EAI *Jack*®environment, a tag object refers to a segment of a figure. For example, in *drink from a mug*, one of the *tag objects* would refer to the mug, and another *tag object* would refer to the head of the human model. Logic dictates that the correct tag object should be the mouth. But, in our human model, there is no separate segment for the mouth. Hence, we choose the head which is the segment containing the mouth.

The *tag objects* may be of different types:

SELF: The tag object is a part of the human model itself - *e.g.,* head.

FIXED: The tag object does not move in the environment - *e.g.,* table.

MOBILE: The tag object can be moved in the environment. *e.g.,* mug.

In our examples involving mobile objects, we assume that the agent interacts with them by grasping or holding them and moving them to another place. In other words, for at least part of the action, the mobile object is constrained to move with an end-effector of the agent. In such cases, the status flag of the *tag object* indicates if the tag object is CONSTRAINED to the agent or if it is FREE.

3.2.4 Spatial Constraint

The process of automatically recognizing a spatial constraint can be summarized as follows: For each tracking sensor, collision-detection is applied, at every zero-crossing frame, to check for spatial proximity between the tracking sensors and each of the tag objects. This is done by checking for collisions between the end effector containing the tracking sensor and a tag object. If there is a collision, a spatial constraint is said to exist between the tracking sensor and the corresponding tag object. The exact location of the spatial constraint to be used for another agent depends on the type of the tag object and its status (if it is a mobile object). This is discussed in detail in the next section. Figure 3.2 shows

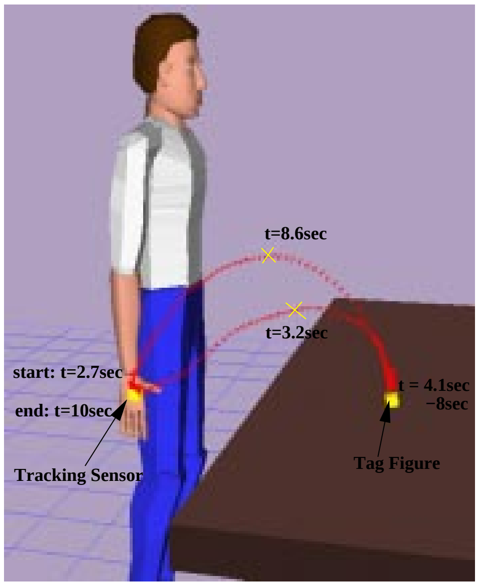


Figure 3.2: Trajectory of the tracking sensor in the example *Touch the Table*

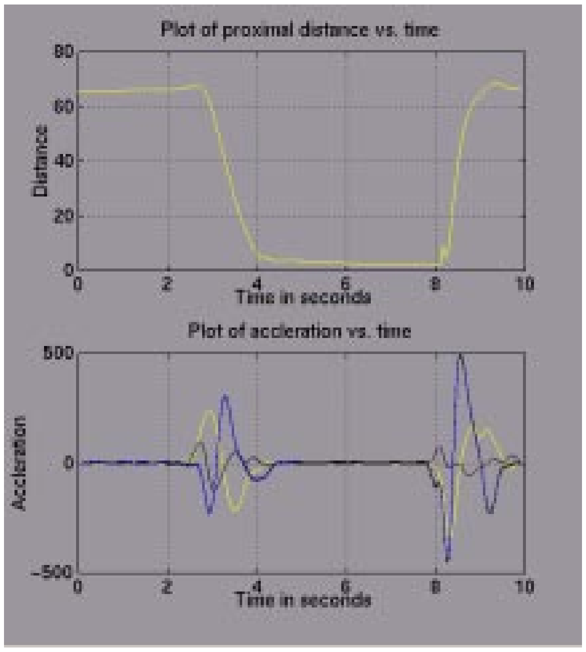


Figure 3.3: Plots of spatial proximity and zero-crossings

the trajectory of the hand tracking sensor for the example *touch the table* and Figure 3.3 shows the corresponding plots of the distance between the tracking sensor (right hand of the agent) and a tag object (table) and the zero-crossings of the accelerations. It can be clearly seen that a spatial constraint is established when the zero-crossings coincide with the close proximity of the tracking sensor and a tag object.

3.3 Determination of Spatial Locations of Constraints

The spatial proximity of each tracking sensor from each tag object is computed at the zero-crossings of the tracking sensor. If a spatial constraint is recognized as outlined above, then the global locations of the constraint need to be used as a constraint location during the secondary agent's action. The global location of the constraint is computed based on the type of the associated *tag object*. If the *tag object* is of type FIXED or MOBILE, then it refers to an external 3D object and the absolute location of the tracking sensor is used as the location of the constraint. But, if the *tag object* is of type SELF, then the relative global location of the tracking sensor is used as the location of the constraint. The relative global location is computed by taking into account the size (lengths of the different segments) of the secondary agent. For example, in *drink from a mug*, for the first spatial constraint established during grasping the mug to pick it up, the absolute global location of the hand sensor at the time of first contact with the mug is used as the location of the constraint. For the second spatial constraint (of the same action) established during holding and bringing the mug to the mouth, the relative global location of the hand sensor when the mug comes in contact with the lips is used. This will cause the secondary agent to grasp the mug at the same location as the primary agent but will hold the mug to his mouth correctly, which may be at a different global location based on the difference in sizes between the two agents.

3.4 Mapping Motions to Other Agents

Once the locations of the spatial constraints are determined, a combination of different techniques may be employed to generate the movements for the secondary agent.

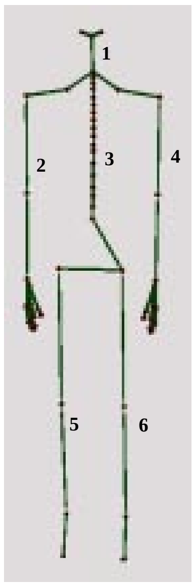


Figure 3.4: Sets of joint chains defined in the human model

Optimization techniques [22, 31, 30] can be used to generate motions for the secondary agent. Here, we describe a few motion generation techniques that allow us to borrow and incorporate different style parameters of the primary agent. As a first step, we segregate the joints in the human body into different kinematic joint chains (Fig. 3.4). We consider each joint chain separately. For the set of joints which are not contained in the same hierarchical chain as any of the tracking sensors, the joint angles computed for the primary agent may be proportionally mapped to the secondary agent. This is possible as they do not have additional constraints imposed on them. All the other joints are driven by the new spatial constraints computed above. As each joint chain is treated separately, it is very important to achieve global synchronization between the different joint chains during the entire action. To do this, we preserve the same timing information *i.e.*, the second agent takes the same amount of time as the primary agent to complete the action.

To solve for the new spatial constraints, a trajectory has to be traced for each joint in the chain containing the tracking sensors. For this, we first use inverse kinematics [61] to solve for the spatial constraints at each of the zero-crossing proximal frames. We then use a linear or cubic spline interpolation in the joint angle space for each *time period* defined between any two successive zero-crossing proximal frames. The interpolating factor can be derived in various ways based on the desired style of motion.

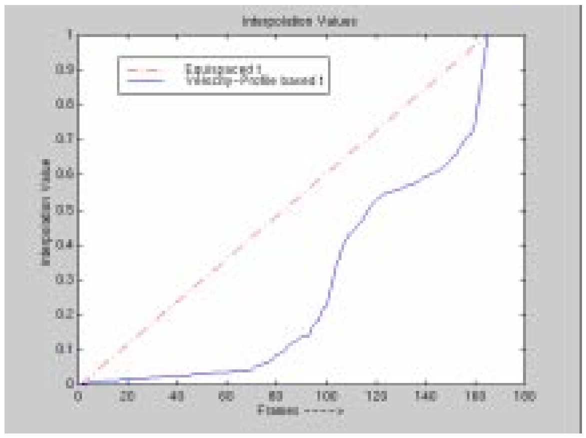


Figure 3.5: Comparison of computed interpolation factor, $\hat{s}$, and equispaced t

3.4.1 Maintaining Velocity Profile

Two motions with completely different end-effector trajectories can share some motion characteristics. Angular velocity is one such characteristic. Here, we define *style* as "frame-wise variations in angular velocity". In an effort to maintain this style of the primary agent, we modify the speed transform method used in [1]. In this case, the interpolating factor $\hat{s}$, for the second motion, is derived by computing the normalized distance moved in the joint space by the primary agent at each frame during the corresponding *time period*:

$$s = \int_0^t |\dot{\theta}(\tau)| d\tau \tag{3.1}$$

where t is time, s is the angular distance moved along the trajectory, and $\dot{\theta}(\tau)$ is the velocity vector of the joint. The data is normalized along the trajectory:

$$t_v = \hat{s} = \frac{\int_0^t |\dot{\theta}(\tau)| d\tau}{\int_0^{t_{end}} |\dot{\theta}(\tau)| d\tau} \tag{3.2}$$

where t_{end} is the duration of the basic period and t_v is the velocity-profile based interpolating factor. Fig 3.5 clearly shows the difference between the common equispaced interpolating factor, t, and the derived interpolating factor, $\hat{s}$. These interpolating values help maintain the angular velocity profile (Fig 3.6) of the primary agent during the course of the action and is independent of the difference in spatial distance covered during each *time period* by the two agents. What is the effect of having similar angular velocities? During the motion, if the first agent paused, started from rest slowly, increased the speed of motion, slowed down, rested for some time, and then increased speed again, the secondary agent's

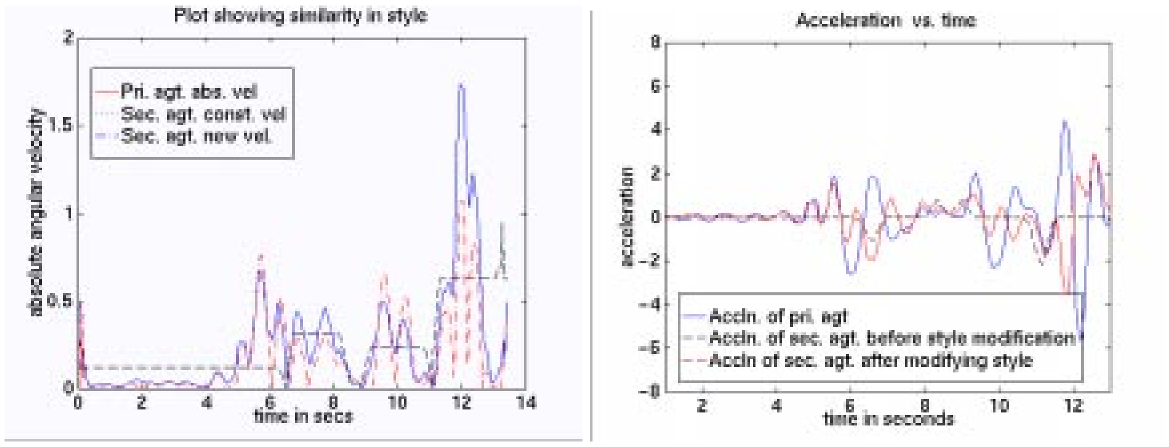


Figure 3.6: Similarity in velocity profiles and invariance of zero-crossing points

motion would follow the same pattern even when the trajectory is completely different. Also, as seen from the second plot in Fig 3.6, the zero-crossings of acceleration are found to be invariant. With an equispaced interpolating factor, the angular velocity would have been very flat and unnatural. But, here, we are able to retain the characteristics of the original motion.

3.4.2 Following End-Effector Trajectory

Here, we define *style* as "variations in the path". So, we use this method when we need the trajectory of the secondary agent's end-effector to follow the shape of the primary agent's end-effector trajectory. In [20], Choi *et al.* use optimization and cubic spline interpolation techniques to correct errors in end-effector positions while preserving the characteristics of the original joint angle data. But this method will be applicable only for correcting the data of the same agent or for applying the motion to another agent of the same size. When we consider mapping a constrained motion to a completely different sized virtual model, one of the end-points in the new motion will not match the original. For example, when we consider mapping a "reach" motion from an adult model to a 9 year old child model, the starting position of the two right-hand end-effectors will not be the same even if both the models are in the same posture. This is due to the difference in their sizes. Hence the trajectory traced by the two end-effectors will be different. But, it is possible to have similar-shaped trajectories.

As mentioned in Section 3.2.1, the zero-crossings in acceleration data correspond to local extrema of the velocity and imply changes in motion such as starting from rest, coming to a stop, or changing the velocity direction. Above, we have used the zero-crossings to detect occurrences of spatial constraints. In a motion segment that has constraints only at the end-frames, we can use the zero-crossings of the acceleration data at the intermediate frames to monitor significant changes in the trajectory. We compute new constraints for the end-effector at each of the intermediate zero-crossing frames, while retaining the constraints at the end-frames as computed in Section 3.3. These constraints are again solved by our IK techniques. The resulting joint angles are interpolated using cubic splines.

The new constraint locations of the end-effector at the intermediate zero-crossing frames are computed independently in the three axial directions (x,y, and z). We only compute new positions for the constraints and retain the original orientations of the primary agent's end-effector at the corresponding frames. The new constraint positions are calculated as follows:

$$\begin{aligned} x' &= \left(\frac{x - x_{start}}{x_{end} - x_{start}}\right)(x'_{end} - x'_{start}) + x'_{start} \qquad (3.3)\\ where\ x, y, z &= \text{end-effector coordinates of primary agent in any frame} \\ x', y', z' &= \text{end-effector coordinates of secondary agent in any frame} \\ x_{start}, x_{end} &= \text{x coords of primary agent's end-effector in start and end frames} \\ x'_{start}, x'_{end} &= \text{x coords of secondary agent's end-effector in start and end frames} \end{aligned}$$

The y and z coords are computed similarly. As an example, we motion captured a person touching a table. The person was asked to approach the table in an indirect way - move the arm randomly before touching the table. We mapped this motion to two secondary agents - one was a clone of the primary agent and the other was a 9 year old child model. Fig 3.7 shows the results of applying this technique for computing the secondary agent's end-effector. We find that when the secondary agent is a clone of the primary agent, the end-effector trajectories are exactly the same. In the case of the smaller secondary agent, the end-effector trajectory is not the same, but has the same shape as the primary agent's end effector trajectory while maintaining all the spatial constraints. Also, the trajectories maintain a similar velocity profile.

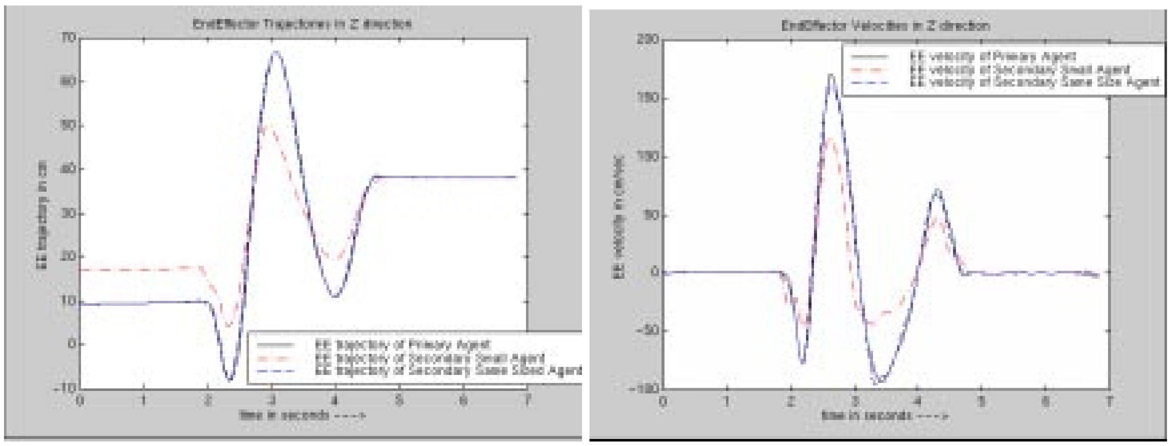


Figure 3.7: Similarity in end-effectors' trajectory shape and velocity profile using the "variations in the path" style for *touch table* action

3.4.3 Mapping End-Effector Forces

Here, we define *style* as "variations in the applied force". We use this method when we need to recognize and map force interactions between the agent's end-effector and an object. Our first goal is to recognize the occurrence and amount of force exerted by the agent on the object. Our second goal is to map these forces to the secondary agent's action and then compute the effect of the force on the object.

While mapping force, we can add additional constraints of "similar force" or "same object motion". For the "similar force" constraint, we proportionally scale the force applied on the object by the agent and then generate a corresponding new motion path for the object. Here, the primary and secondary agents have similar end effector trajectories and exert proportional forces while the object may have completely different trajectories. The object's new motion is a causal effect of the secondary agent's end effector exerting force on it. So, the new trajectory is completely dependent on the exerted force. For the "same motion" constraint, we can constrain the object to have the same motion independent of the agent applying force on it. In this case, based on the object's motion and the agent's size, we would need to compute both the force applied by the secondary agent and the corresponding end-effector trajectory. But, it may be possible that the secondary agent is unable to exert the required force to move the object exactly and the whole system may fail. To remedy this, we could employ some strategic methods, like generate muscular forces to exert the additional forces. But, we are not interested in those approaches. Hence, we choose to use the first technique of "similar force" as its results are natural.

To do this, we consider the example of the primary agent hitting a box, resting on a table, with his hand causing the box to move. In this example, we need to recognize the amount of force exerted by the primary agent on the box. Then, for any given secondary agent, we need to compute the corresponding force with which the agent would hit the box and the resulting path taken by the box after it is hit.

In [33], Hodgins et al., describe a technique that adapts an example behavior to the physical characteristics of a new character. This technique works by scaling control system parameters based on a dynamic analysis of the two characters. They define two different types of scaling - geometric and mass. Geometric scaling is done only for geometrically similar characters where it is assumed the scaling is uniform in all dimensions and that densities and acceleration due to gravity are the same for the two characters. The control system parameters are scaled based on a scaling factor which depends on the action being performed. For walking or running, the scaling factors are based on the character's height and leg length. For bicycling, they used the ratio of wheel radii (but, what if both agents are riding the same bicycle?). But, geometric scaling alone may not be sufficient to scale all actions correctly. The main problem may be attributed to a single scaling factor. A scale factor computed based on relative leg lengths may not be appropriate for scaling the gains that control the arm motion. To overcome this problem, they scale each gain factor at the different joints separately depending on the function of the joint. Also, if the figures are not uniformly scaled, the resulting behaviors may not be dynamically similar. Hence, they also use mass scaling which corrects for differences in masses and relative moments of inertia. As correctly pointed out by the authors, it is not sufficient to just scale the motion when the action involves interaction with the environment. We need to study the variation of motion.

In our technique, we use physics based models to generate the motion of the object. The parameters for the model, like coefficient of restitution and damping coefficients, are obtained from direct observation of the primary agent's interaction with the object. In all our examples, we use the following steps to map and generate the action to the secondary agent.

1. Compute the exact time of contact between the primary agent and the object.
2. Compute the coefficient of restitution from the primary agent's contact with the object.
3. Compute the initial velocity of the object and the impulse force exerted by the secondary agent's end effector on the object.
4. Compute the new trajectory of the object after impact.

We will now discuss each of these steps in detail.

3.4.3.1 Time of contact

As described in Section 3.2, we use the concepts of zero-crossing and spatial proximity to determine the time of contact between the primary agent's end effector and the object.

3.4.3.2 Coefficient of Restitution

We compute the coefficient of restitution from the primary agent's end effector velocity and the corresponding object's velocity. We describe the method of doing this by first defining the concept of coefficient of restitution as explained in [12].

When two bodies A and B with velocities v_A and v_B collide, they will deform under the impact. At the end of the deformation, they will have velocities u_A and u_B which have equal components along the line of impact. A period of restitution will then take place, at the end of which A and B will have velocities v'_A and v'_B. If the bodies are assumed to be frictionless, then they exert forces $\int Pdt$ during the period of deformation and $\int Rdt$ during restitution on each other along the line of impact. The coefficient of restitution is defined as the ratio

$$e = \frac{\int Rdt}{\int Pdt} \tag{3.4}$$

This can be further simplified such that

$$e = \frac{(v'_B)_n - (v'_A)_n}{(v_A)_n - (v_B)_n} \tag{3.5}$$

In our example of observing an agent's interaction with an object, we use sensors to track both the primary agent's end effector and the object. Hence, we easily compute the

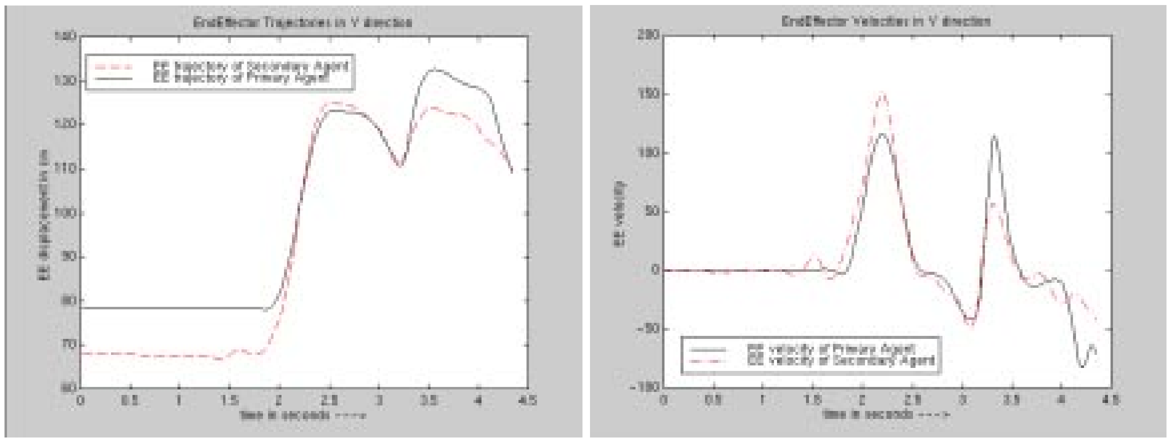


Figure 3.8: End-effector trajectories and velocities for the *push box* action, using the "variations in path" style

velocities of the end-effector and the object before and after impact. We then compute the coefficient of restitution using equation 3.5.

