Chapter 1
Action Capture: A VR-based Method for Character Animation

Bernhard Jung, Heni Ben Amor, Guido Heumer, and Arnd Vitzthum

Abstract This contribution describes a Virtual Reality (VR) based method for character animation that extends conventional motion capture by not only tracking an actor's movements but also his or her interactions with the objects of a virtual environment. Rather than merely replaying the actor's movements, the idea is that virtual characters learn to imitate the actor's goal-directed behavior while interacting with the virtual scene. Following Arbib's equation action = movement + goal we call this approach Action Capture. For this, the VR user's body movements are analyzed and transformed into a multi-layered action representation. Behavioral animation techniques are then applied to synthesize animations which closely resemble the demonstrated action sequences. As an advantage, captured actions can often be naturally applied to virtual characters of different sizes and body proportions, thus avoiding retargetting problems of motion capture.

1.1 Introduction

Motivation and Basic Idea

Supporting the analysis of the interaction between a user and a technical product in early phases of the product's design is an important application area for 3D computer graphics and Virtual Reality (VR). Consider the problem of evaluating the ergonomics of a virtual prototype, e.g. a car interior. One approach for analyzing the user-friendliness of the prototype's operation is the application of immersive Virtual Reality (VR) where a VR user performs various operation procedures on the prototype. An advantage of this approach is that the user interaction is highly natu-

VR and Multimedia Group
Institute of Informatics
TU Bergakademie Freiberg
http://vr.tu-freiberg.de

ral, as it involves the same movements as would be used on a real prototype or final product. However, a disadvantage of the approach results from the evaluation setup which relies on the subjective experience of a single or a few VR users only. The limited group of test users could be a cause that some crucial insights are missed in the analysis of the prototype.

In order to gain more general insights on the usability aspects of virtual prototypes, ergonomic analyses nowadays often make use of virtual humans. As an advantage, virtual humans can come in many sizes and body proportions to serve as arbitrarily large group of test persons. Further, with virtual humans, it is possible to repeat the simulated procedures many times. Ergonomic analyses can become much more objective this way. However, difficulties arise from specifying life-like animations of virtual humans in desktop settings. When animating complex, articulated 3D models such as virtual humans via desktop GUIs, subtle details of human movements might be missed. In consequence, the resulting ergonomic analyses might be rendered less meaningful as compared to animations based on tracking the user's movements in 3D space.

Our idea is to combine the advantages of the two approaches: First, a VR user simulates the operation of a virtual prototype using immersive VR technology, such as 6 DOF tracking devices and data gloves. Then, by exploiting the interaction protocols of the VR user's performance, animations of a variety of virtual humans repeating the demonstrated operating procedures are generated. We call this approach *action capture*. Action capture extends conventional motion capture as it not only records an actors movements in 3D space but also his or her interactions with the objects of a virtual environment.

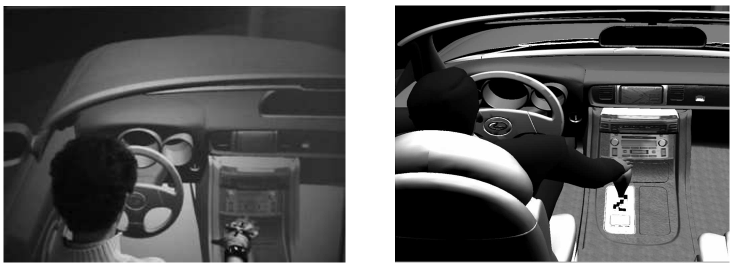


Fig. 1.1 Left: A VR user interacts with the virtual prototype of a car. Right: The user actions are repeated by a virtual character.

Challenges

The goal of action capture is to synthesize natural-looking animations of virtual humans from example interactions of a Virtual Reality user. While this idea may appear simple at first glance, its realization faces several non-trivial challenges:

1. Motion capture is not enough: Today, motion capture is a standard method for generating natural looking animations in games and movie productions. However, when applying recorded motion data to virtual characters of different body sizes, the resulting animations will be slightly different in each case. The problem becomes particularly evident when the animation involves interactions with virtual objects, i.e. the retargetting problem [Gle98]. Implementing action capture should instead comprise techniques for automatic animation retargetting. We tackle this challenge by employing procedural animation techniques that enable the virtual humans of displaying situationally adjustable, goal-directed behavior.
2. Inaccuracies of VR input devices: VR input devices such as position trackers and data gloves are sometimes hard to calibrate. Even when sufficiently calibrated, the delivered sensor readings often do not perfectly match the user's body and hand movements. One way of coping with these slight, yet possibly troublesome inaccuracies of VR input devices is to abstract from raw motion data and to represent actions at a higher level instead. For example, instead of resynthesizing hand shapes during grasping from recorded joint angles directly, we first classify hand shapes w.r.t. a grasp taxononmy and then animate the grasp from this symbolic description. In doing so, we can also optimize contact conditions between the virtual human's hand and the virtual object.
3. Unnaturalness of the user interaction in VR: Due to the slight inaccuracies of VR input devices and, even more important, the lack of convincing haptic feedback in typical immersive VR settings, VR users often interact with virtual objects in a somewhat cautious or even unnatural manner. When animating virtual humans, we do not want to replicate jitters in the VR user's movements while performing operating procedures on a virtual prototype. Instead, the resulting animations should appear natural and life-like. This challenge can be tackled e.g. by training the system with statistical models of natural reach motions and hand shapes. Animation synthesis is then a mixed data- and model-driven process to ensure that the generated animations are both goal-directed and natural-looking, quite possibly exceeding the original interactions of the VR user in these respects.

Background: Imitation Learning

Action capture is a method for synthezing animations of virtual humans from interactions of a human VR user in a virtual environment. To put this in in slightly different, more anthropomorphic terms: Action capture is a method that equips virtual humans with the ability to *imitate*[1] the behavior of a VR user.

Learning by imitation is a powerful ability of humans (and higher animals), which enables them to acquire new skills by observing actions of others. An instructive distinction between different stages of imitiative abilities during child development is proposed by Meltzoff and coworkers (see [Mel96] and [RSM04]). After the initial body babbling phase, the imitative abilities progress as follows:

[1] According to Thorndike [Tho98] imitation is: "from an act witnessed learn to do an act."

1. **Imitation of Body Movements:** The infant uses its body parts to imitate observed body movements or facial acts.
2. **Imitation of Actions on Objects:** Later, infants learn to imitate the manipulation of objects which are external to their body.
3. **Inferring Intentions:** In an even higher form of imitative learning, a demonstrator's goals and intentions are inferred from his observed behavior. In such a case, even an unsuccessful act can be correctly imitated.

The name 'action capture' owes to this distinction which is also expressed in the formula of neuroscientist M. Arbib: action = movement + goal, i.e. actions are always associated with a target object [Arb02]. Thus, whereas motion capture serves to reproduce an actor's body movements, action capture aims to reproduce a VR-user's actions on objects in the virtual environment. The implementation of imitation at the level of intentions could possibly be achieved by the application of Artificial Intelligence techniques but is beyond the action capture framework presented here.

Recent years have seen a growing interest in technical implementations of imitation learning, mainly in the field of robotics as a method of 'Programming by Demonstration (PbD)'; the edited collections of Dautenhahn & Nehaniv [DN02] and Billard & Siegwart [BS04] provide general overviews. Technical implementations of imitation learning generally provide solutions for three subtasks (cf. [BK96]):

1. **Observation:** The demonstrator's actions are observed, segmented, and abstracted into suitable action primitives.
2. **Representation:** The actions are represented through an internal model.
3. **Reproduction:** Based on the internal model, the actions are adapted to the current situation to reproduce an appropriate variant of the actions.

As VR-based instantiation of imitation learning, the technical realization of action capture implements these subtasks, as will be described below.

Action Capture: The Basic Method

Action capture aims to take advantage of increasingly available, complete VR systems for the purpose of virtual character animation. Similar to motion capture, the user's movements are recorded by means of position trackers and data-gloves. However, not only the user's movements are tracked but also his or her interactions with scene objects. User movements and interactions are then abstracted to higher level action representations. Each action is described in terms of an action primitive and the scene objects involved. In the playback phase, these action sequences are reproduced by virtual characters using behavioral animation techniques (cf. e.g. [Tom05]. By resynthesizing complete actions on objects rather than mere movements, valid animations can be reproduced for virtual characters of different sizes and body proportions as well as in situations where the task environment slighty differs from the original recording situation, e.g. in the case of repositioned control elements.

Overview of this contribution

Section 1.2 presents a general framework for action capture including its setting and a system architecture for its implementation. Also, we briefly describe our approach adding interaction capabilties to virtual worlds, including a method for realitic virtual grasping. Section 1.3 describes techniques for analyzing the VR user