3.4.3.3 Initial Velocity and Impulse Force

To compute the initial velocity of the object and the impulse forces that will be exerted on the object by the secondary agent at the time of contact, we first need to compute a new trajectory for the secondary agent's end effector and then calculate the velocity and acceleration at the time of contact. We use the style of "variations in the path" (Section 3.4.2) of the primary agent's end effector to the object to compute the new trajectory for the secondary agent's end-effector. This is done independent of any other object's trajectory. The two motions of the primary and the secondary agents will then have similar end-effector trajectories and velocity profiles as shown in Fig 3.8. At the contact frame, we use the velocity of the secondary agent's end effector, the computed coefficient of restitution, and equation 3.5 to determine the initial velocity of the object after impact.

We use the velocity and acceleration of the end effector at the time of contact to compute the impulse force exerted by the end effector at the time of contact. The force is calculated using $F = ma$ in the direction of the velocity. As we are considering linked manipulators, we consider the mass of the entire link and assume it to be centered at the end-effector. The masses of the segments have been precomputed using anthropometric distributions and so are more precise than simple scaling based on the limb lengths [33].

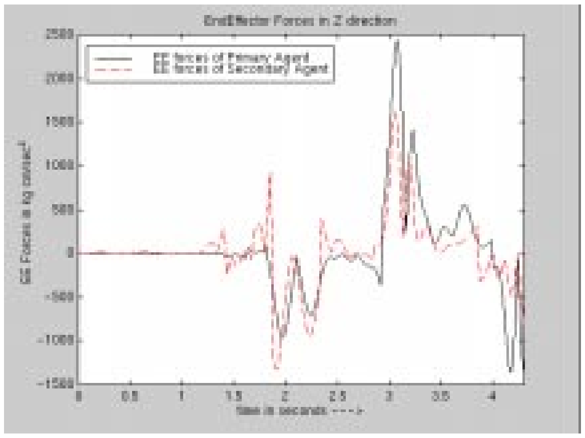


Figure 3.9: End-effector forces for the *push box* action, using the "variations in path" style

Fig 3.9 shows the end effector forces at all frames for both the primary and the secondary agents. But we are interested in the forces only at the time of impact.

3.4.3.4 Trajectory of Object

We use the impulse force computed above to compute a new trajectory for the object. This external impulse force exerted by the agent on the object is active only at the instant of impact. At all other instants, only the damping and the gravitational forces act on the object.

A damping force has the form $\mathbf{f} = -k_d\mathbf{v}$ where k_d is the called the *damping coefficient.* The effect of a damping force is to resist motion, make a particle or rigid body gradually come to rest in the absence of other influences. Here, we describe a technique to compute the damping coefficient from a given action.

As discussed earlier, we assume that after impact, the damping force is the only force acting on the object till it comes to rest. We can represent this relationship as

$$\mathbf{M}\ddot{x} = k_d\dot{x} \tag{3.6}$$

where $\mathbf{M}$ is the mass of the object, $\ddot{x}$ and $\dot{x}$ are the acceleration and velocity of the object respectively. From this, we can calculate the damping coefficient, k_d as

$$k_d = \mathbf{M}\ddot{x}/\dot{x}. \tag{3.7}$$

In our example of an agent hitting the box, the box, after impact, moves on the top surface of the table before falling off. We consider the path of the object while it is in contact with the table after impact with the primary agent's end-effector. During this phase, we compute the velocity and acceleration of the object. We find that $\ddot{x}/\dot{x}$ *vs.* time is fairly linear with respect to time except at the very end. As an approximation, we consider the linear portion of the path and fit a straight line to it using the method of least squares regression. To do this, we consider the equation of the straight line

$$y = a_0 + a_1 x \tag{3.8}$$

The constants a_0 and a_1 can be computed from

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} \tag{3.9}$$

$$a_0 = y_{mean} - a_1 x_{mean} \tag{3.10}$$

The slope of this straight line, a_1, indicates how the damping coefficient varies over time. To find the actual damping coefficient, k_d, we need to integrate this over time. So,

$$k_d = a_1 \int dt = a_1(t_2 - t_1) \tag{3.11}$$

Hence $k_d = a_1$. As the motion of the object on the surface of the table is in the global x-y plane, we compute the damping coefficients separately in the x and y directions. The damping in the z direction is assumed to be 0.

To compute the trajectory, we consider the state space of the object. As the object is a 3D rigid body, the state space comprises the position, orientation, linear and angular momentums of the object and is represented as

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} \tag{3.12}$$

The new location of the object at each time step is calculated [11] by solving the following ordinary differential equations (ODE) of motion using the Runge-Kutta method.

$$\text{Linear velocity } v(t) = \dot{x(t)}$$

$$\dot{R(t)} = \omega(t) * R(t) \text{ where } \omega(t) \text{ is the angular velocity}$$

$$\begin{aligned} \text{Linear Momentum } P(t) &= Mv(t) \\ \dot{P(t)} &= F(t) \text{ where F is the force} \\ \text{Angular Momentum } L(t) &= I(t)\omega(t) \\ \dot{L(t)} &= \tau(t) \text{ where } \tau(t) \text{ is the torque} \quad (3.13) \\ & \quad (3.14) \end{aligned}$$

The new locations are calculated at consecutive time-steps until convergence. Initially, only the computed external force and its corresponding torque are applied as impulse force and impulse torque. At subsequent time steps, only the damping and gravitational forces are used. The torques are zero.

During the motion of the object, we need to consider different situations:

1. The resulting motion of the object is planar - i.e. it moves only on the surface on which it is resting. This would be the simple case and is currently implemented.

2. The resulting motion is 3D and hence may move above the surface and fall due to gravity. In this case, we need to consider subsequent collisions of the object with other objects/surfaces in the environment. In case of collisions, we need to prevent penetration and hence may need to introduce more impulse moments at the points of collision. To do this, after a new location for the object has been computed by the ODE solver, we check for collisions against user specified non-penetrable faces in the environment. In our example of an agent hitting a box, the colliding faces are the top of table on which the box is resting and the ground. If there is a collision, the point on the object that has penetrated the most is identified and the whole object is moved back to the non-penetrating face by the amount of penetration. The normal component of the object's velocity is then reversed and scaled by the coefficient of restitution. So,

$$v_n = -k_r * v_n \quad (3.15)$$

3.5 Visual Attention Tracking

Capturing and maintaining visual attention is very important for movement realism in the secondary agent. Without it, actions appear unnatural even if all the other constraints are correctly satisfied. For example, while picking up an object, the secondary agent would look extremely unnatural if she looked at some other point in space. Here, we use the above technique to easily address the visual attention constraint.

During interaction with objects, we tend to always look at the object that we are interacting with (at least when we first come in contact with it). This direction is automatically captured for the primary agent by the sensors on the head. If we naively map head motions of the primary agent to a different secondary agent, this gaze direction will be lost and *cannot be re-captured by simple signal processing techniques.* Instead, we define the sensor on the head as a *tracking sensor.* The zero-crossings in the acceleration space of the head sensor indicate a change in gaze direction or indicate gaze at a specific point in space. During these zero-crossings, we check for visual attention constraints by using the line of sight of the agent. For efficiency, we compute the intersections of the line of sight with the bounding boxes of the tag objects only. If there is any tagged object in the direction of the line of sight, the global location of the point of attention is calculated and used as the visual (alignment) constraint for the secondary agent during its motion computation. We use a head-eye tracking model to solve for the joint angles in the eyes, head, and neck at the gaze direction zero crossing frames. For the remainder of the frames, we use joint angle interpolation while maintaining the angular velocity profile as outlined in section 3.4.1.

3.6 Summary

In this chapter, we first described the process of recognizing and mapping spatial and visual constraints. We then showed how the various style parameters could be derived and mapped to secondary agents.

We have tested this technique by mapping the actions of an adult to the virtual model of a nine year old child. We have captured *touching a table* which involves only a FIXED tag object and *drinking from a mug* which involves a MOBILE object. In both cases, we were able to successfully recognize the spatial constraints and map the motions correctly to the second agent. Of these, the example of *drink from mug* is more complicated and we discuss this in detail here.

In the example of *drink from mug*, the primary agent bends over, picks up a mug from the table, drinks from it and places it back on the table. In this case, there are two tag objects - mug (MOBILE) and head (SELF). There are three spatial constraints - the agent's hand grasping and picking up the mug, the mug touching the mouth of the agent, and the agent's hand putting down the mug. The locations of the spatial constraints remains the same for the secondary agent while picking up and putting down the mug as the mug is of type EXTERNAL. But, a new spatial constraint based on the size of the secondary agent has to be calculated for the constraint of the mug coming in contact with the mouth of the agent while drinking. Figure 3.10 shows the plots of the trajectories of the hand

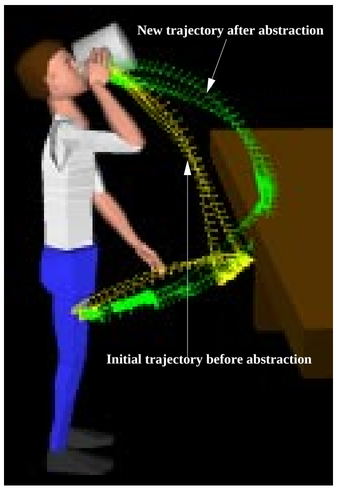

New trajectory after abstraction
Initial trajectory before abstraction

Figure 3.10: Trajectory plots of the secondary agent's hand end-effector (corresponding to the tracking sensor on the hand of the adult) before and after abstraction

end-effector of the secondary agent before and after abstraction. Before abstraction, the trajectory obtained is the result of direct mapping of the joint angles of the primary agent to the child model. It can be clearly seen that the constraint of picking the mug cannot be satisfied. But after the automatic recognition of spatial constraints and subsequent remapping of the motions as outlined in this chapter, the motion of the secondary agent is corrected as can be seen by the modified trajectory.

Figure 3.11 shows the various stages of the drinking motion as captured for the primary agent. Figure 3.12 shows the various stages of the drinking motion for the secondary agent after abstraction and mapping have been applied.

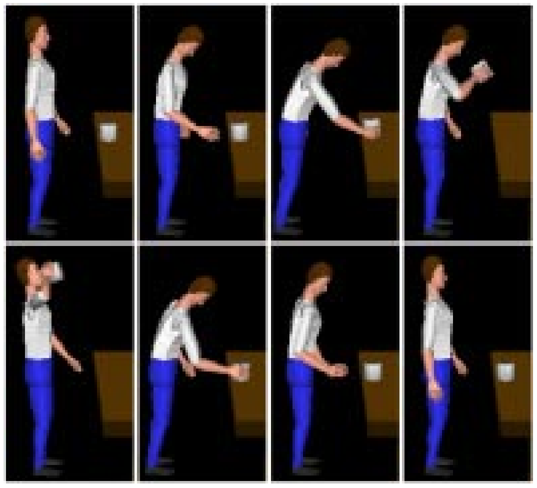


Figure 3.11: Different stages in *drink from mug* of the primary agent (adult male)

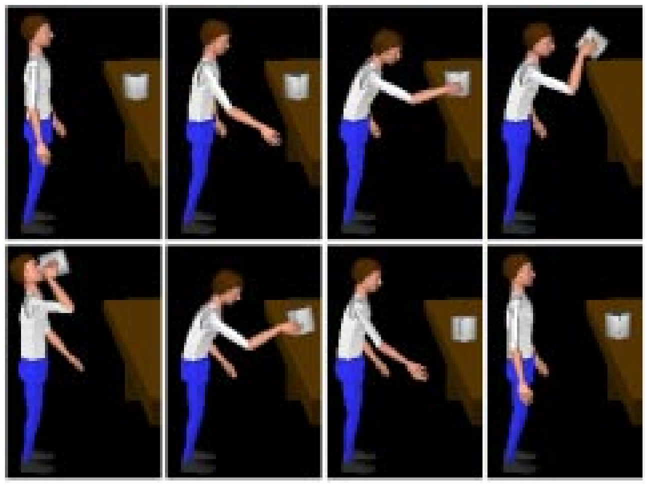


Figure 3.12: Different stages in *drink from mug* of the secondary agent (a nine year old child) - after mapping

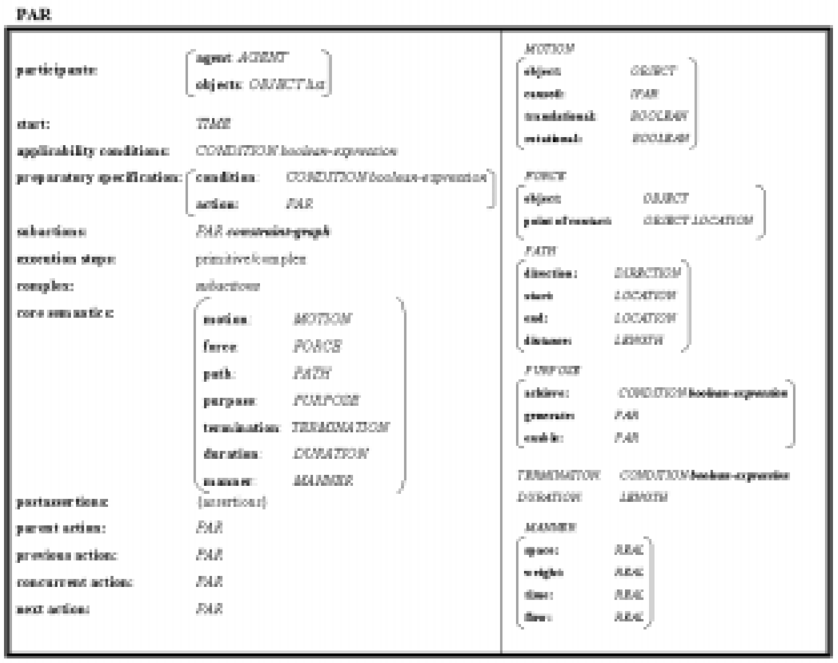
Chapter 4

PAR Generation

The main goal of this thesis is to build agent-size neutral semantic representations of actions from observing a single trial of a person's actions. In this chapter, we introduce the CaPAR system to achieve this. In Chapter 3, we introduced techniques to abstract semantic (kinematic and dynamic parameters) and style information from an action and generate similar constrained actions for differently sized virtual models. Now, we extend this notion to use the extracted information to build conceptual representations of actions in the form of Parameterized Action Representations (PAR). We describe the main features of a PAR in Section 4.1, and we describe the architecture of the CaPAR system in Section 4.2. In the rest of the chapter, we discuss the various components of the CaPAR system in detail.

4.1 Parameterized Action Representation (PAR)

The PAR [8] was conceptualized to bridge the gap between natural language and animation. A PAR gives a description of an action. The PAR has to specify the agent of the action as well as any relevant objects and information about path, location, manner, and purpose for a particular action. There are linguistic constraints on how this information can be conveyed by the language; agents and objects tend to be verb arguments, path is often a prepositional phrase, and manner and purpose might be in additional clauses [51]. A parser and translator map the components of an instruction into the parameters or variables of



PAR

participants:	**agent:** *AGENT*; **objects:** *OBJECT* list
start:	*TIME*
applicability conditions:	*CONDITION* boolean-expression
preparatory specification:	**condition:** *CONDITION* boolean-expression; **action:** *PAR*
subactions:	*PAR* **constraint-graph**
execution steps:	primitive/complex
complex:	subactions
core semantics:	**motion:** *MOTION*; **force:** *FORCE*; **path:** *PATH*; **purpose:** *PURPOSE*; **termination:** *TERMINATION*; **duration:** *DURATION*; **manner:** *MANNER*
partassertions:	{assertions}
parent action:	*PAR*
previous action:	*PAR*
concurrent action:	*PAR*
next action:	*PAR*

MOTION

object:	*OBJECT*
caused:	*PAR*
translational:	*BOOLEAN*
rotational:	*BOOLEAN*

FORCE

object:	*OBJECT*
point of contact:	*OBJECT LOCATION*

PATH

direction:	*DIRECTION*
start:	*LOCATION*
end:	*LOCATION*
distance:	*LENGTH*

PURPOSE

achieve:	*CONDITION* **boolean-expression**
generate:	*PAR*
enable:	*PAR*

TERMINATION	*CONDITION* **boolean-expression**
DURATION	*LENGTH*

MANNER

space:	*REAL*
weight:	*REAL*
time:	*REAL*
flow:	*REAL*

Figure 4.1: Syntactic representation of a PAR

the PAR, which is then linked directly to PaT-Nets [32] executing the specified movement generators. Natural language often describes actions at a high level, leaving out many of the details that have to be specified for animation [48]. The PAR must provide links to omitted details. We use the example "Walk to the door and turn the handle slowly" to illustrate the function of the PAR. Whether or not the PAR system processes this instruction, there is nothing explicit in the linguistic representation about grasping the handle or which direction it will have to be turned, yet this information is necessary to the action's actual visible performance. The PAR has to include information about applicability, preparatory, and termination conditions in order to fill in these gaps. It also has to be parameterized, because other details of the action depend on the PAR's participants, including agents, objects, and other attributes. The representation of the "handle" object lists the actions that the object can perform and what state changes they cause [25, 36]. The number of steps it will take to get to the door depends on the agent's size and starting location. Next, we briefly describe some of the terminology and concepts used to define a PAR.

```
type agent representation =
        (coordinate-system: site;
         state: state space;
         rel-dir: relative directions;
         special-dir: special directions;
         grasp-sites: sequence site;
         capabilities: sequence actions-and-applicability;
         nominals: sequence value-ranges).


type actions-and-applicability =
        (action: parameterized action;
         applicability: sequence disjunctive-queries).

type value-ranges =
        (var: powerset parameter;
         mean: powerset var-types;
         standard deviation: powerset var-types;
         distribution: powerset distribution).

type parameter =
        (id: string).

type var-types =
        (real, real vector, integer).

type distribution =
        (normal, poisson, uniform).
```

Figure 4.2: The agent representation type

4.1.1 PAR Terminology

Some of the parameters in a PAR template are shown in Fig 4.1:

Participants:

Agent: The agent executes the action. An agent, considered as a special type of object, has a number of properties and is stored as part of the hierarchical object database. Fig 4.2 [6] shows all the current properties of an agent.

Objects: The object type is defined explicitly for a complete representation of a physical object and is stored hierarchically in a database. Each object in the

```
type object representation =
        (coordinate-system: site;
         state: state space;
         rel-dir: sequence relative direction;
         special-dir: sequence special direction;
         grasp-sites: sequence site;
         actions: sequence parameterized action).

type site =
        (position: real vector;
         orientation: real vector).

type state space =
        (position: real vector;
         velocity: real vector;
         acceleration: real vector;
         force: real vector;
         torque: real vector).

type relative direction =
        (name: (front, back, left, along, inside);
         value: real vector).

type special direction =
        (name: string; value: real vector).
```

Figure 4.3: The object representation type

environment is an instance of this type and is associated with a graphical model in a scene graph. The state field of an object describes a set of constraints on the object that leave it in a default state. The object continues in this state until a new set of constraints is imposed on the object by an action that causes a change in state. The other important fields are the reference coordinate frame, a list of grasp sites and their purpose, and intrinsic directions (top, front, etc.) defined with respect to the object. In our example, the walking action has an implicit floor as an object, while the turn action refers to the handle. Fig 4.3 [6] shows all the current properties of an object.

Start: This is the time at which the action begins.

Applicability Conditions: The applicability conditions of an action specify what needs to be true in the world in order to carry out an action. These can refer to agent capabilities, object configurations, and other unchangeable or uncontrollable aspects of the environment. The conditions in this boolean expression must be true to perform the action. For “walk,” one of the applicability conditions may be “Can the agent walk?” If conditions are not satisfied, the action cannot be executed. The applicability conditions of an action have to be set explicitly by the user and cannot be recognized or generated automatically by the CaPAR system.

Preparatory Specifications: This is a list of <CONDITION, action> statements. The conditions are evaluated first and have to be satisfied before the current action can proceed. If the conditions are not satisfied, then the corresponding action is executed; it may be a single action or a very complex combination of actions, but it has the same format as the execution steps described below. In our example, one of the conditions to be checked could be *posture(agent)==stand* and the corresponding action could be (“stand”,agents: (“Jack”)). If the agent is in a sitting posture or prone posture, then the action causes him to change to the standing posture. The preparatory specifications are completely generated by the CaPAR system at run-time.

Execution Steps: A PAR can describe either a primitive or a complex action. The execution steps contain the details of executing the action after all the applicability and preparatory conditions have been satisfied. If it is a primitive action, the underlying Pat-Net for the action is directly invoked. A complex action can list a number of sub-actions that may need to be executed in sequence, in parallel, or as a combination of both. A complex action can be considered done if all of its sub-actions are done or if its explicit termination conditions are satisfied. The CaPAR generates the execution steps for both primitive and complex actions.

Core semantics: The core semantics represent an action’s primary components of meaning and include motion, force, path, purpose, termination conditions, duration, and agent manner.

Motion: This specifies the object that is being moved and the type of motion - rotational, translational or both. It also specifies if this is a causal motion.

Force: This points to the affected object and indicates the force or torque amount and point of contact.

Path: This contains information on the location of the object at the beginning or the end of the motion and the directional changes that model the approximate path of the motion.

Purpose: This specifies the state or condition that will be achieved as a result of this motion. It also points to the PARs that are either generated or enabled during the course of or at the end of the motion.

Termination Conditions: This is a list of conditions that, when satisfied, complete the action. These can be generated from natural language or from motion capture. From natural language, the termination condition can be determined from the main verb or attached clauses [16]. From motion capture, the CaPAR system generates this automatically at run-time as explained in Section 5.3.2.

Duration: This specifies the duration of the motion.

Manner: Manner specifications describe the way in which an agent carries out an action. As explained in Section 3.4, the CaPAR system abstracts and derives an agent's style or manner of doing the action.

Post Assertions: This is a list of statements or assertions that are executed after the termination conditions of the action have been satisfied. These assertions update the database to record the changes in the environment. The changes may be due to direct or side effects of the action.

4.1.2 PAR Representations

A PAR takes on two different forms: uninstantiated (UPAR) and instantiated (IPAR). We store all instances of the UPAR, which contains default applicability conditions, preparatory specifications, and execution steps, in a hierarchical database called the Actionary. An IPAR is a UPAR instantiated with specific information on the agent, physical object(s), manner, termination conditions, and other bound parameters. Any new information in an IPAR overrides the corresponding UPAR default. The CaPAR system generates UPARs for the observed actions.

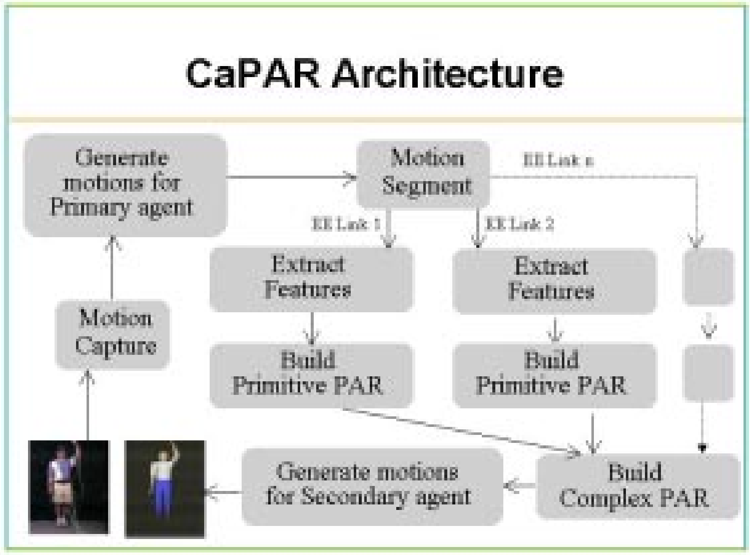


Figure 4.4: CaPAR architecture

4.2 CaPAR Architecture

In this section, we introduce and describe the CaPAR interactive system that we have built to automatically abstract, understand, recognize and parameterize a complex physical action into a PAR such that the actions can be reproduced by any sized virtual human model. There are two main applications of this system. The first application is to generate UPARs from an observed action and the second is to provide a means for the user to custom generate the secondary agent's motion from the extracted UPAR.

The architecture of the CaPAR system is shown in Fig. 4.4. The actions of a person performing a complex physical action is first motion captured. These actions are then generated directly for a virtual human model built to the same size (referred to as a primary agent) as the performing human. This process has been described in Section 3.1. The system then uses this resulting motion in the primary agent for further abstraction and analysis. The user interacts with the CaPAR system through a graphical user interface (GUI) and initiates the process of generating PARs by specifying the tracking sensors (Section 3.2.2 and the tag objects (Section section:tagobject) in the environment.

Our main goal is to automatically build a PAR for an observed complex action. As the structure of the PAR is itself very complex, it would become an extremely large and difficult problem to directly build a PAR for the entire complex action in one step. Instead, the system segments the given action into several sub-actions. Each of these sub-actions would correspond to a primitive action. The system first generates a primitive PAR for each of these sub-actions and then combines them together to form a complex PAR.

As shown in Fig. 4.4, the first task after action segmentation is to extract the relevant semantic features and store them in a feature table for the purposes of PAR characterization and PAR generation. During PAR characterization, a specific UPAR is identified for each primitive action. Whenever a new action is observed, the CaPAR system extracts the relevant semantic features and groups them together to form a feature set. Each feature set is then compared against existing feature sets in the feature table. The feature table is initially empty. If there is a match, then the motion segment is labeled with the matched UPAR name. Otherwise, the system prompts the user for a new UPAR name and stores it along with the feature set in the feature table. During this process, the user interacts with the system to provide any missing parameters or to resolve any ambiguity that arises from the automatic extraction and labeling of the actions. For example, the system may find multiple UPARs in the feature table that match the newly extracted feature set. The user is then asked to identify a single appropriate label. At this time, the user could also reject the system's suggestions and prefer to create a new UPAR. In the latter case, the extracted feature set is first inserted at the appropriate place in the feature table tree and associated with the user-specified UPAR name.

During PAR generation, the system creates a new UPAR, and sets some of the parameters like execution steps, core semantics, parent action, end-effector, etc. It also updates the appropriate object model with the constraint locations. Based on some of the motion and agent characteristics at the beginning and at the end of the motion segment, the system suggests some preparatory conditions and termination conditions. The user can then use this information to create appropriate preparatory specifications, termination and applicability conditions. Finally, at the user's request, the system stores the extracted agent's style of action (Section 3.4) in the agent model.

During the process of motion generation for the secondary agent, the user can interactively custom generate the motion by specifying the agent, the objects and new parameter values for constraint locations, style of action, etc. All user specified values override the default extracted values and new motions can be generated as described in Section 4.7.

In the rest of this chapter, we explain the components of the CaPAR system in detail. In Section 4.3, we explain the process of action segmentation based on correct action perception. In Section 4.4, we describe the set of features that we isolate for unique characterization of primitive PARs. In section 4.5, we describe the process of characterizing a PAR from the extracted semantic features. In Section 4.6, we describe in detail the process of generating both a primitive PAR and a complex PAR. Finally, in Section 4.7, we describe the process of generating motions from these PARs for any given secondary agent.

4.3 Action Segmentation

Experiments [49] show that people normally segment events into actions, even when they are not required to do so by instructions. Also, we know by experience that it is easier to learn complex tasks by breaking them into simpler tasks and then learning them. Accordingly, while observing a complex action, we first break it up into multiple simple or primitive actions and analyze each separately. This breakup or segmentation has to be perceptually correct.

In Section 4.3.1, we first discuss related studies and techniques of action segmentation based on visual perception. Using the results of these studies, we describe our technique of action segmentation in Section 4.3.2.

4.3.1 Action Perception

The process of generating a PAR from observing a person's action correlates to building a conceptual representation of an action based on visual perception. Miller [45] stated that the analysis of perception required at least four primitive categories: objects and attributes, states and events. We define the following terminologies to help us better understand the concepts of perception.

Change: Change denotes the perception that the pattern of simulation or a property of an object or motion is different from the pattern of simulation or property at the preceding moment [45]. The thing that is changed is called a state.

Event: An event is something that happens at a given place and time [27]. Events are transient and have a temporal and spatial location. They correspond to a distinctive local change set in a context of great redundancy from moment to moment [45]. Hence, events correspond to changes. But, not all changes are events. While building conceptual representations, a change constitutes an event when the sensory input to the system causes an update in the conceptual representation of the world.

Motion: Motion or movement is the most atomic primitive and involves a change in the position or location of something [14]. It does not require any contextual or sequential knowledge to be recognized.

Action: Action is a larger scale event which encompasses motions and typically includes causal relationships and interaction with the environment [14].

Causality Perception: Causality is the relation between causes and effects [27]. Causality perception relates to recognizing causality using a few sensory inputs. One example of causality perception is when people observe an object colliding with another object, and perceive that the first object caused a motion in the second object. Michotte [43] called this type of perception an ampliation.

Motion Perception: In motion perception, two different spatially separated motions are perceived as one coordinated movement [60]. For example, the combined motion of the segmented parts of an arm is perceived as one single arm motion.

Action Perception: Action perception involves discrete motions of objects [60]. Here, emphasis is as much on conditions in the beginning and ending of motions as it is with the motions themselves. In [43], Michotte described action perception as perceiving individual actions which may not always depend on the perception of causality.

Newston *et al.* [49] carried out a number of experiments to study the relation between the segmentation of ongoing behavior sequences into their component actions and the movement in those sequences. They determined that humans perceive an action stream

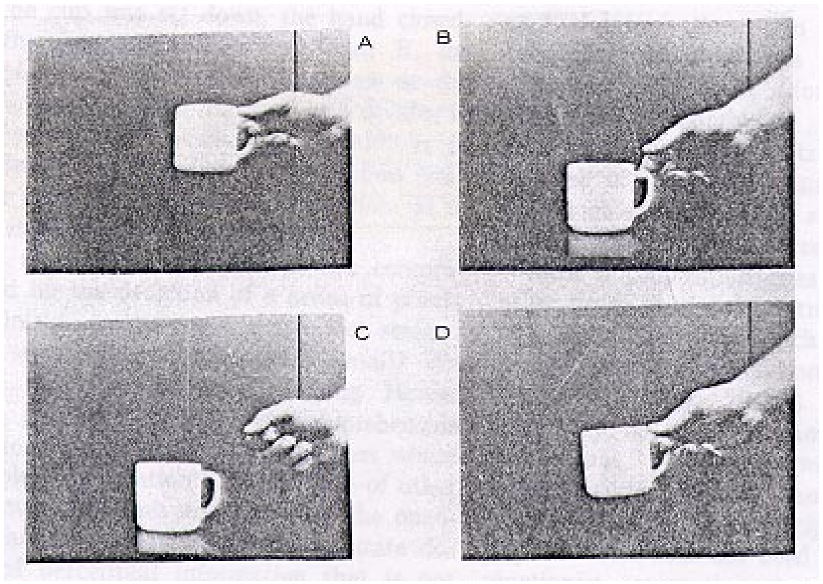


Figure 4.5: Newston's example of a point-to-point action definition

as a sequence of clearly segmented "action units" with easily discriminable boundaries between them. These boundaries, also referred to as breakpoints or action-definition points, were found to have distinctive properties that differentiated them from other parts of the behavior stream. The breakpoints contained defining information for one action while providing a basis of discrimination from the preceding action. For their experiments, informants watched an action sequence and were asked to indicate their perception of when an action occured by pressing a button. They separately considered two very simple examples of putting a cup down and picking up a cup as shown in Fig. 4.5. On observation, the informants found point B to be the breakpoint in the put-down action. This corresponds to the instant just after the cup is put-down on the table and the hand is released. Similarly point D, when the cup just leaves the surface of the table, is a breakpoint in the pick-up action. From their experiments, they concluded that the key features useful for action recognition are mainly detected during a transition from one state to another and not during the action itself. Their findings were found to be consistent with Miller and Laird's theory [45] that events are perceived when changes occur.

In [60], Thibadeau proposed building a system to computationally perceive actions using Newston's points of action definition. He derived knowledge representations of actions using logico-linguistic derivations. For this, he used detection of second order changes as a basis of segmenting actions, and for building schemas for recognizing actions. Here, first order changes refer to changes in state like change in position. But tracking first order changes does not result in correct point-to-point action definition points. Second order changes refer to changes in changes in state, like changes in change of position. He found that the use of second-order changes resulted in more stable schemas than those based on first-order changes. The final goal of his system was to generate causal or intentional descriptions of actions.

In [7], Badler generated conceptual descriptions, of changes in a synthesized 2D visual scene, from object and event representations. Each object was associated with a number of properties - type, visibility, mobility, location, orientation and size. The event representation was hierarchical and proceeded from the lowest level of describing an object's motion to the highest level of recognizing specific motion verbs. Each event node was also associated with a number of properties - subject, agent, instrument, reference, direction, trajectory, velocity, start and end times, etc. Changes in object locations triggered separate processes called demons to compute trajectories and velocities of the objects. Then motion demons observed the motions at every frame, recognizing an event only when a change in state or motion was detected. Lower level demons placed data in event nodes which were analyzed by higher level demons. Intermediate level event demons included directional adverbials which were able to recognize the direction of motion in terms of across, along, clockwise, onto, on, etc. At the higher levels, the demons could recognize repetitive actions and finally generate motion verbs. The temporal relations between events were recognized using linguistic concepts such as before, during, while, until, etc. Whenever a new event was observed, the current event node was terminated and a new node was spawned which inherited some of the properties of the old one. The two event nodes were then connected together by the "next" property. Event nodes were terminated when there were significant changes in properties like change in velocity from zero to positive value or from positive to zero, change in subject, change in contact relation between agent and subject, change in direction, etc.

4.3.2 Event-based Action Segmentation

In this section, we explain our event-based action segmentation approach. The results obtained by [7, 49, 60] (Section 4.3.1) show that appropriate action-definition points have to be determined for the correct segmentation of actions. Also, these action definition points have to be computed based on second or higher order changes in state. In [7], Badler used higher order changes to abstract higher levels of action definitions and parameters. As we have shown in Section 3.2.1, the zero-crossings of the second derivative of the motion data indicate significant changes in the motion. These zero-crossings correspond to third order changes in state and are good candidates for action definition points. At the zero-crossing points, the velocity has its local extrema and the acceleration crosses the zero-value as it goes from positive to negative or vice versa. Perceptually, these points correspond to changes in motion such as starting from rest, coming to a stop, or changing the velocity direction which are significant events by themselves.

In this thesis, we mainly consider constrained actions involving interactions of the agent with other objects or with self. In our effort to define an event basis for action segmentation, we continue to use the concepts of tracking sensors and tag objects as defined in sections 3.2.2 and 3.2.3 respectively. The CaPAR system computes the zero-crossings of acceleration data of only the tracking sensors. We conducted several experiments to study the relation between zero-crossings in acceleration space and the proximity of the tracking sensor from the objects.

Fig. 3.2 shows the trajectory of the hand tracking sensor for the example *touch the table* and Fig. 3.3 shows the corresponding plots of the distance between the tracking sensor (right hand of the agent) and a tag object (table) and the zero-crossings of the accelerations. It can be clearly seen that an important event has occured when the instant of zero-crossings coincides with the close proximity of the tracking sensor and a tag object. This event corresponds to an occurrence of a spatial constraint (Section 3.2). Similarly, during visual attention tracking (Section 3.5), the event corresponds to the occurrence of a visual constraint.

Using the above results, the CaPAR system uses the spatial or visual constraint event as the event-basis for all action segmentation. We will refer to the frame at which this event occurs as the *event-frame.* The system uses the event-frames to segment the given action into a number of sub-actions. We can clearly see that the sub-action preceding the event frame is very different from the sub-action after the event-frame. The event-frame is a discriminator between the two actions. We will show that the motion information at the event frame defines the previous sub-action. Thus, by Newston's definition, the event-frame is an action-definition point. As an example, we shall consider the action *touch an object* in which the agent starts from a rest position, reaches across and touches the object. This action has two motion segments. At the event-frame, the agent just comes in contact with the object. Before the event-frame, the sub-action corresponds to the agent reaching for the object. After the event-frame, the sub-action corresponds to the agent continuing to touch the object.

For a given action, one or more tracking sensors may be active at the same time i.e., the action may involve whole body motion. For example, *touch with left hand* and *touch with right hand* should be abstracted to a single action of *two-handed touch.* In such cases, it becomes very difficult to build conceptual representations for the whole action. To alleviate this problem, the CaPAR system considers one tracking sensor at a time. Each tracking sensor acts as the end-effector of a kinematic link in the agent's body. We perceive the observed action to be a complex combination of motions. Each motion belongs to a separate kinematic link in the virtual model's body. The system computes the event-frames for each motion and segments it accordingly. So, the problem of action segmentation is now reduced to the problem of motion segmentation. For each motion segment in each kinematic link, the CaPAR system separately analyzes the segment's relation with the environment and builds corresponding conceptual representations. Later, as described in Section 4.6.2, the system combines the conceptual representations of actions resulting from each kinematic link and generates one complex conceptual representation for the entire action.

In all our examples, the agent interacts with one or more objects in the environment. The agent may also interact with itself, *i.e.,* interact with another part of the agent's body (for example: *touch head with hand, clap both hands together, etc.*). In such cases, we consider the interacting body part (eg: head in *touch head with hand* and one of the hands in *clap both hands together*) to be another object in the environment. In general, in the presence of multiple objects in an action, we assume that only one of the objects actively moves around during the action, and is directly touched, manipulated, and moved by the end-effector. We refer to this object as the *dominant object.* The rest of the objects (if any) are referred to as *subordinate objects* and are assumed to be passive during the action. The subordinate objects may come in contact only with the end-effector or with the dominant object. Also, the contact between the dominant object and a subordinate object is significant only after the possession of the dominant object by the end-effector. At all other times, the contact between the dominant object and the subordinate object is insignificant and ignored. Finally, we assume that the end-effector or dominant object may be in contact with only one other object at a time *i.e.,* the end-effector can be in contact with a dominant object or a subordinate object but not both. Similarly, the dominant object can be in contact with the end-effector and only one other subordinate object at any given time. In general, we only consider actions that can be applied iteratively to a set of concrete objects.

Our motion segmentation techniques should be able to address very complex situations. For example, consider an observed action that involves touching multiple objects (labeled as $obj_1, obj_2, \dots obj_n$) in the environment with and without an instrument. For example, using the right hand, the agent may first touch obj_1, then obj_2 and obj_3 at consecutive zero-crossing frames and then continue to touch obj_3 for some continuous period of time, then again touch obj_1, grasp it as an instrument, and with it touch obj_2 for some continuous period of time, then touch obj_3 and obj_4 at consecutive zero-crossing frames, release obj_1 as an instrument and then directly touch obj_2. The CaPAR system needs to be able to distinguish and recognize each of the transitions between touching the various objects.

The process of checking for proximities between the end-effector and a tag object at zero-crossing frames helps us to detect the presence or absence of constraints. But that alone will not help us to recognize redundant constraints or constraints with different objects at consecutive frames. Due to the inherent problems associated with the process of motion capture, the data obtained may not be very clean and may result in redundant zero-crossings of acceleration. Hence, even after the end-effector and an object are in contact, the system may recognize multiple occurrences of constraints between the same two objects. It needs to ignore the redundant ones. Also, the end-effector may come in contact with different objects at consecutive zero-crossing frames. The CaPAR system needs to take these into account and distinguish them from the cases of contact with the same object at consecutive zero-crossing frames. To simplify this process, for each monitoring sensor or kinematic link, the system generates two binary arrays and two integer arrays, all of the same size. Each element in an array corresponds to a zero-crossing frame. For this, the system considers all zero-crossing frames, even those that are not associated with spatial proximities. One of the binary arrays (referred to as *eeObjProx*) indicates proximity between the end-effector and one of the objects at each of the zero-crossing frames. '1' indicates a proximity and '0' indicates no proximity. Similarly, the second binary array (referred to as *objObjProx*) indicates proximity between the dominant object and any other subordinate object. The first integer array (referred to as *eeObjTagIndex*) contains the object number that the end-effector came in contact with at the corresponding zero-crossing frame. Similarly, the second integer array (referred to as *objObjTagIndex*) contains the number of the subordinate object that the dominant object came in contact with at the corresponding zero-crossing frame. For the above example, the four arrays generated would be:

$$\begin{aligned}
\text{bool eeObjProx[]} &= (0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0);\\
\text{int eeObjTagIndex[]} &= (0,0,0,1,1,1,2,3,3,3,1,1,1,1,1,1,1,1,1,1,2,2,0,0);\\
\text{bool objObjProx[]} &= (0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0);\\
\text{int objObjTagIndex[]} &= (0,0,0,0,0,0,0,0,0,0,0,2,2,2,3,4,4,4,0,0,0,0,0,0);
\end{aligned}$$

The detection of each new constraint generates a new motion segment and the resulting motion segmentation would be as shown in Fig. 4.6.

As shown by Newston [49], the key features of an action are detected during a transition from one state to another. Hence, the CaPAR system extracts most of the information about the sub-actions at the event-frames. Only some of the style parameters are extracted during the active phase of the motion. In Section 4.4, we describe the extraction and analysis of motion parameters for each motion segment.

Segment #	Start Frame#	End Frame#	Segment #	Start Frame#	End Frame#
1	0	3	8	13	14
2	3	6	9	14	15
3	6	7	10	15	17
4	7	9	11	17	19
5	9	10	12	19	20
6	10	11	13	20	21
7	11	13	14	21	23

Figure 4.6: Action segmentation from the binary arrays

4.4 Semantic Features Extraction

The two main purposes for extracting the semantic features from a motion segment are *PAR characterization* and *PAR generation*. For PAR characterization, we identify the features that characterize the action being observed and uniquely identify a UPAR. For PAR generation, we extract and store the features of the action in the identified UPAR and the associated agent and object models such that the extracted information can be reused later for generating similar actions for different sized agents in any simulation or environment. Whereas all the extracted features may be needed for generating similar constrained motions for a secondary agent, only some of the features are needed to characterize the PAR. For example, for PAR characterization, we are neither interested in the style with which the primary agent executed the action nor are we interested in the exact constraint location. But these two features are important for motion generation. In the *Reach* action, for motion generation, we are interested in knowing where and how the agent reached. But for PAR characterization, we are only interested in the features that indicate the agent reached for something.

All the semantic features extracted from a motion cannot be stored within the UPAR structure. Some of the features (*e.g.*, style) are specific to an agent and some (*e.g.*, constraint location) are specific to the object type. Hence, as shown in Fig 4.7, the extracted features are stored appropriately in the corresponding UPAR, or the agent model or the object model which are also part of the Actionary [8] database. In this section, we discuss each semantic feature and suggest extensions to the UPAR, agent and object models

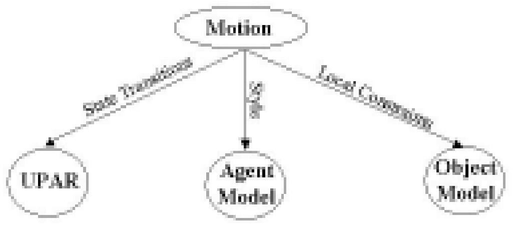


Figure 4.7: Storage of features extracted from a motion

to facilitate the storage and retrieval of the parameters.

4.4.1 Features for PAR Characterization

Here, we consider the semantic features and predicates used for characterizing a PAR. We consider the complex action *drink from a mug* and its sub-actions *Reach* and *MoveObjTo* to illustrate the different features extracted from a motion segment. In the *drink from a mug* action, an agent, who is initially in a neutral standing posture and not in contact with any object in the environment, reaches across to pick up the mug, drinks from it, puts the mug back on the table, and goes back to a posture similar to the initial one. Here, the mug is the dominant object, whereas the table and the mouth are the subordinate objects.

Type of EndEffector We assume that each primitive PAR has a single active end-effector, which can be one of right hand, left hand, right foot, left foot, head, elbow, etc. For the PAR characterization process, we are only interested in knowing the type of end-effector and not the side used or the handedness of the agent. For example, we would like to know if it is the hand or the leg but not if it is a right hand or left hand. A *Reach* is a reach whether it is done with the left hand or the right hand. Hence we use the general type of end-effector (hand, foot, etc) for PAR characterization and the specific type of end-effector (right hand, left foot, etc) during the PAR generation process. During motion regeneration process, we specify the exact side of end-effector to be used. If it is the same as the observed side, then the data can be directly used. But, if it is opposite to the observed side, then as will be explained in Section 4.4.2.2, we can easily compute the new mirrored constraint locations. In the *drink from a mug* action and all its sub-actions, the CaPAR system determines the end-effector for PAR characterization to be the *hand* and the end-effector for PAR generation to be the *right hand*.

EE Object contact relation Here, the CaPAR system extracts the contact relations between the end effector and the different objects both at the beginning and at the end of the motion. The contact relation can be either 0 (no contact) or 1 (contact). As described in section 4.3.2, we are only concerned with the contacts between an end-effector and any of the objects and between the dominant object and any subordinate object. We represent the contact relations at the start and end of the motions by two separate binary strings (*EEObjRelStart* and *EEObjRelEnd*). If we have n objects in the action, then the length of each string is $2n-1$. The first n elements in the string represent the contact relation between the end-effector and each of the n objects. The next $n-1$ elements in the string represent the contact relation between the dominant object and each of the other $n-1$ subordinate objects. Both these strings are directly generated by the CaPAR system from the information in the 4 arrays described in Section 4.3.2. This feature is required for both PAR characterization and PAR generation processes.

In the motion segment corresponding to the *Reach* action, the number of objects is 1. Hence the string length is also 1. In the frame corresponding to the start of the motion segment for *Reach*, no contact is detected by the system between the right hand and the dominant object, the mug. Hence the *EEObjRelStart* feature generated for this motion segment is '0'. In the frame corresponding to the end of the motion segment for *Reach*, there is a contact between the right hand and the dominant object, the mug. Hence the *EEObjRelEnd* feature generated by the system for this motion segment is '1'. In the *drink from a mug* complex action, there are two instances of *MoveObjTo* action. The first one corresponds to the motion segment in which the agent moves the mug, he is holding, to the mouth. The second instance of *MoveObjTo* action corresponds to the motion segment in which the agent moves the mug from the mouth to the table. The number of objects detected for this motion segment is 2 and therefore, the length of the corresponding strings for the *EEObjRelStart* and *EEObjRelEnd* features is 3. In the frame corresponding to the start of the motion segment for *MoveObjTo*, a contact is detected by the system between the right hand and the dominant object, the mug. Hence the first bit in the binary string is set by the system to '1'. But no contact is detected between the right hand and the subordinate object (the mouth in the first instance and the table in the second instance) and also between the dominant object and the subordinate object i.e. between the mug and the mouth (in the first instance) or between the mug and the table (in the second instance). Hence each of the next two bits in the binary string representing the feature is set to '0'. Combining these bit representations of the contact relations, the CaPAR system automatically generates '100' as the *EEObjRelStart* feature for the motion segment corresponding to the *MoveObjTo* action. Similarly, at the frame corresponding to the end of the motion segment, a contact continues to be detected between the right hand and the dominant object, the mug. Hence the first bit in the binary string is set to '1' by the system. But no contact is detected between the right hand and the subordinate object (the mouth in the first instance and the table in the second instance). So the second bit in the binary representation of the *EEObjRelEnd* feature is set to '0'. Now, a new contact is detected between the mug and the mouth (in the first instance) or between the mug and the table (in the second instance). Hence the third bit in the binary string is set to '1'. Combining these bit representations of the contact relations, the CaPAR system automatically generates '101' as the *EEObjRelEnd* feature for this motion segment.

Change in Object Location Here, the system checks each of the n objects for any motion during the action. This is done by checking the relative transformation between the locations of the object at the start and the locations at each frame. If the system detects any change in location, the object is considered to have moved during the motion segment. The motion status of all n objects is represented by a binary string of length n. If the object has moved, then the system sets the corresponding element in the string to '1'. Otherwise, the element in the string is set to '0' to indicate that the object was stationary during the action. This feature is required for both PAR characterization and PAR generation processes.

In the motion segment corresponding to the *Reach* sub-action of the *drink from a mug* complex action, the dominant object, the mug, is not found to move during the action. Hence the *ObjChangeLoc* feature generated for this motion segment is '0'. In the motion segment corresponding to the *MoveOBjTo* action, only the mug is found to have moved but not the table or the mouth. Hence the *ObjChangeLoc* feature generated for the *MoveObjTo* action by the system is '10'.

Number of objects As mentioned in Section 4.1.2, our system generates UPARs in which the agents and objects are unknown. But, a UPAR has a fixed number of objects. During our PAR generation process, we continue to use the concepts of user specified tracking sensors and tag objects. The number of objects in the UPAR generated for the observed complex action corresponds to the number of tag objects defined by the user. But, the number of objects in the primitive PARs generated for each of the sub-actions may not be the same. For example, *Reach* is a sub-action of most contact-based actions. Simple *Reach* needs only one object - the object to reach for. But each complex action containing *Reach* as a sub-action may have a different number of objects. We want to be able to correctly recognize the *Reach* sub-action of any complex action. To compute the correct number of objects in a sub-action, the CaPAR system first generates the two strings *EEObjRelStart* and *EEObjRelEnd* by considering all the N objects where N is the total number of objects in the observed complex action. Hence the string lengths of these *EEObjRelStart* and *EEObjRelEnd* are $2N-1$. From these strings, the system checks for any contact relations between the end-effector and any of the objects. If there is a contact either at the beginning or at the end, the system adds that object to the list of objects for the sub-action. Next, if there is a contact between the end-effector and the dominant object, the system checks the *EEObjRelStart* and *EEObjRelEnd* strings for contacts between the dominant object and each of the other subordinate objects. If there is a contact with a subordinate object at the end of the action, the system adds that object (if not previously added) to the list of objects for the sub-action. By this process, the system automatically determines the number of objects in the sub-action to be n, where $n \leq N$. This feature is explicitly used for PAR generation and implicitly used for PAR characterization. In the *drink from a mug* example, the CaPAR system correctly determines the number of objects in the motion segment corresponding to the *Reach* action to be 1 and the number of objects in the motion segment corresponding to the *MoveObjTo* action to be 2.

Force Applied This feature specifies if any force is applied to the object. In the beginning of the motion abstraction process, the user interactively specifies whether we need to consider force transfers for the entire complex action. But this does not imply that all the sub-actions involve force transfers. Based on the user's specification and the *EEObjRelStart* and *EEObjRelEnd* features derived for that motion segment, the CaPAR system automatically generates the binary string representing the force transfer for each object. For an object, indexed by i, if *EEObjRelStart*[i] is '0' and *EEObjRelEnd*[i] is '1' and if the user had specified that a force transfer was involved in the complex action, then the CaPAR system sets the ith bit for that object in the binary string to be '1'. Otherwise, it is set to a '0'. For example, in the motion segment corresponding to a *Reach* action, the force applied feature generated by the CaPAR system is a '0'. But for the *Hit* action, the binary string generated is a '1'.

The applied force causes a propelled motion in an object. The force may be applied by an end-effector on a dominant object or by the dominant object on a subordinate object. In the latter case, the dominant object refers to the instrument used in the action. We use Miller's [44] definition of *propels X* to imply *applies force to move X*. So, we refer to the motion of an object caused by the application of force on it as a propelled motion. For example, *hit, kick* are propelled motions, whereas *move, pick-up* actions are simple causal motions. During the motion regeneration process, the system computes new trajectories, due to force interactions, only for objects with propelled motions. This feature is used for both PAR characterization and PAR generation.

4.4.2 Features for PAR Generation

In this section, we discuss the features which are used only for the PAR generation process. These include the locations of the spatial and visual constraints and the style of performing the various actions by the individual agents.

4.4.2.1 Objects in Contact

In this thesis, we are interested only in actions involving interactions between an agent with other objects or with self. Each motion segment, detected by the CaPAR system, is associated with a spatial constraint which involves contact between two objects, where one of the objects involved in the constraint may be one of the end-effectors of the agent. Also, as described in Section 4.4.1, each motion segment may be associated with one or more objects. If there is only one object and there is a new contact relationship to be established at the end of the action, then it can be safely assumed that the contact is going to occur between the end-effector defined for the action, and the object. But, if there is more then one object identified for the motion segment, then the new contact may be between the end-effector and one of the objects, or if the end-effector is already in contact with the dominant object at the start of the motion, the new contact may be between the dominant object and one of the subordinate objects. Hence, it is essential that the system correctly detects the two objects that will be involved in the new constraint or contact relationship at the end of the action. During the initial process of checking for spatial constraints in the observed action, the CaPAR system determines the objects that are in contact and identifies them by their indices - the end-effector is identified by -1 and the objects are identified by their index number in the object list of the motion segment. The index numbers of the contacting objects are then stored by the system in the *CollidingObjects* feature. This information will then be used by the system during the motion regeneration process to know the specific objects that will be involved in the constraint.

4.4.2.2 Constraint Location

In our technique, each motion segment of a given action is separately analyzed and conceptualized. As mentioned earlier, the occurrence of a spatial constraint signals the end of one motion and the beginning of another. Any secondary agent should also maintain the same or similar spatial constraint while imitating the action of the primary agent. In Section 3.3, we described in detail the method used to compute new constraint locations for a given secondary agent. In that case of direct mapping, we had apriori knowledge of the secondary agent and its size. But, now, the secondary agent is unknown at the time of PAR generation. Therefore, we need to store enough information in the PAR about the constraint to enable the correct constraint location to be computed for any given secondary agent. The constraint is established between the end-effector(agent) and the object or between two objects. The generated UPAR for the action should be applicable to any object and any agent. It is not feasible to store the constraint location within the UPAR construct. As described in Section 3.3, the location of the constraint for the secondary agent depends on the type of the tag object. In other words, the constraint location is a property of the object.

For correct reproduction of the action, we need to store both the position and the orientation of the constraint. The correct constraint position is given by the collision detection algorithm [34] when it detects a collision between the end-effector segment and an object or between two objects. As mentioned in Section 4.3.2, during the processes of motion segmentation and PAR generation of a primitive action, the CaPAR system processes each kinematic link separately. During the motion regeneration process, the system uses a specific end-effector site for each kinematic link. For example, for the kinematic link consisting of the right arm and the right hand and associated with the monitoring sensor on the right hand, it uses the agent's *right_palm.palmcenter* as the end-effector site. During the process of extracting features for PAR generation, the system uses the global orientation of this end-effector site as the global orientation of the constraint.

The constraint location is specific to the observed object type. The instances of this object type may be of different sizes. To be applicable to all instances of the object class, the CaPAR system stores the position and orientation of the constraint local to the coordinate system of the object. Further, it normalizes the position of the constraint by the size of the object. We refer to these constraint locations as Normalized Local Transforms (NLT). During the process of regenerating motions from the PAR for a secondary agent interacting with a similar object, the stored constraint values are used to compute the correct global constraint location based on the size and positions of the agent and the object in the new environment. Further, to increase the efficiency of storing the constraints, the system stores the constraint locations as a vector of length 7. The first 3 elements of the vector contain the position of the constraint and the next 4 elements of the vector contain the orientation of the constraint in the form of a quaternion [58].

The CaPAR system tracks both spatial and visual constraints. During the abstraction and direct mapping process, the visual constraints were tracked separately. As explained in Section 3.5, a visual constraint was recognized when a zero crossing of acceleration of the tracking sensor on the head occured together with the intersection of the line of sight of the agent with a tag object. But, this becomes a difficult problem during the PAR generation and action segmentation process. This is mainly due to synchronization issues. To overcome this, the system checks for visual constraints whenever a spatial constraint is detected. The visual constraint location is then stored together with the spatial constraint location. Similar to the spatial constraints, the visual constraint locations are also stored as NLTs of the object.

As described in Section 4.3.2, a constraint can occur between an end-effector and an object or between the dominant object and a subordinate object. In the former case, as we assume a specific end-effector site, the system needs to only store the constraint location computed on the object as the goal NLT of the constraint in the corresponding object model. But in the latter case, when the dominant object possessed by the end-effector comes in contact with a subordinate object, we need to know the relative constraint between the dominant object and the subordinate object *i.e.,* we need to know the specific location on the dominant object that comes in contact with a specific location on the subordinate object. For example, in the *drink from mug* action, when the mug held by the agent's hand comes in contact with the agent's mouth during the actual process of drinking, we need to know the specific location on the mug that comes in contact with the specific location of the agent's mouth. To be able to do this, the system computes the constraint locations on both the dominant object and on the subordinate object and computes their corresponding NLTs. The dominant object's NLT is stored as the *eeNLT* in its object model and the subordinate object's NLT is stored as the *goalNLT* in its object model. We refer to the NLT of the visual constraint as the *vNLT* and store it in the corresponding object model.

It is possible that the same object or object type may be used for multiple actions. But, for each action, the observed constraint locations may be very different. So, while storing the various NLTs as properties of the object, it is necessary to qualify the constraint location by the PAR name. In our PAR generation by action segmentation method, a sub-action associated with the same agent and same object may be common to multiple complex actions. For example, the *Reach* action is common to *touch the box, hit the box, pickup the box* actions. For all these actions, the agent and object may be the same. But the constraint locations will be different as the intent of or purpose for reaching the box is different in each case. For the *touch the box* action, the agent may just touch a surface of the box. So, the purpose of *Reach* is *touch.* For *pickup the box* action, the agent will grasp the box at a different location to get a proper grip. Here, the purpose of *Reach* is *pickup.* To take care of this problem, we associate the constraint with a user-specified purpose. Finally, when an agent is coming in contact with an object, especially while using the arms or legs to do so, either the left limb or the right limb could be used. The goal constraint location on the object will vary based on the side (right/left) of the limb used. We desire to store both types of constraint locations. Hence we also associate the constraint with a side. Later, during the motion regeneration process, if we need to know the constraint location for the same UPAR with the same purpose, but on the opposite side, it would be trivial to compute it from the stored information and the geometry of the object.

With all the above considerations, we have extended the structure of the object model in the Actionary (Fig. 4.3) to include the constraint locations as shown in Fig 4.8. Further, to allow easy access for insertion and retrieval, the constraint location structure in the object model is implemented as a tree as shown in Fig 4.9.

4.4.2.3 Style

Each person has an individual style for doing an action. It is sometimes desirable to replicate the style of another person. Here, the system extracts from an action the different style parameters (described in Section 3.4). The style is specific to an agent and so should be stored in the agent model. Similar to storing constraint locations, the style information is associated with the PAR name and the user-specified purpose. Additionally, we qualify

```
type object representation =
        (coordinate-system: site;
         state: state space;
         rel-dir: sequence relative direction;
         special-dir: sequence special direction;
         grasp-sites: sequence site;
         actions: sequence parameterized action;
         constraints: sequence constraint location)

type constraint location =
        (action: parameterized action;
         purpose: string;
         side: direction;
         locations: vector transform;
        )

type vector transform =
        (eeNLT : real vector;
         goalNLT: real vector;
         vNLT: real vector;
        )
```

Figure 4.8: Modified object representation type to include spatial constraints

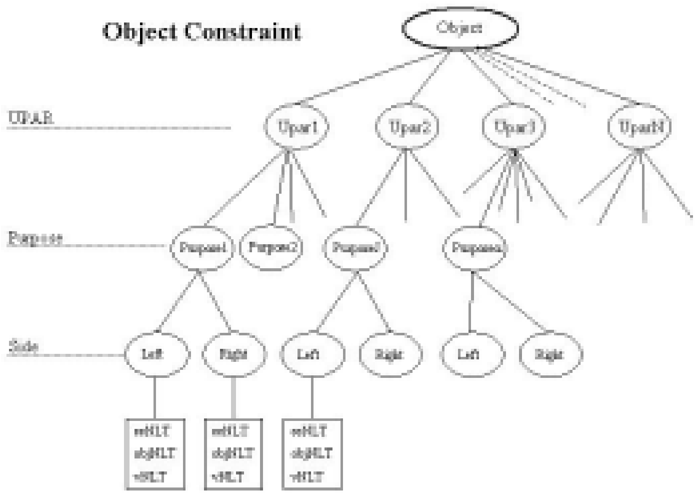


Figure 4.9: Constraint locations stored within a tree in the object model

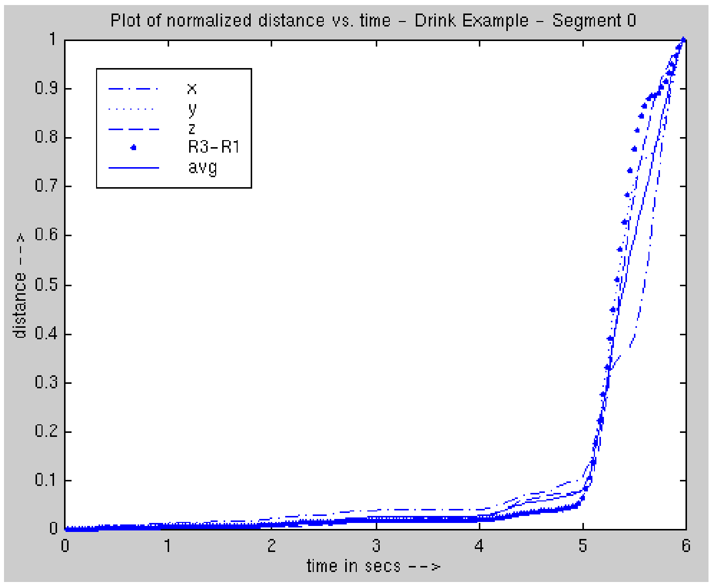


Figure 4.10: Normalized distance moved by the end-effector

the style by the specific end-effector too. For example, the style of the reach action done by the same person for the same purpose may not be the same for both the left and the right arms. Here, we will consider each style separately:

Frame-wise Variations in End-Effector Velocity (*VelStyle*): Using this style information, the secondary agent will be able to maintain the same speed in the action along any trajectory. Previously, during the motion abstraction and direct mapping process (explained in Section3.4.1), we computed the normalized angular distance moved in the joint space by the primary agent at each frame. During the PAR generation process, we need to store this style information for future use. But storing the information at each frame for each degree of freedom of each joint amounts to a large amount of data and is also redundant. As mentioned earlier, each primitive UPAR is associated only with a single end effector. To increase the efficiency of storing the style data, the CaPAR system computes the normalized distance, $\hat{s}_i$, moved along the trajectory at each frame along each of the 3 coordinate axis of the end-effector.

$$\hat{s}_i = \frac{\int_0^t |v(\tau)| d\tau}{\int_0^{t_{end}} |v(\tau)| d\tau} \tag{4.1}$$

where t_{end} is the duration of the motion segment. The system then computes and stores the average of the three values at each frame. Hence for each motion segment, it only stores a one dimensional vector of normalized distances. Fig 4.10 shows the variation of the normalized distances along the different axes and compares it with their average value. During the motion regeneration process, these distances are used as interpolating factors for the end-effector interpolation.

Variations in Approach Path (*PathStyle*): We use this style when we need the trajectory of the secondary agent's end-effector to have the same shape as the trajectory of the primary agent's end-effector. For this, we need to store the intermediate significant end-effector locations. The process of extracting this information has been explained in detail in Section 3.4.2. If during the motion regeneration process, the duration of the action is different from the observed duration, then the key frame numbers corresponding to the significant end-effector positions have to be computed correctly. To allow this, the system also stores the instant of time, normalized by the duration of motion, corresponding to the frame number at which the significant end-effector location has been detected by the system.

```
type agent style =
        (action: parameterized action;
         purpose: string;
         EE: int;
         styles: style information;
        )
type style information =
        (style1 : sequence vel style;
         style2 : sequence path style;
        )
type vel style =
        (time: real value;
         interpolation factor: real value)
type path style =
        (time: real value;
         significant-points: real vector)
```

Figure 4.11: Style information added to the agent model

With all the above considerations, we have extended the structure of the agent model

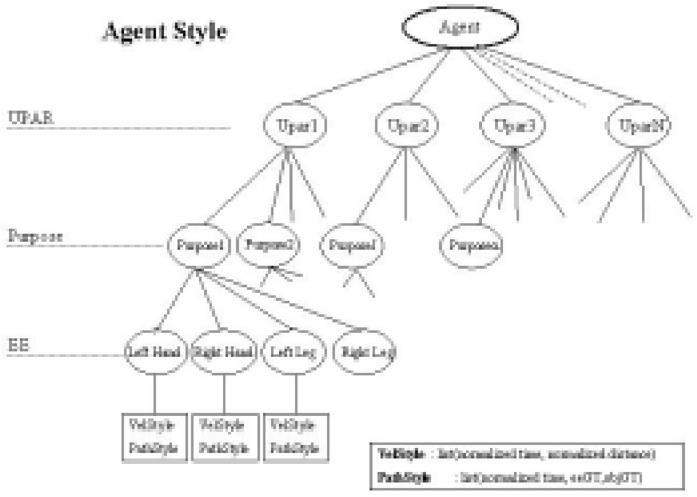


Figure 4.12: Style information stored as a tree in the agent model

of the Actionary shown in Fig 4.2 to include the style information as shown in Fig 4.11. Again to increase efficiency, we store the global end-effector locations as vectors. Further, to allow easy access for insertion and retrieval, the agent style structure in the agent model is implemented as a tree as shown in Fig 4.12.

Variations in Applied Force: We use this style when we need to map force interactions between the agent's end-effector and an object. This is an extension of the *PathSTyle* and has been explained in detail in Section 3.4.3. The trajectory of the secondary agent's end-effector is generated similarly by the system, but we need additional information like the computed coefficient of restitution and damping forces to compute the new trajectory of the object. These factors are specific to the object type and so the system stores them in the object model. We again extend the state space property of the object data type to include this as shown in Fig 4.13

4.5 PAR Characterization

In this section, we discuss our technique of characterizing a PAR from the extracted semantic features. We also explain our method for efficiently recognizing a previously identified PAR from the given set of features. The feature table is initially empty.

```
type state space =
        (position: real vector;
         velocity: real vector;
         acceleration: real vector;
         force: real vector;
         torque: real vector;
         damping force: real vector;
         coefficient of restitution: real value).
```

Figure 4.13: Force components to be added to the object model

Whenever a new action is observed, the CaPAR system extracts the *EEObjRelStart, EEObjRelEnd, ObjChangeLoc, EE,* and *Force* semantic features, as described in Section 4.4.1, and groups them together to form a feature set. Each feature set is then compared against existing feature sets in the feature table. As all the features are stored as binary strings, the comparison of individual features reduces to a simple string comparison problem. If there is a match, then the motion segment is labeled with the matched UPAR name. Otherwise, the system prompts the user for a new UPAR name and stores it along with the feature set in the feature table.

If we use a flat feature table, then as the number of feature sets stored in the table increases, the computation time required to search for a match also increases. To minimize the search time, we use the concepts of pattern classification and store the feature table as a decision tree (Fig 4.14). Each feature in the feature set is stored at a different level in the tree. The features are sorted by priority. The higher priority features are stored in the higher levels of the tree. The *EEObjRelStart* and *EEObjRelStart* features are very important in characterizing a PAR and are hence stored at the top 2 levels. This is followed by the next lower priority feature, *ObjChangeLoc*, which is stored one level lower in the tree. The number of objects is implicitly included in each of these 3 features as the length of the strings is either $2n-1$ (for EEObjRelStart) or n (for *ObjChangeLoc*) where n is the number of objects (section 4.4.1). Hence, though the number of objects is a very important PAR characterizing feature, it need not be stored explicitly in the feature table tree.

This tree structure makes it very easy and efficient to search for a matching feature and for inserting a new feature in the tree. The search process proceeds from the top of

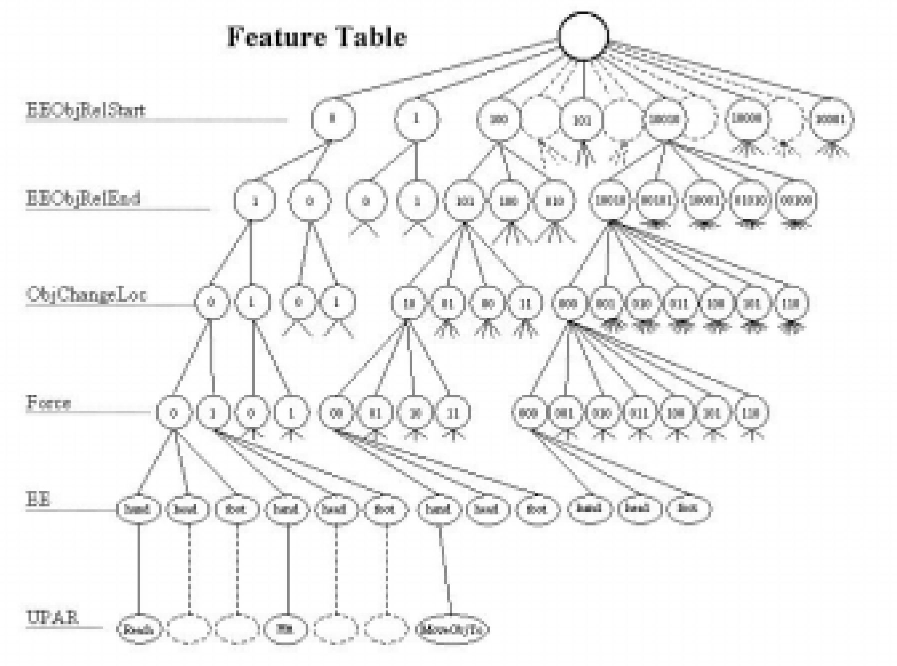
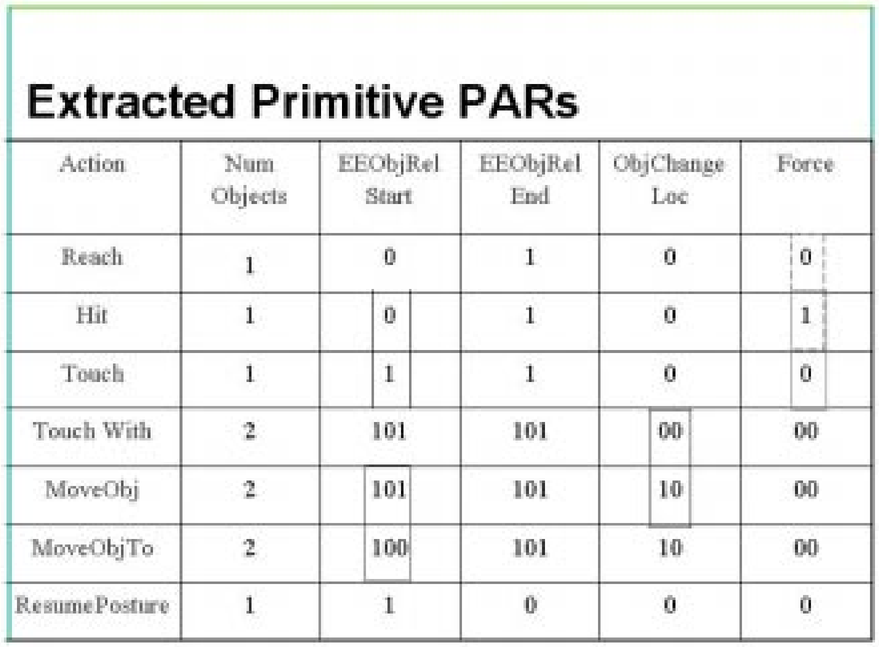


Figure 4.14: Feature table

the tree to the bottom. At any level, a comparison is made by the system between the feature stored at that level and the corresponding feature in the new feature set. If there is a match at a node, then only that node's children are traversed to continue comparisons with the other features in the feature set. If there is a match at every level, then the system returns the unique PAR that matches the feature set completely. But, if there is no match at a level, then the system returns the list of all possible UPARs that have matched the feature values until that level and prompts the user for a unique UPAR name. While inserting a new feature set into the tree, a similar search process is first done. If there is no match at one of the tree levels, then a subtree is created at that level and the rest of the features are inserted into that subtree.

Using the above technique, the CaPAR system can easily find matches for various primitive actions as shown in Fig 4.15. Later, if we need to add more specific properties like direction of motion, we can easily add them to the feature table. The tree structure allows for easy extensions and additions. Similar to Miller's verb tree [44], the specific UPARs contain all the semantic features of the general UPARs, plus others.



Extracted Primitive PARs

Action	Num Objects	EEObjRel Start	EEObjRel End	ObjChange Loc	Force
Reach	1	0	1	0	0
Hit	1	0	1	0	1
Touch	1	1	1	0	0
Touch With	2	101	101	00	00
MoveObj	2	101	101	10	00
MoveObjTo	2	100	101	10	00
ResumePosture	1	1	0	0	0

Figure 4.15: Extracted primitive PARs

4.6 PAR Generation

In this section, we describe the technique for generating both a primitive UPAR and a complex UPAR for the observed action. For this, we use all the features extracted for both PAR characterization and PAR generation.

4.6.1 Creating Primitive PAR

For each derived motion segment (Section 4.3), the CaPAR system identifies a UPAR (Section 4.5). For each newly identified UPAR, the system uses the extracted features to derive and set all the default parameters. A PAR (Section 4.1) has both high-level and low-level parameters. The high level parameters include the applicability conditions, termination conditions, preparatory specifications and execution steps. The specifications for these parameters are all stored separately as script files in Python [52]. The CaPAR system derives the default parametric values for all these conditions and generates the corresponding Python scripts. The low level parameters extracted include the constraints and the duration of the action. In this section, we will discus each parameter separately.

4.6.1.1 Preparatory Specifications

As mentioned in Section 4.1.1, the preparatory specification is a list of <CONDITION, action> statements. We refer to the CONDITION in the specification as the precondition. As we are considering only constrained actions (contact verbs), the preconditions can be easily derived from the *EEObjRelStart* feature which specifies the contact relation between the end-effector and the other objects and between the dominant object and other subordinate objects at the start of the motion segment. So, the associated actions to satisfy these preconditions will always be one of *Reach* which causes the end-effector to come in contact with a specified object or *MoveObjTo* which causes the dominant object held by the end-effector to come in contact with the specified subordinate object. These actions will be discussed in detail in Section 5.2. If there is no contact relation specified by the *EEObjRelStart*, then the CaPAR system uses the posture of the agent at the start of the motion segment as a precondition. The associated action will then be a *PostureChange* UPAR. For all our complex actions, the agent was in an initial neutral standing posture.

Consider the *MoveObjTo* action which specifies 2 objects. The action causes the first object (dominant object) to move such that it is in contact with the second object at the end of the action. The EEObjRelStart for this action (Section 4.4.1) is 100 which specifies that at the start of the motion segment, the end-effector is in contact with the dominant object but the dominant object is not in contact with the second object. This should translate to the following when written in Python.

Preparatory Specification in Python for the action *MoveObjTo*:

```
def preparatory_spec(self, agent, obj1, obj2):
    if(contact(EE,obj1)):
         return 1
    else
         actions = {'PRIMITIVE':("Reach",{'agents':agent,
                                          'objects':(obj1)})};
         return actions
```

Unfortunately, the PAR system does not allow us to currently check for the end-effector values or any other instantiated PAR values except for the agent and object models within the Python scripts. This implies that we cannot currently check for the contact relation between the end-effector and any other object within the preparatory specifications. We would be able to do this with changes to the internal PAR architecture. Until then, the CaPAR system uses *EEObjRelStart* and the necessary posture information, and overcomes this problem by generating and checking for the pre-conditions at run-time during the process of motion regeneration. The associated actions (*Reach, MoveObjTo*), instantiated with the necessary parameters, are also generated at run-time. This will be explained in detail in Section 5.3.2.

4.6.1.2 Termination Conditions

For the termination conditions, we use *EEObjRelEnd* feature. For example, for *Reach* action, the *EEObjRelEnd* is 1. Correspondingly, the termination condition should check for the contact between the end-effector and the object. Again, as explained above, we cannot currently check for this within the Python scripts. Hence, the CaPAR system generates and checks for the termination conditions also at run-time during the process of motion regeneration. Additional features and user input may be needed to completely specify the termination conditions. For example, the additional feature of *duration* is used to specify the termination of a *touch* action, where there is no difference between the starting and ending contact relations in the action.

4.6.1.3 Execution Steps

The execution steps for a primitive action are very straightforward. The CaPAR system needs to correctly specify the objects. The execution steps described in Python for the *Reach* example are shown below:

```
def execution_steps(self, agent,obj1):
       actions = {'PRIMITIVE':("Reach", {'agents':agent,
                                         'objects':(obj1)})}
       return actions;
```

4.6.1.4 Applicability Conditions

For the applicability conditions, we do not use any extracted features. The system only generates the standard default applicability conditions as shown below:

```
def applicability_condition(self, agent,obj1):
        if checkCapability(agent, self.name, [agent]):
                return 1;
        else:
                return 0;
```

4.6.1.5 Constraints

In this thesis, we address only constrained actions. Hence it is very important to store all the relevant constraint information for the action in the PAR so that it can be easily used later during the motion regeneration process. To completely define a constraint, we need to know the exact objects in the PAR that will be involved in the constraint, the start and end contact relationships between the various objects, and if any of the involved objects need to change location during the motion segment. For the start and end contact relationships, we can directly use the parameters *EEObjRelStart, EEObjRelEnd* and for the change of location of the object we can use the parameter *ObjChangeLoc* that were all derived (section 4.4.1) for the PAR characterization process. For the objects in contact, we use the *CollidingObjects* feature generated for the PAR generation process.

With all the above considerations, we have extended the structure of the PAR to include the constraint information as shown in Fig. 4.16. The CaPAR system sets all the constraint parameters with the extracted information.

4.6.2 Creating Complex PARs

As mentioned in Section 4.3.2, the CaPAR system considers each tracking sensor separately, segments the motions in the corresponding kinematic link and generates UPARs for each segment. It now needs to combine all the UPARs together to form one complex UPAR representing the observed action. For this, we have built a hierarchical motion abstractor.

```
type core semantics =
        (motion: MOTION;
         force: FORCE;
         ......
         constraint: CONSTRAINT;
         )
type CONSTRAINT =
        (end-effector: int;
         CollidingObjects : int,int;
         EEObjRelStart: string;
         EEObjRelEnd: string;
         ObjChangeLoc: string;
         ApplyForce: string;
        )
```

Figure 4.16: Action constraint added to the core semantics of the UPAR

At the first (lowest) level, the individual motion segments are abstracted and characterized as PARs as described in (section 4.5). At the second level, individual kinematic links are observed by the CaPAR system and the UPARs identified for each motion segment in the link are connected together sequentially in time. If there is only one kinematic link, then the complex UPAR is just a sequential combination of these UPARs. At the third level, the system has to combine these sequential PAR chains to form the complex PAR. One way of forming the complex UPAR for the whole action is to simply connect these sequential UPAR chains in parallel. This would be sufficient if the kinematic links were completely independent of each other. But, generally, this is not the case. If there is more than one active kinematic link, then they are interdependent. For this, the CaPAR system needs additional information to temporally synchronize the various UPARs in the different kinematic links to ensure that the relative timings of the motion segments in the different kinematic links are maintained correctly. For example, if in the observed action, the agent first reached for the object with the right hand, touched the object for some time, then reached to touch the object with the left hand (while continuing to touch with the right hand) and then used both hands to lift the box, the system needs to replicate this exactly with a secondary agent. To accomplish this, it normalizes the observed duration of the individual motion segments by the duration of the entire action such that the sum of all the normalized durations in a kinematic link equals 1. It then pass this duration as a parameter of the primitive action in the execution steps for the complex action. The complex UPARs are generated as a parallel combination of the sequence of actions in the individual kinematic links. During motion regeneration, each of these normalized durations is multiplied by the user-specified duration (which maybe different from the observed duration) for the complex action such that the relative start and end instants of the various motion segments, executing either sequentially or in parallel, is maintained correctly. This can be used for characterizing and recognizing coordinated actions like *pickup box with both hands, clapping, etc.*

Among the high-level parameters of the complex action, the CaPAR system only generates specific information for the execution steps. The preparatory specifications, the termination conditions and the applicability conditions for the complex action will be taken care of by the individual primitive sub-actions that make up the complex action. Hence, there is no need for the system to generate them again for the complex action. In the execution steps of the complex action, the system specifies the complex combination of the individual primitive actions, derived from the hierarchical motion abstractor as described above. In addition to specifying the observed normalized duration as a parameter of the individual primitive actions, the system also specifies some specific parameters for the individual primitive motions. We will now discuss each of these parameters.

Agents: During the PAR generation process, this is just a variable name inside the Python script and will be initialized by a user specified agent name during the motion regeneration process.

Objects: During the PAR generation process, these are just variable names inside the Python script and will be initialized by the user specified object names during the motion regeneration process.

Duration: This is the duration of the primitive action, normalized by the CaPAR system over the duration of the entire observed complex action. During the motion regeneration process, this normalized value is multiplied by the new user-specified duration for the complex action to derive the correct duration for the corresponding primitive action.

Purpose: This is a user-specified string identifying the *purpose* of the primitive action. As explained in Sections 4.4.2.2 and 4.4.2.3, for the correct execution of the sub-action, the *purpose* of the action is used to correctly retrieve the constraint locations and the agent's style of executing the action from the corresponding objects and the agent.

EE: This is the specific end-effector used in the primitive action. During the PAR characterization process, we do not distinguish between right hand and left hand or between left foot and right foot in the action. Instead, we only use a general type of end-effector like hand, foot, etc. But during the motion regeneration process, we need to know the exact end-effector (left-hand, right-hand, left-foot, etc.) to be used in order to generate the motions correctly. So, the CaPAR system uses this parameter to specify the observed end-effector. For example, in a two-handed reach action, the complex PAR may be a parallel combination of two *Reach* actions. It is the EE parameter that helps in completely specifying the parallel combination of left-handed reach and right-handed reach.

ComplexParent: This is the user specified name of the observed complex action. It is used during the generation of the instantiated PAR for a secondary agent to retrieve some parameters from the complex action on which the primitive action's parameters are dependent.

Posture: This is a pair of strings containing the posture name and the posture file name. If either the first or the last sub-actions in the complex PAR does not involve any constraints at the end of the sub-action, then the CaPAR system automatically records the posture of the agent in a file and sets the posture name and posture filename parameters.

For the *DrinkFromMug* complex action, where the agent is assumed to be in an initial standing posture, the abstracted and generated execution steps described in Python are as follows:

```
from actionDef import *
def execution_steps(self, agent,obj1,obj2,obj3):
         complex = (SEQUENCE,("Reach", {'agents':agent,
                          'objects':(obj1),
                          'duration':0.445274,
                          'purpose':"PickUp",
                          'EE':15,
                          'ComplexParent':"DrinkFromMug"
                         }),
              ("MoveObjTo", {'agents':agent,
                         'objects':(obj1,obj3,obj2),
                          'duration':0.171642,
                          'purpose':"Drink",
                          'EE':15,
                          'ComplexParent':"DrinkFromMug"
                          }),
              ("Drink", {'agents':agent,
                          'objects':(obj1,obj2),
                          'duration':0.0472637,
                          'purpose':"Drink",
                          'EE':15,
                          'ComplexParent':"DrinkFromMug"
                          }),
              ("MoveObjTo", {'agents':agent,
                          'objects':(obj1,obj2,obj3),
                          'duration':0.121891,
                          'purpose':"PutDown",
                          'EE':15,
                          'ComplexParent':"DrinkFromMug"
                          }),
              ("TouchWith", {'agents':agent,
                          'objects':(obj1,obj3),
                          'duration':0.0472637,
                          'purpose':"Release",
                                        'EE':15,
                                        'ComplexParent':"DrinkFromMug"
                                        }),
                                ("Position", {'agents':agent,
                                        'objects':(obj1),
                                        'duration':0.179104,
                                        'purpose':"Release",
                                        'EE':15,
                                        'ComplexParent':"DrinkFromMug",
                                        'Posture':("drinkEnd",drinkEnd.post"}))
        actions = {'COMPLEX': complex}
        return actions;
```

4.7 Motion ReGeneration from PAR

In this section, we describe the technique to generate motions for any secondary agent within the PAR architecture [8]. To regenerate any of the observed complex agents for a different agent in a different environment, the user first specifies the complex action's name, the new agent's name and the objects in the environment to be used for the complex action. With this information, the corresponding UPAR of the specified complex action is instantiated by the PAR system to form the IPAR (Section 4.1.2). The user can additionally specify a new duration for the entire complex action. Using this information along with the parameters specified for each of the primitive actions in the execution steps of the complex action, the corresponding UPARs of the primitive sub-actions are instantiated correctly by the PAR system to form individual IPARs. For the complex action, the user can also specify an agent whose style of executing a similar complex action can be copied. This agent's style is then used for all the sub-actions.

During the motion generation of the primitive actions, the CaPAR system first maps the instantiated objects in the IPAR to the objects, specified in the *colliding objects* feature of the action constraint (Fig. 4.16) in the UPAR, and determines the specific objects that will be actively involved in the new constraint. The *colliding objects* feature specifies the two objects that should form the constraint for this primitive action. The object specified as the first parameter in the *colliding objects* feature is the end-effector in the constraint and the second object is the goal object. With this knowledge and using the UPAR name, the purpose specified as a parameter of the primitive action, and the side to be used, the CaPAR system correctly extracts the eeNLT and the goalNLT from the corresponding object models. Now, knowing the current size and the locations of the objects in the new environment, the system computes the global constraint locations. Next, using the *EEObjRelStart, EEObjRelEnd,* and *ObjChangeLoc* features, the CaPAR system determines the type of constraint to be established. Independent of the action to be performed, it is found that all the actions to be generated from a PAR derived from the CaPAR system can be generalized to four specific cases:

1. There is initially no contact between the end-effector and the object. This happens when $EEObjRelStart[i] ==' 0'$ and $EEObjRelEnd[i] ==' 1'$ where i corresponds to the index of the specified object in the object list of the sub-action. At the end of the motion, the CaPAR system needs to establish a constraint between the end-effector and the identified object such that they come in contact: for example, *Reach, Hit.* In this case, inverse kinematics is used to solve the constraint. During each frame of execution of the motion, the system uses the specified style information to interpolate the end-effector location, between the start and goal locations, and uses inverse-kinematics to solve for all the relevant joint angles. If a visual constraint exists, it solves the spatial and visual constraints simultaneously.

2. The end-effector and the object are already in contact. In addition, the object needs to change location during the action. This happens when $EEObjRelStart[i] ==' 1'$ and $EEObjRelEnd[i] ==' 1'$ and $ObjChangeLoc[0] ==' 1'$: for example, *MoveObj, MoveObjTo.* In this case, the CaPAR system slowly moves the object from the current location to the computed goal location. At each frame of the animation, the interpolated location of the object between the start and the goal locations is used as the new location of the object. The system solves for the individual joint angles indirectly by allowing the end-effector which is already in contact with the object to passively follow the object.

3. The end-effector and the object are already in contact. Also, the object does not change location during the action. This happens when $EEObjRelStart[i] ==' 1'$ and $EEObjRelEnd[i] ==' 1'$ and $ObjChangeLoc[0] ==' 0'$: for example, *Touch, TouchWith.* In this case, the system does nothing and just maintains the contact relationship for the specified amount of time.

4. The end-effector and the object are initially in contact and should no longer be in contact at the end of the action. This happens when $EEObjRelStart[i] ==' 1'$ and $EEObjRelEnd[i] ==' 0'$ and all $EEObjRelEnd[i] ==' 0'$ indicating that the end-effector will not be in contact with any object at the end of the action: for example, *ResumePosture, Neutral.* In this case, the system slowly changes the posture of the agent to that specified in the *posture* parameter of the primitive action in the execution steps of the complex action. At each frame of the animation, the CaPAR system interpolates the joint angles, from the current posture to the final specified posture.

4.8 Summary

In this chapter, we have presented our method for generating complex PARs from observing motion captured data using the CaPAR system. Using our results from Chapter 3, we have shown how the CaPAR system segments the motions, extracts all the semantic features from an action, and combines them to form complex PARs. We have also shown the technique for generating motions from the extracted PAR for any secondary agent in a different situation.

Chapter 5

Results

In chapters 3 and 4, we described in detail our technique for motion abstraction, parameterization, and mapping. In this chapter, we discuss the results obtained by applying these techniques on various sets of data. In Section 5.1, we describe our experiment setup, the techniques we used to overcome some inherent problems, and the various sources of data. In Section 5.2, we discuss in detail the results of motion abstraction and parameterization for some specific examples. In Section 5.3, we discuss the results of motion regeneration from the PARs generated for these examples. Finally, we conclude this chapter with section 5.5 where we summarize and present our observations of the obtained results.

5.1 Experiment Setup

The main goal of the CaPAR system is to observe a single trial of a human performing some complex task involving interaction with self or other objects in the environment and to automatically generate semantically rich information about the action in such a way that the action can be reproduced by any sized virtual human model. To accomplish this, the single most important input into the CaPAR system is a stream of 6DOF data representing the 3DOF position (x,y,z coordinates) and the 3DOF orientation (rotation in the axial directions) of the various sites of a human over the duration of the performed action. These sites correspond to various points on the body that would yield enough information for reproduction of the subject's body movements. This 6DOF data can be obtained naturally and most effectively by the process of *motion capture*. It can also be obtained from manually generated keyframe, procedural, or scripted animations. There are multiple sources of motion captured data - magnetic, optical, video, etc. The techniques introduced in this thesis are applicable to any of the above types of data.

As explained in Section 3.1, we have chosen to use the MotionStar system, from Ascension Technology. It consists of one Extended Range Controller (ERC), one Extended Range Transmitter, and 12 Bird units, each controlling a single sensor. Although this system is cost-effective, productive and efficient in generating data, it has one main drawback. It is an electro-magnetic tracker and is hence susceptible to interference from neighboring external sources of fields [50]. The interference of the power supply frequency and its harmonics with the sampling frequency introduces noise. To minimize this, we use a sampling frequency of 103.3 Hz. Further, we have conducted several experiments to calibrate the space around the transmitter in an effort to further reduce noise in the data. We have found that the best data is obtained when the transmitter is placed more than 6 feet away from the controller and when the sensors are all within 3-5 feet of the transmitter. Also, all the sensors have to be at a minimum height of one foot above the ground to minimize noise from cables under the floor in the laboratory. For this, we use a wooden platform which is 4 feet wide, 8 feet long, and 1 foot high. The human performers stand on this platform and perform the various actions.

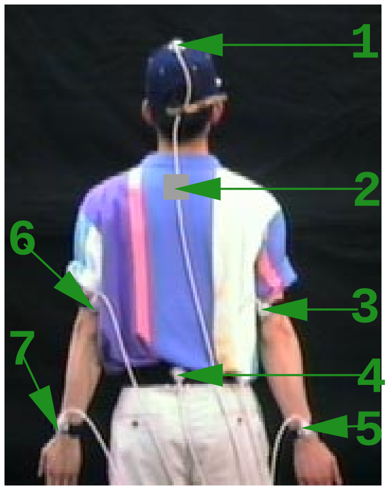


Figure 5.1: Locations of 7 electro-magnetic sensors on the upper body of a person

During a typical motion capture session, we use 12 magnetic sensors. Eleven of these sensors are on the human body and the twelfth sensor is placed on the object that the person will interact with. One of the sensors is always placed on the lower back of the subject roughly corresponding to the *L5* segment of the spine (sacro-iliac). This is used for calibrating the virtual human model with the human performer. For recording the upper body motions, we need six sensors in addition to the sensor at the lower back. These sensors are placed in the following places: top of the head, base of the neck, one at the back of each hand, one at the back of each elbow on the bony portion just above the elbow joint. Fig. 5.1 shows the the placement of 7 sensors on a subject for recording upper body motions. For the lower body motions, the four sensors are placed in the following places : one above the knee joint on each leg and one on the top side of each foot. All the sites were chosen for the amount of information they would yield for the reproduction of the subject's body movements. The sensors at the hands and elbows are affixed to the subject using 3/4" wide velcro straps. Each strap is wrapped firmly around the specific site to ensure that shifting during movement does not occur. The sensor at the sacro-iliac joint is affixed using a 2" wide velcro strap, much like a belt, again firmly wrapped so that shifting does not occur. The receiver at the top of the head is mounted on a tight fitting baseball cap, and the cap is placed on the subject's head such that the receiver is on the top. Receivers are attached to the velcro straps and to the cap using nylon screws (white plastic), to avoid the use of metal near the magnetically-based receivers.

The first step in deriving motions from motion capture data for the primary agent is to build an avatar that is the same size as the subject. The human model in *Jack*®comprises a number of segments each of whose measurements are required to generate an accurate model. So, after the actual motion recording, the subject's anthropometric data is taken. Twenty-five sites, including the upper and lower arm, upper, lower and center torso, upper and lower legs, neck, head, hands and feet, corresponding to the SASS anthropometric database [4] sites, are measured in the x, y and z directions. These measurements are input to *Jack*®and the desired human model is generated.

As explained in Section 3.1, to generate motions in the virtual human model, we first establish constraints between the human model and the sensors. We then use inverse kinematics to solve for the constraints. This causes the human model to closely follow the

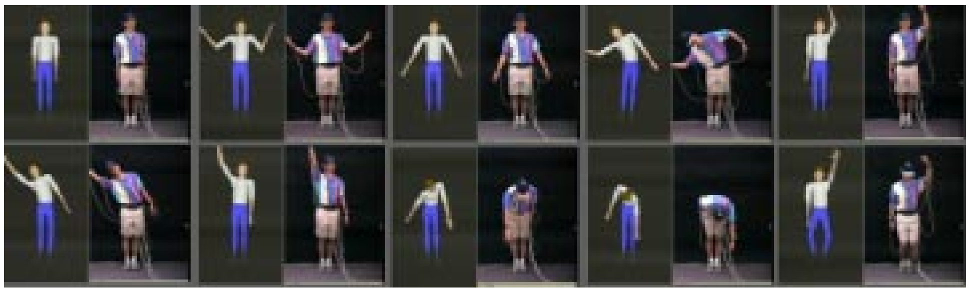


Figure 5.2: Real-time motion capture - Generating motions for a virtual human model

performer. Fig. 5.2 shows a sample session where the human model is following the actions of the performer in real-time.

To collect enough data for the experiments and to evaluate the results from the CaPAR system, we collected data from 3 subjects - 2 male and 1 female. While it was relatively easy to generate human models corresponding to the male subjects, it was difficult to do the same for the female subject. We attribute this to inherent problems in the *Jack*®system, which generates models based mainly on male data.

In the CaPAR system, we are mainly interested in observing and abstracting complex actions. Hence we recorded the following actions for each of the 3 subjects: *Drink From Mug, Touch Object, Slide Object,* and *Pick Up Box With Both Hands.* We will discuss each of these examples in detail in section 5.2. While studying agent-object interactions, one of the main tasks before analyzing a simulated action in a virtual environment is to ensure that the virtual environment emulates the real environment to the smallest detail *i.e.,* we need to ensure that the sizes of the relevant objects and their placement in the virtual environment replicates the real environment. But, even with all the precautions taken to minimize errors in the motion capture data, we find that the inverse kinematics may introduce some new errors. We compensate for it by slightly changing the sizes and the locations of the objects. So, with a little trial and error, we are able to build a virtual environment in which the actions of the performer can be generated for the human model and used reliably for further analysis. Note that this perfection is required only during motion abstraction and not during motion regeneration.

5.2 Results from Motion Abstraction and PAR Generation

In this section, we discuss the results obtained by using our techniques of motion abstraction and PAR generation on several complex examples. Each of these examples was motion captured using the techniques described in Section 5.1.

5.2.1 Drink From Mug

For this example, each of the performers was asked to stand in front of a table on which was placed a mug. One of the electro-magnetic sensors was attached to the mug by velcro straps. In all our examples, the performers were initially in a neutral standing posture not in contact with any object in the environment. For this example, the performers were asked to pickup the mug, drink from it, put the mug back on the table and to go back to a posture similar to the initial one. (Note: it was not necessarily the exact same posture.) No further specific instructions were given. The data was recorded and motions were generated in the corresponding primary agents using the techniques described in Sections 3.1 and 5.1. We found that the data from the two male subjects yielded correct motions whereas the data from the female subject was not good. This could be attributed to the fact that the female's body size did not correctly match the actual performer's measurements and hence the collision with the *self* type of tag object (mouth of the agent) could not be detected correctly. Hence, we discarded the female subject's results. For the primary agent's actions, three tag objects (Section 3.2.3) and two monitoring sensors (Section 3.2.2) were identified by the user. The tag objects were the mug, the table on which the mug was resting, and the head of the primary agent. The monitoring sensors were the right hand and the head of the primary agent.

Using the techniques described in chapters 3 and 4, the resulting motions of the two subjects were analyzed, segmented and parameterized . We obtained two main results. One of the results showed, as expected, the information from the actions of the two subjects to differ in durations of the primitive sub-actions, in the style of execution of the action, and in the locations of the spatial and visual constraints. But the single most important result was that the parameters extracted for characterizing the primitive PARs from any motion segment were exactly the same for both sets of data. Accordingly, from each of the motions, the same complex PAR shown in Fig. 5.3 was automatically generated. The execution steps generated for this complex action was shown and described in Section 4.6.2.

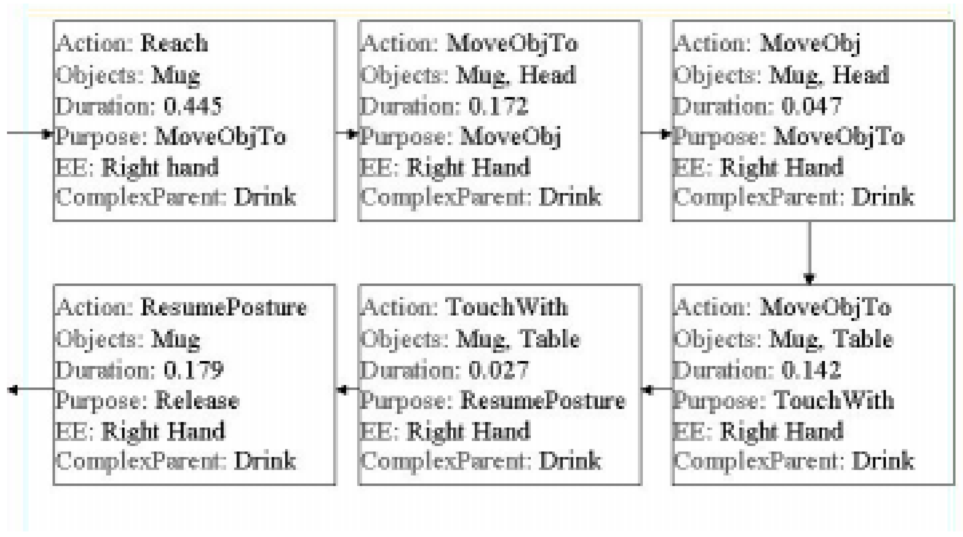


Figure 5.3: Complex PAR generated for the action *DrinkFromMug*

As explained in Section 4.5, if an extracted feature set has not been previously stored in the feature table, the user has to specify the UPAR name to associate the primitive action with. We will now consider each of the derived motion segments and discuss the association of the extracted semantic features with their UPAR names.

Reach The following features were extracted from both the data sets for the first motion segment, where the subject, who is in an initial neutral standing posture not touching anything, reaches across for the mug with the explicit purpose of picking it up:

Number of objects: 1

EEObjRelStart: 0

EEObjRelEnd: 1

ObjChangeLoc: 0

While reaching across to touch an object, we know that there is only one important object - the object to reach to. The object (table) on which the mug is resting is not important for the reaching action. We note that even though the total number of objects for the *DrinkFromMug* action is 3 (the number of tag objects identified above), the CaPAR system, using the technique explained in section 4.4.1, correctly estimates the number of objects for this motion segment to be 1. From the other parameters, it can be understood that the end-effector of the primary agent is not initially in contact with any object but comes in contact with an object at the end of the action. The first time this feature set is encountered, the user identifies it as a *Reach* action. Subsequently, for the same or for other complex actions, a motion segment with these same features will be correctly matched against the existing feature set in the feature tree (explained in Section 4.5) and identified automatically by the CaPAR system as a *Reach* action.

MoveObjTo This corresponds to the motion segments in which the end-effector is in contact with the primary object both at the start and at the end of the motion. Additionally, the primary object moves during the motion to come in contact with another object at the end of the motion. This is identified by the features:
Number of objects: 2
EEObjRelStart: 100
EEObjRelEnd: 101
ObjChangeLoc: 10
In the *DrinkFromMug* complex action, the CaPAR system identifies this sub-action twice. The first instance corresponds to the second motion segment where the agent picks up the mug and takes it to his mouth for the purpose of drinking. The second instance corresponds to the fourth motion segment where the agent moves the mug from his mouth to rest it on the table for the purpose of putting it down. When considered very generally, in both the motion segments, the primary object (mug) is brought in contact with a secondary object (the mouth in the second motion segment and the table in the fourth motion segment). The object that the primary object is in contact with at the start of the motion (the table in the second motion segment and the mouth in the fourth motion segment) is not important. But the object that the primary object will come in contact with at the end of the motion is important. Hence the CaPAR system correctly identifies the number of objects in this motion segment to be 2. The motion characteristics of both instances of this sub-action are

the same except for the objects that the primary object will come in contact with at the end of the motion and the points of contact. The specific secondary objects for each instance are specified as parameters of the primitive motions as described in Section 4.6.2. The exact points of contact are stored as NLTs in the respective object models as described in Section 4.4.2.2. Hence, during motion regeneration, the two distinct instances of the *MoveObjTo* sub-action are executed correctly.

MoveObj This corresponds to the motion segments in which the end-effector is in contact with the primary object and the primary object is also in contact with the secondary object both at the start and at the end of the motion. Additionally, the primary object moves during the motion while continuing to maintain contact with both the end-effector and the secondary object. This is identified by the features:
Number of objects: 2
EEObjRelStart: 101
EEObjRelEnd: 101
ObjChangeLoc: 10
In the *DrinkFromMug* complex action, this sub-action corresponds to the third motion segment where the agent, who in the previous motion segment of *MovObjTo* had taken the mug to the mouth for the purpose of drinking, now actually drinks from it. During the process of drinking, the mug is naturally moved or tilted to get to the last drop of the liquid in the mug. But, we are not concerned with the contents of the mug.

TouchWith This corresponds to the motion segments in which the end-effector is in contact with the primary object and the primary object is also in contact with the secondary object both at the start and at the end of the motion. Unlike in the *MoveObj* sub-action, the primary object in this sub-action remains stationary during the motion. This is identified by the features:
Number of objects: 2
EEObjRelStart: 101
EEObjRelEnd: 101
ObjChangeLoc: 00

In the *DrinkFromMug* complex action, this sub-action corresponds to the fifth motion segment where the agent, who in the previous motion segment of *MovObjTo* had put down the mug on the table, now continues to hold the mug on the table.

ResumePosture This corresponds to the motion segments in which the end-effector and the object are initially in contact but are no longer in contact at the end of the action. Also, at the end of the action, the end-effector is free and not in contact with any object in the environment. This is identified by the features:
Number of objects: 1
EEObjRelStart: 1
EEObjRelEnd: 0
ObjChangeLoc: 0
In the *DrinkFromMug* complex action, this sub-action corresponds to the last motion segment where the agent releases contact with the mug and moves back to a neutral posture. The final posture that the agent moves into is recorded and sent as a parameter of the primitive action within the execution step of the complex action. This is described in Section 4.6.2.

5.2.2 TouchFromRest

For this example, each of the performers was asked to stand in front of a table on which was placed a box. One of the electro-magnetic sensors was attached to the box by velcro straps. Again, the performers were initially in a neutral standing posture not in contact with any object in the environment. The performers were asked to reach across and touch the box. No further specific instructions were given. We later realized that one of the agents had stopped the action after touching the box whereas the other two agents had gone back to the neutral posture after touching the box for a small period of time. Accordingly, we derived two different complex actions - *TouchFromRest* and *TouchCycle*. The complex PARs for these actions are shown in Fig. 5.4 and Fig. 5.5 respectively.

It is important to note that both instances of *TouchFromRest* generated the same complex PAR. Also, as expected, the complex PAR for the *TouchCycle* action was the same as that generated for the *TouchFromRest* action except for the addition of the

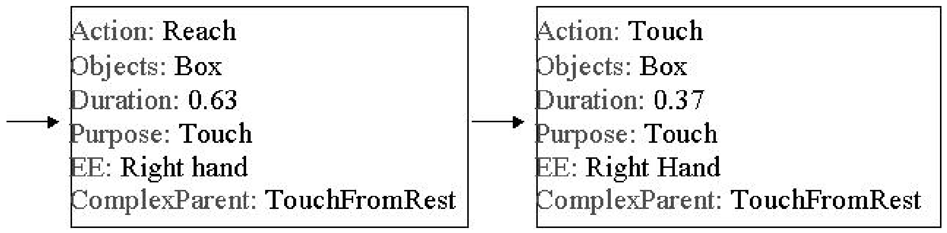


Figure 5.4: Complex PAR generated for the action *TouchFromRest*

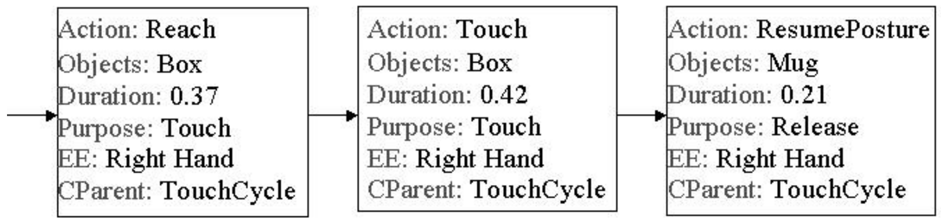


Figure 5.5: Complex PAR generated for the action *TouchCycle*

ResumePosture sub-action, which correctly corresponded to the agent going back to the neutral position. As the *Reach* and ResumePosture changes have been explained in detail in section 5.2.1, we will now only consider the *Touch* action.

The *Touch* action corresponds to the motion segments in which the end-effector is in contact with the primary object both at the start and at the end of the motion. Also, like the *TouchWith* sub-action, the primary object remains stationary during the motion. Basically, the *Touch* action is a more general form of the *TouchWith* action. This is identified by the features:

Number of objects: 1

EEObjRelStart: 1

EEObjRelEnd: 1

ObjChangeLoc: 0

In the *TouchFromRest* or *TouchCycle* complex actions, this sub-action corresponds to the second motion segment where the agent, who in the previous motion segment of *MovObjTo* had reached across to touch the box, now continues to touch the box. For the simple *touch* action, the object that the box is resting on is not important.

5.2.3 SlideObject

The setup for this action was similar to the *TouchFromRest* complex action. The performers were asked to reach across to touch the box and to move it. The PAR generated for this action for each of the three agents is shown in Fig. 5.6. The complex PAR for this action consists of the *Reach* primitive action followed by the *MoveObject* primitive action in sequence.

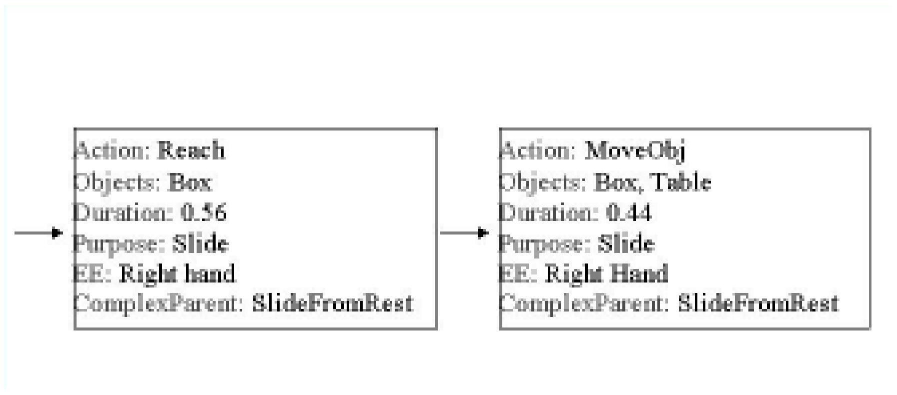


Figure 5.6: Complex PAR generated for the action *SlideObject*

5.2.4 PickUpWithBothHands

The setup for this action was similar to the *TouchFromRest* complex action except that a bigger box was now placed on top of the table. The performers were instructed to pick up the box with both hands. For this action, we found that only one of the sets of data was good for further analysis. The problem was that the inverse kinematics was not able to reproduce the motions exactly for the primary agent from the other two subjects' data. Hence, we discarded the two sets. Fig. 5.7 shows the block diagram of the complex PAR for this action. This action involves two kinematic links - the right arm and the left arm. The movements of the two links have to be carefully synchronized in time. As explained in Section 4.6.2, the normalized durations for the individual sub-actions are computed and sent as a parameter to the corresponding primitive action. When each kinematic link is analyzed, the *ResumePosture* action is separately recognized by each link. As explained in section 4.7, the motions for this action are regenerated by returning the agent to the specified posture using joint angle interpolation. Instead of generating

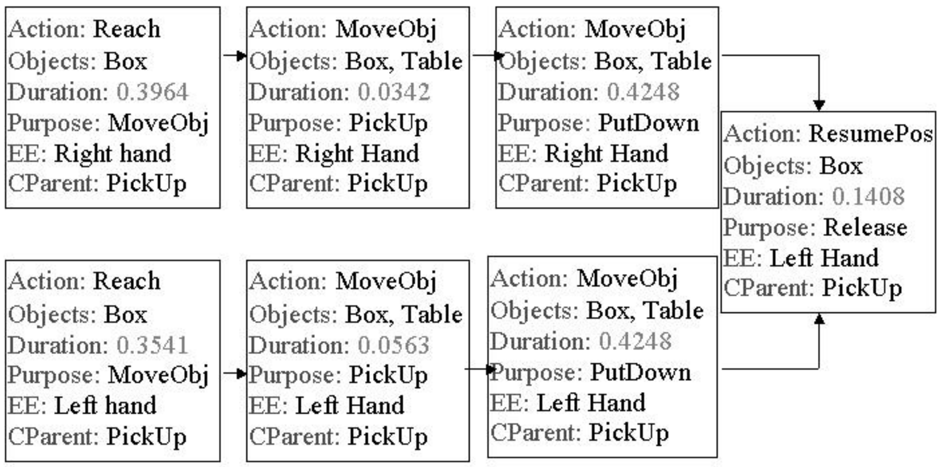


Figure 5.7: Complex PAR generated for the action *PickUpWithBothHands*

two different motions in parallel for the same action, the CaPAR system reduces it to one action and executes it in sequence after the rest of the sub-actions in the two links have finished executing.

Referring to Fig. 5.7, the second motion segment has been identified as *MoveObj*. This should have actually been *TouchWith*, where the agent touches the box, resting on the table, before picking it up. The discrepancy can be attributed to a small motion in the box (due to noise in the data), which caused the system to recognize it as a *MoveObj* action. As explained in Section 5.2.1, the features of the *TouchWith* and *MoveObj* actions differ only in the motion of the objects.

The second *MoveObj* sub-action refers to the motion segment in which the agent lifts up the box and lowers it down onto the table. This should have correctly been recognized by the system as two separate sub-actions - lift and lower. The reason for identifying the combined actions of *lift* and *lower* in only one motion segment instead of two is that in our system, the end of an action is recognized either by the end of the entire complex action or by the co-occurrence of the zero crossings of acceleration and the spatial proximity of two objects. If the *lift* action occurs in the middle of a complex action, the corresponding motion segment is easily recognized if the object that is lifted comes in contact with another object at the end of the action. But, in this example, it was not so. Hence the separate motion segment for *lift* was not recognized by the system.

5.3 Results from Motion Regeneration

For each of the examples discussed above, the CaPAR system used the techniques described in Section 4.7 and generated the motions in a new environment consisting of an anthropometrically different sized agent and different sized objects placed at different locations. As we are mainly concerned with constraint based motions, we use inverse kinematics to generate most of the motions. Unexpectedly, we came across some inherent problems with the inverse kinematics. As designing new inverse kinematics solutions was beyond the scope of this thesis, we overcame these problems in different ways. We discuss reasons for our problems with inverse kinematics and our solutions in Section 5.3.1. In section 5.3.2, we discuss the automatic process of generating and checking for preparatory specifications and termination conditions in real time. In Section 5.3.3, we discuss the effect of using the different extracted styles during motion regeneration. Finally, in Section 5.4, we discuss the results of evaluation by external people of the motions generated with different styles.

5.3.1 Inverse Kinematics

Previously, as explained in Section 3.4, during the process of mapping the motions directly to a secondary agent, we had generated new motions only for the kinematic chains containing the monitoring sensors. For the remaining kinematic chains (for example, the spine), we had used the same joint angles as that of the primary agent. This was partially possible as the rest of the environment was assumed to be exactly the same. But, the PARs generated from the CaPAR system have to be applicable to any new environment. This implies that the environment can be completely different. For example, assume that in the original environment, the object is in front of the agent on a low table. Then, to reach for the object, the agent would bend forward and down. Now, in the new environment, let the object be placed on a high table slightly to the left of the agent. Then, to reach for the object, the agent has to slightly bend forward and also twist to the left. This action would not be possible if we were to use the same joint angles as that of the primary agent. Hence, we need to generate new joint angles for all the effected kinematic chains. Currently, we have only considered upper body motions. This means that we need to generate new joint angles specifically for the spine. To accomplish this, we allow the constraint between a hand and the object to be controlled from the waist instead of from the shoulder. While this satisfies one criterion, it generates other problems. The main problem is that the kinematic chain for the new constraint between the waist and the hand is very long, consisting of numerous joints and degrees of freedom. So, there is a lot of redundancy and the solution obtained is not very stable. To overcome this problem, we consider the visual constraints that we also recognize and store (section 4.4.2.2) during the motion abstraction process.

During the motion regeneration process, the CaPAR system computes the new locations for the visual constraints and then creates a new constraint to rotate the head in the direction of the visual constraint. This constraint controls the joint at the base of the neck. We found that solving the spatial and visual constraints in parallel got rid of most of the redundancy problems and we got better solutions. Still, some quirks can be occasionally found in the resulting animations that can be directly attributed to the problem with the inverse kinematics. These can be removed by using better inverse kinematic techniques.

Finally, it is possible that not all spatial constraints have associated visual constraints. To take care of such cases, we initially experimented with using the goal location of the spatial constraint as the location of the visual constraint. This would correspond to the naturalness of people looking at the object they are interacting with. But the resulting motions were not good. Hence, we consider two separate solutions. If there is no visual constraint, implying that the agent was not looking at anything specific, then the system solves for the spatial constraint by controlling only the corresponding arm. But, if there a visual constraint, then it controls the spatial constraint from the waist while solving it in parallel with the visual constraint.

5.3.2 Preparatory Specifications and Termination Conditions

As explained in sections 4.6.1.1 and 4.6.1.2, int the current implementation, the preparatory specifications and termination conditions cannot be fully specified and checked for within the Python scripts for the UPARs. Hence, the CaPAR system generates and checks for those conditions at run-time during the process of motion regeneration.

As we are only interested in contact-based actions, the system needs to check for the contacts specified in the *EEObjRelStart* and the *EEObjRelEnd* relations of the action constraint in the corresponding UPAR. This specifies the contact relation between the end-effector and the other objects and between the primary object and other secondary objects at the start of the motion segment. During motion regeneration, the CaPAR system needs to check for the contacts between the corresponding end-effector and the instantiated objects. Instead of checking for physical contacts using collision detection techniques, we simplify this process by maintaining a global table within the PAR system to keep a record of the contact status between the various pairs of objects or between any end-effector and any object. All the end-effectors and the objects in the table are only identified by their names. During initialization of the environment, the table is initialized by the user, with the contact status between relevant pairs of objects and end-effectors. Then, as part of the post-assertion process (Section 4.1) of each primitive PAR, the CaPAR system updates the contact table with the new contact status for the pairs of objects. Then, when checked either for preconditions or for termination conditions, the correct contact status is obtained.

For the preparatory specifications, in case of failure of the preconditions, we need to execute another primitive action to achieve the precondition and make it true. But, as explained in section 4.6.1.1, the associated actions to satisfy these preconditions will always be either *Reach*, which causes the end-effector to come in contact with a specified object, or *MoveObjTo*, which causes the primary object held by the end-effector to come in contact with the specified secondary object. So, if *only* the end-effector needs to come in contact with an object, the system generates the *Reach* action and in all other cases, it generates the *MoveObjTo* action. To correctly execute these actions, the system needs to specify the purpose of the action and which end-effector to use. As explained in Section 4.4.2.2, the system needs to know the purpose of the action to correctly obtain the constraint goal locations from the corresponding objects. So, it passes the current UPAR's name as the purpose parameter for the associated action. For the end-effector, it specifies the current action's end-effector for the associated action.

During motion regeneration, the preparatory specifications and termination conditions generated at run-time by this method were checked for all the examples and found to work correctly.

5.3.3 Styles of Execution

As explained in sections 3.4 and 4.4.2.3, the different styles of execution of the action by the different primary agents are recorded during the process of motion abstraction. During motion regeneration, these styles are used to generate the motions for the specified secondary agents. The extracted *VelStyle* and *PathStyle* values for each action of each agent is stored in a file. During the initialization process of the motion regeneration, these style values (identified by the UPAR name of the action, the purpose of the action and the end-effector used) are read into the corresponding agent model. We minimize the amount of data stored for each action by using the compaction techniques discussed in Section 4.4.2.3. The effect of using the different styles was discussed in section 3.4.

During motion abstraction, we record both the styles. During motion regeneration, we use only one of the styles. Further, for some specific examples, we found that we were able to regenerate the complete action using the *PathStyle*, but not the *VelStyle*. For example, in the *TouchFromRest* example, the subjects were asked to indirectly approach the box that they had to touch; i.e. instead of reaching for the box directly, they were asked to move the arm randomly before touching. If we were to use the *VelStyle* for this, we would lose the information on the approach path. But with *PathStyle*, the random motions of the primary agent's arm were correctly reproduced in the secondary agent's motion. Also, for the *PickUpWithBothHands* action, even though the sub-actions were not correctly recognized as *lift* and *lower*, the motions generated by the CaPAR system for the secondary agent from the extracted information but using the *PathStyle* of the primary agent's motion, were found to correctly mimic the lift and lower motions of the primary agent. We used the *VelStyle* to correctly generate the motions for the *DrinkFromMug* and the *SlideFromRest* complex actions. We used the *PathStyle* to generate the motions for the secondary agent for the *TouchFromRest* and the *PickUpWithBothHands* complex actions.

5.4 Evaluations

In this section, we describe the evaluations done to validate both motion abstraction and motion regeneration results obtained from the CaPAR system. For evaluating the motion abstraction and parameterization processes, we obtained two sets of data for each action from each of the three subjects. The CaPAR system processed each data set separately and generated the same features for each motion segment as discussed above, thus self validating the techniques of motion segmentation, abstraction, and parameterization.

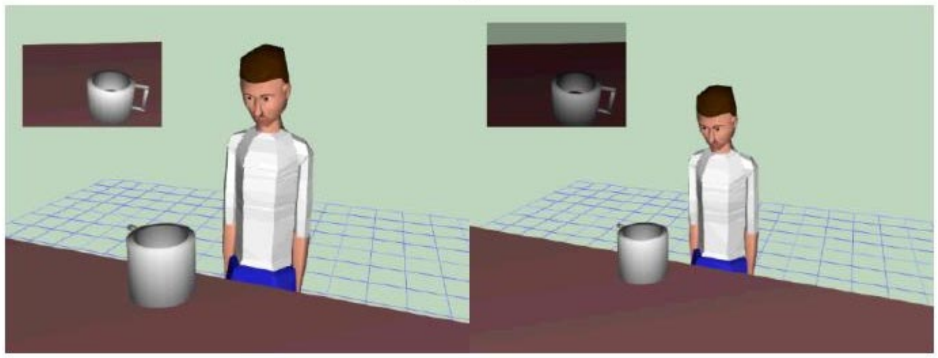


Figure 5.8: Evaluation of VelStyle

In the motion regeneration process, we used external evaluations to validate the style extraction and mapping processes. To do this, we asked 6 different people who were not familiar with the CaPAR system to evaluate it. Three of these people were experienced in generating various types of animations and the rest had absolutely no experience with such systems. First, we considered the *DrinkFromMug* action to test the *VelStyle*. We found that as all the subjects had performed the actions smoothly, there was very little difference between the *VelStyle* of the various agents. Hence we only evaluated the regenerated actions for the presence of the *VelStyle*. The experiment was setup as shown in Fig. 5.8. On the left side was the movie clip corresponding to the action regenerated for the child model using only the constant interpolation factor. On the right side, the movie clip corresponded to the action regenerated for the child model using the *VelStyle* of the first primary agent. The evaluators were not informed of these details. They were asked to watch both the movie clips and to report whether they found any visible difference between the two actions and to specify which action they found more natural. All the six evaluators correctly found that there was a distinct difference between the two motions and that the motion regenerated using the extracted *VelStyle* appeared more natural.

To evaluate the *PathStyle* extraction and mapping processes, we used the *TouchFromRest* complex action and set up the experiment as shown in Fig 5.9. We

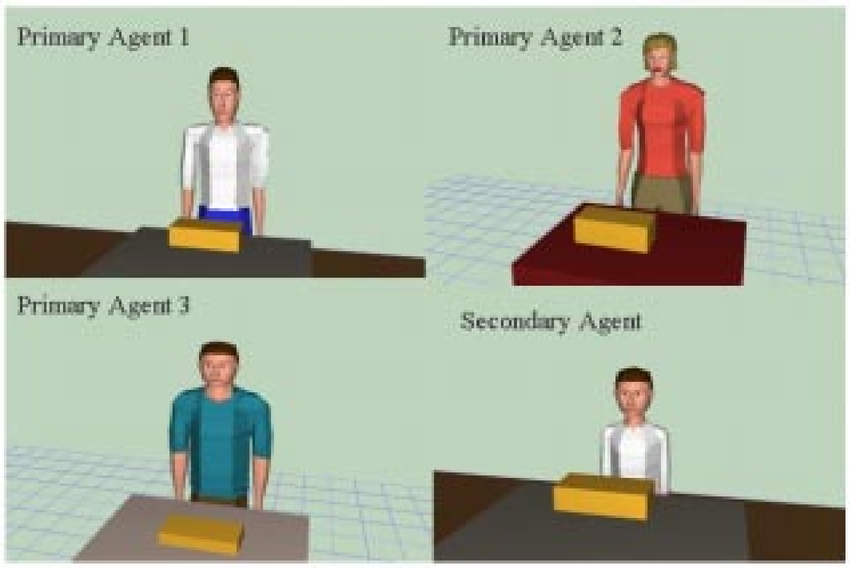


Figure 5.9: Evaluation of PathStyle

first played the movie clips of each of the primary agents and then played the movie clip corresponding to the regenerated motion for the secondary agent. The evaluators were asked to identify the primary agent who they considered the secondary agent's style was derived from. Five out of the six people correctly identified the primary agent. We attribute the single failure to the quirks in the elbow position introduced by the inverse kinematics as explained in Section 5.3.1.

5.5 Conclusion

In this chapter, we have presented all the results obtained for the motion abstraction and motion regeneration techniques. We have shown through multiple examples that the same PARs are generated for the same action even when performed by more than one agent. This substantiates our claim that a single trial of data is sufficient to abstract a motion and parameterize it for regeneration on other anthropometrically different-sized models. One of the most important observations that can be from this technique about motions for contact verbs is that all the contact-based complex actions are only a combination of a small set of primitive actions - *Reach, MoveObj, MoveObjTo, Hit, Touch, TouchWith*, and *ResumePosture*. These actions can then be used as good set of base actions from which different complex actions can be built. We have also shown that from a single PAR, we can regenerate motions for different agents with different styles in different environments containing similar objects, but of different sizes. Finally, we have shown that in case of actions corresponding to contact-based verbs, only a small set of extracted features - *EEObjRelStart, EEObjRelEnd*, and *ObjChangeLoc*, are required to distinguish most of the base primitive actions, to estimate the number of objects in a primitive action, to distinguish different cases of motion regeneration and to automatically generate the preparatory specifications and termination conditions for the actions.

Chapter 6

Contributions and Future Work

The main goal of this thesis was to build an interactive system that can observe a single trial of a person's actions and automatically abstract, understand, recognize and parameterize the observed complex physical action into an agent-size neutral semantic representation such that the actions can be reproduced by any sized virtual human model. We have described the process of building such a system, the CaPAR, in Chapter 4 and demonstrated its capabilities through several examples in Chapter 6. In this chapter, we summarize the capabilities of the CaPAR system, discuss our contributions and suggest some useful extensions to this thesis.

Broadly speaking, the techniques introduced in the CaPAR system are applicable to actions involving interaction with solid non-deformable objects and to actions that can be applied iteratively to a set of solid objects. This arises from one of our basic assumptions that the end-effector can be in contact with only one object at a time (section 4.3.2). We have tested the system extensively with several examples involving upper body motions. The system can be very easily extended to include full body motions. In section 6.1, we discuss the capabilities of the CaPAR system in terms of the linguistic classification of verbs. In section 6.2, we discuss all the contributions from this research. We conclude this thesis with section 6.3 where we discuss some of the relevant future work.

6.1 Linguistic Classification

In this section, we linguistically classify the set of actions that the CaPAR system is capable of addressing. We situate the actions that we have extracted from motion capture in terms of different verb classes. For this, we use two different classifications: WordNet [27] and Levin classes [38]. First, we associate the different verbs that we have extracted for our seven basic primitive actions with the different senses in WordNet as shown below:

Extracted Verb	WordNet Verb	WordNet Sense#
Reach	Reach	6
Hit	Hit	1,4
Touch	Touch	1,5,8
TouchWith	Touch	1,5,8
MoveObj	Move	2
MoveObjTo	Move	2
ResumePosture	Resume	2

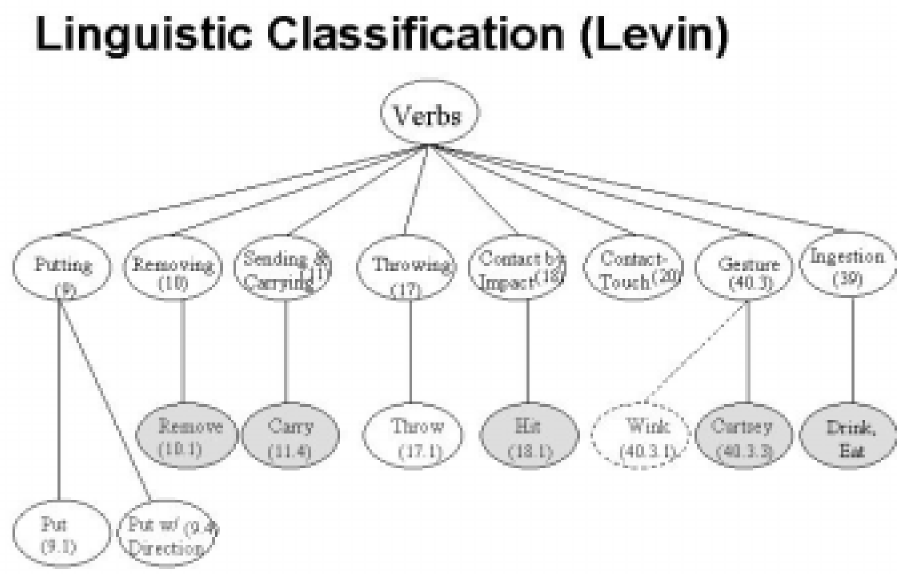


Figure 6.1: Levin classes of verbs addressable by CaPAR system

In the CaPAR system, we are interested only in actions involving interactions between an agent and other objects or with the self. We would now like to relate these actions with relevant linguistic verb classifications and study the applicability extent of the CaPAR system to these verbs. For this, we consider the verb classification suggested by Levin. Levin verb classes are based on the ability or inability of a verb to occur in pairs of syntactic frames that are in some sense meaning preserving. The sets of syntactic frames associated with a particular Levin class are supposed to reflect underlying semantic components that constrain allowable arguments and adjuncts. Fig. 6.1 shows the subset of Levin classes of verbs that we consider are related to the group of actions we are interested in. The numbers in the parenthesis refer to the section numbers in the book [38]. The CaPAR system can be applied directly to the verb classes shown in the shaded boxes. With a few extensions to the system, it can be applied to the verb classes shown in the non-shaded boxes outlined by solid lines. But it cannot be applied to the verb classes shown within the dotted lines. We will now consider each of these classes of verbs and discuss the applicability of the system towards it.

The CaPAR system is designed to recognize and identify gross motions like the movement of the arm, legs, head, etc. But it cannot recognize fine grained motions involving movement of the eyelids, fingers, lips, etc. like *blink eyes, point finger, squint eyes, etc.* (40.3.1) as we cannot readily record this type of information from the performer. Hence, the system can address general *verbs of contact*(20) like *touch* or *touch with.* For example, Carrie *touched* the box *with* the stick. But, it cannot address fine grained *contact*(20) actions like *tickle, caress, nudge, kiss, etc.* The system can easily address high level gestures like *clap hands*(40.3). Here, the contact is between the agent's two hands which the system can readily handle. We can also address *curtsy verbs*(40.3.3) like *salaam, salute, etc.* as they again involve contact between the hand and the head of the agent.

Hit is one of the primitive actions that we have shown to be recognized and parameterized (Section 3.4.3) by the CaPAR system. This allows us to readily address *verbs of contact by impact* (18) like *hit, kick, strike* and their related synonyms. *Throw*(17) verbs which cause ballistic motions by imparting a force, linguistically belong to a different class of verbs. But, in animations, they are very similar to *hit* verbs and just treated as a special case. In a *hit* action, the force is computed at impact. But in a *throw* action, the force is inherent in the arm holding the object and should be computed at the instant the hand releases the object. The CaPAR system can be easily extended to address the *throw* verbs. However it cannot be extended to handle *swat*(18.2) verbs like *bite, claw, pick, etc.*, which are also *verbs of contact by impact*(18), but involve fine grained motions of teeth or fingers.

In Section 5.2.1, we describe in detail the complex action of *drink*, that can be easily parameterized and regenerated by the CaPAR system. This allows us to easily address *verbs of ingesting*(39) like *drink* and *eat*.

The *MoveObj* and *MoveObjTo* primitive actions of the CaPAR system allow us to readily address *move* verbs which cause an object to be moved from one location to another. Although in general, we are not attempting to capture the differences in *manners of motion*(51) such as running and jogging, there are a few motion verbs which can be characterized to a simple *MoveObj* or *MoveObjTo* action. For example, the *carry* action which belongs to the *class of send and carry verbs*(11) defaults to a *MoveObjTo* action. A *slide* action is a specialized form of *move* which assumes that the object is in continuous contact with the surface of another object while it moves. Whereas we do not check for the surface contact during the motion, we can still readily address the *slide* action as long as it does not involve change of possession i.e., the agent has to remain in contact with the object during the motion.

The *put* action is a specialized form of the *MoveObjTo* primitive action. Linguistically, it has three obligatory arguments, two participants and a locative preposition like *in, into, on, etc.* Currently, we are not able to recognize from motion capture any prepositional relationships between the contacting objects. But, a very natural and important extension to the CaPAR system will be to include this. Then for motion regeneration, we can use the principles outlined in [65]. This will then readily allow the system to address verbs of putting like *put*(9.1), *rest*(9.1), *stood*(9.2), *funnel*(9.3), etc.

Verbs like *lift, raise, lower, etc.* are *put verbs with specific directions*(9.4). These verbs can be addressed by the system provided that they meet two additional requirements. First, the motion segments corresponding to these actions should not be followed by any other sub-actions in the whole complex action. In our system, the end of an action is recognized either by new contact relations or by end of the entire complex action. If the *lift* action occurs in the middle of a complex action, then the corresponding motion segment is easily recognized if the object that is lifted comes in contact with another object at the end of the action. If not, the motion segment will not be recognized. Secondly, the system needs to be extended to recognize relative directions of movement between two objects like *up, down, etc.* This will then allow the system to distinguish the *lower* and *raise* actions.

6.2 Contributions

The following are our contributions in this thesis:

- Introduced the concept of using zero crossings of acceleration to automatically segment motions, retarget motions, while maintaining spatial and visual constraints, and to identify key points in an end-effector trajectory.
- Demonstrated the capability to detect and map interactions with objects as well as with self.
- Automatically extracted various motion styles of the performer.
- Automatically extracted physics-based model parameters from an observed action.
- Demonstrated effectively that only a single trial of observed motion is sufficient to define an action for the purpose of action recognition.
- Built the interactive system that can automatically parameterize and build conceptual representations of observed complex actions using temporal event linking.
- Introduced techniques to automatically determine preconditions and termination conditions from the observed action.

6.3 Future Work

Here are some of our thoughts and suggestions on extensions to the CaPAR system that would greatly enhance its capabilities:

- As explained in Section 6.1, one of the most important extensions to this system will be to add the capability of recognizing prepositional phrases from actions. We currently recognize the spatial relationship between the objects in the environment in the form of spatial and visual constraints. But we do not recognize the locative prepositional relations like *in, into, on, etc.* This would greatly expand the range of actions that CaPAR can address.

- In the CaPAR system, we currently consider only point-to-point constraints. We could extend this to recognize and map point-to-line and point-to-plane constraints. This would also aid in the recognition of the locative prepositions.

- We have currently limited the set of actions to those involving only upper body motions. A natural extension of this system is to add the capabilities of addressing full body motions and also fine grained motions like the finger motions. The vocabulary addressable by the CaPAR system would then include *grasp, walk, jump, kick, etc.* But, while considering the lower body motions, we need to correctly determine the supportive or the weight bearing leg both during the abstraction and the motion regeneration processes.

- In all our actions, we assume that the end-effector can be in contact with only one object at a time. An interesting and easy extension to the system would be to allow multiple objects to come in contact with the end-effector at the same time thus allowing the system to address actions like *playing jacks.*

- During the motion regeneration process, we currently do not consider collisions that might occur between the agent's body and the new environment. A good addition to the system would be to modify the new motion paths using body-awareness techniques [68].

- In the CaPAR system, we currently rely on the user to inform the system of possible force interactions in the action. The system can be extended to automatically detect the force interactions without any user input.

- We are currently unable to apply the CaPAR system to actions like *catch a ball* which include indeterminable parameters like the instant at which the ball is thrown and the duration of the ball's motion before it comes in contact with the agent's hand. This action also falls into the group of actions involving interaction between two agents which the system currently does not address. Extending the system to handle such interactions will be a very interesting and challenging project.

- The CaPAR system has been built to capture, record and analyze information from the actions of a human performer. A very important extension to this system would be to further enhance the derived motions with some emotions and manner using the EMOTE system [18].

- The principles we used would likely be found applicable to motion captured data by other means such as by video cameras. The extension to CaPAR in the abstraction of 2D data to 3D constraints would be a challenging but feasible project.

Bibliography

[1] Kenji Amaya, Armin Bruderlin, and Tom Calvert. Emotion from motion. In *Graphics Interface*, pages 222–229, 1996.

[2] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *International Conference on Machine Learning*, 1997.

[3] Douglas Ayers and Mubarak Shah. Monitoring human behavior in an office environment. In *Fourth IEEE Workshop on Applications of Computer Vision, WACV'98*, pages 42–47, Los Altimos, CA, 1998.

[4] F. Azuola, N. I. Badler, P. Ho, I. A. Kakadiaris, D. Metaxas, and B.J. Ting. Building anthropometry-based virtual human models. In *Proceedings of the IMAGE VII Society Conference*, Tucson, AZ, 1994.

[5] N. Badler, C. Phillips, and B. Webber. *Simulating Humans: Computer Graphics, Animation and Control.* Oxford University Press, New York, NY, 1993.

[6] N. Badler, B. Webber, M. Palmer, T. Noma, M. Stone, J. Rosenzweig, S. Chopra, K. Stanley, J. Bourne, and B. Di Eugenio. Final report to Air Force HRGA regarding feasibility of natural language text generation from task networks for use in automatic generation of Technical Orders from DEPTH simulations. Technical report, CIS, University of Pennsylvania, 1997.

[7] Norman Badler. *Temporal Scene Analysis: Conceptual descriptions of object movements.* PhD thesis, CS, University of Toronto, 1975.

[8] Norman Badler, Rama Bindiganavale, Jan Allbeck, William Schuler, Liwei Zhao, and

Martha Palmer. Parameterized action representation for virtual human agents. In Justine Cassell et al., editors, *Embodied Conversational Agents.* MIT Press, 2000.

[9] Norman Badler, Michael Hollick, and John Granieri. Real-time control of a virtual human using minimal sensors. *Presence*, 2(1):82–86, 1993.

[10] Selim Balcisoy and Daniel Thalmann. Interaction between real and virtual humans in augmented reality. In *Proc. Computer Animation'97*, pages 31–38, Geneva, 1997. IEEE, IEEE CS Press.

[11] David Baraff and Andrew Witkin. Physically based modeling. In *SIGGRAPH 98 Course Notes.* ACM SIGGRAPH, 1998.

[12] Ferdinand P. Beer and E. Russell Johnston Jr. *Vector Mechanics for Engineers: Dynamics.* McGraw-Hill, sixth edition, 1997.

[13] Joshua Bers. A body model server for human motion capture and representation. *Presence*, 5(4):381–392, 1996.

[14] Aaron Bobick. Movement, activity and action: The role of knowledge in the perception of motion. *Philosophical Transactions: Biological Sciences*, 352:1257–1265, 1997.

[15] Aaron F. Bobick, Stephen S, Intille, James W. Davis, Freedom Baird, and Claudio S. Pinhanez. The kids room: A perceptually-based interactive and immersive story environment. Technical Report 398, M.I.T. Media Laboratory Perceptual Computing Section, 1998.

[16] Juliet C. Bourne. *Generating Effective natural language instructions based on agent expertise.* PhD thesis, CIS, UNiversity of Pennsylvania, 1999.

[17] Armin Bruderlin and Lance Williams. Motion signal processing. In *Computer Graphics Proceedings*, pages 97–104. ACM SIGGRAPH, 1995.

[18] Diane Chi, Monica Costa, Liwei Zhao, and Norman I. Badler. The EMOTE model for Effort and Shape. In *Computer Graphics Proceedings.* ACM SIGGRAPH, 2000.

[19] Kwang-Jin Choi and Hyeong-Seok Ko. On-line motion retargetting. In *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications*, pages 32–42, Los Alamitos, CA, 1999. IEEE.

[20] Kwang-Jin Choi, Sang-Hyun PArk, and Hyeong-Seok Ko. Processing motion capture data to achieve positional accuracy. *Graphics Models and Image Processing*, 61(5):260–273, September 1999.

[21] Sonu Chopra. Where to look? automating some visual attending behaviors of human characters. *International Journal on Agents and Multiagent Systems*, 2000. to appear.

[22] Michael Cohen. Interactive spacetime control for animation. In *Computer Graphics Proceedings*, volume 26, pages 294–302. ACM SIGGRAPH, 1992.

[23] James W. Davis and Aaron F. Bobick. The representation and recognition of action using temporal templates. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 928–934, Puerto Rico, 1997.

[24] James William Davis. Appearance-based motion recognition of human actions. Technical Report 387, M.I.T. Media Lab Perceptual Computing Group, 1996.

[25] Brett Douville, Libby Levison, and Norman Badler. Task level object grasping for simulated agents. *Presence*, 5(4):416–430, 1996.

[26] Luc Emering, Ronan Boulic, and Daniel Thalmann. Conferring human action recognition skills to life-like agents. *Applied Artificial Intelligence*, 13(4–5):539–565, June–August 1999.

[27] Christiane Fellbaum, editor. *WordNet, An Electronic Lexical Database.* MIT Press, 1998.

[28] E. G. Gilbert, D.W. Johnson, and S.S Kerthi. A fast procedure for computing the distance between objects in three dimensional space. *IEEE Journal on Robotics and Automation*, RA-4:193–203, 1988.

[29] Michael Gleicher. Motion editing with spacetime constraints. In *Symposium on Interactive 3D Graphics*, pages 139–148. ACM, 1997.

[30] Michael Gleicher. Retargetting motion to new characters. In *Computer Graphics Proceedings*, pages 33–42. ACM SIGGRAPH, 1998.

[31] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation. *The Journal of Visualization and Computer Animation*, 9(2):65–94, 1998.

[32] John P. Granieri, Welton Becket, Barry D. Reich, Jonathan Crabtree, and Norman I. Badler. Behavioral control for real-time simulated human agents. In *Symposium on Interactive 3D Graphics*, pages 173–180. ACM Press, April 9-12 1995.

[33] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *Computer Graphics Proceedings*, pages 153–162. ACM SIGGRAPH, 1997.

[34] Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. V-COLLIDE: Accelerated collision detection for VRML. In *Proceedings of VRML*, 1997.

[35] Katsushi Ikeuchi and Takashi Suehiro. Toward an assembly plan from observation. part 1: Task recognition with polyhedral objects. *IEEE Transactions on Robotics and Automation*, 10(3):368–385, June 1994.

[36] M. Kallmann and Daniel Thalmann. A behavioral interface to simulate agent-object interactions in real-time. In *Proceedings of Computer Animation*, pages 138–146, Los Alamitos, CA, 1999. IEEE Computer Society.

[37] Yasuo Kuniyoshi and Hirochika Inoue. Qualitative recognition of ongoing human action sequences. In *Proceedings of the thirteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1600–1609, San Mateo, California, 1993. IJCAI, Kaufmann Publishers.

[38] Beth Levin. *English Verb Classes and Alternations: A Preliminary Investigation.* The University of Chicago Press, 1993.

[39] Richard Mann and Allan Jepson. Towards the computational perception of action.

In *Processdings, Computer Vision and Pattern Recognition*, pages 794–799, Santa Barbara, CA, 1998.

[40] Richard Mann, Allan Jepson, and Jeffrey Mark Siskind. Computational perception of scene dynamics. *Computer Vision and Image Understanding*, 65(2):113–128, February 1997.

[41] Alberto Menache. *Understanding Motion Capture for Computer Animation and Video Games.* Morgan Kaufmann Publishers, 1999.

[42] Kenneth Meyer and Hugh L. Applewhite. A survey of position trackers. *Presence*, 1(2):173–200, 1992.

[43] A.E. Michotte, E. Miles, and T.R.Miles. *The perception of causality.* Basic Books, New York, 1963.

[44] George A. Miller. English verbs of motion: A case study in semantics and lexical memory. In Arthur W. Melton and Edwin Martin, editors, *Coding Processes in Human Memory*, chapter 14, pages 335–372. John Wiley and Sons, 1972.

[45] George A. Miller and Philip N. Johnson-Laird. *Language and Perception.* The Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1976.

[46] Tom Molet, Ronan Boulic, and Daniel Thalmann. A real time anatomical converter for human motion capture. In *Eurographics Workshop on Computer Animation and Simulation*, pages 79–94, 1996.

[47] Vishvjit S. Nalwa. *A Guided Tour of Computer Vision.* Addison-Wesley Publishing Company, 1993.

[48] S. Narayanan. Talking the talk is like walking the walk. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society*, pages 548–553, Palo Alto, Calif. Mahwah, NJ, 1997. Lawrence Erlbaum and Associates.

[49] Darren Newston, Gretchen Engquist, and Joyce Bois. The objective basis of behavior units. *Journal of Personality and Social Psychology*, 35(12):847–862, 1977.

[50] Mark A. Nixon, Bruce C. McCallum, W. Richard Fright, and N. Brent Price. The effects of metals and interfering fields on electromagnetic trackers. *Presence*, 7(2):204–218, April 1998.

[51] Martha Palmer, Joseph Rosenzweig, and William Schuler. Capturing motion verb generalizations with synchronous tag. In P. St. Dizier, editor, *Predicative Forms in NLP. Text, Speech, and Language Technology Series*, pages 250–277. Muwer Press, Dordrecht, The Netherlands, 1998.

[52] Python language, 1998. http://www.python.org.

[53] Jennifer L. Romack. Information in visual event perception and its use in observational learning. *Studies in Perception and Action III*, pages 289–292, 1995.

[54] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael Cohen. Efficient generation of motion transitions using spacetime constraints. In *Computer Graphics Proceedings*, volume 30, pages 147–154. ACM SIGGRAPH, 1996.

[55] Stefan Schaal. Learning from demonstration. In M.C. Mozer, M. Jordan, and T. Petsche (eds.), editors, *Advances in Neural Information Processing Systems 9*, pages 1040–1046. Cambridge, MA: MIT Press, 1997.

[56] Sudhanshu Semwal, Ron Hightower, and Sharon Stansfield. Closed form and geometric algorithms for real-time control of an avatar. In *Proceedings of VRAIS*, pages 177–184. IEEE Computer Society, 1996.

[57] Sudhanshu Semwal, Ron Hightower, and Sharon Stansfield. Mapping algorithms for real-time control of an avatar using eight sensors. *Presence*, 7(1):1–21, February 1998.

[58] Ken Shoemake. Animating rotation with quaternion curves. In *Computer Graphics Proceedings*, pages 245–254. ACM SIGGRAPH, 1985.

[59] Jeffrey Mark Siskind and Quaid Morris. A maximum-likelihood approach to visual event classification. In *Proceedings of the European Conference on Computer Vision*, pages 347–360. Springer Verlag, April 1996.

[60] Robert H. Thibadeau. Artificial perception of actions. *Cognitive Science*, 10(2):117–149, 1986.

[61] Deepak Tolani, Ambarish Goswami, and Norman I. Badler. Real-time inverse kinematics techniques for anthropomorphic models. *Graphical Models*, 2000. To appear.

[62] Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. In *Computer Graphics Proceedings*, pages 91–96. ACM SIGGRAPH, 1995.

[63] Douglas Wiley and James Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, pages 39–45, November/December 1997.

[64] Andrew Witkin and Zoran Popovic. Motion warping. In *Computer Graphics Proceedings*, pages 105–108. ACM SIGGRAPH, 1995.

[65] Dianna Xu and Norman Badler. Algorithms for generating motion trajectories described by prepositions. In *Computer Animation*, pages 33–39. IEEE Computer Graphics Society, May 2000.

[66] Y.A.Ivanov and Aaron F. Bobick. Action recognition using probabilistic parsing. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 196–202, Los Alatimos, CA, 1998.

[67] Jianmin Zhao and Norman Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, 1994.

[68] Xinmin Zhao and Norman I. Badler. Near real-time body awareness. *Computer Aided Design*, 26(12):861–868, December 1994.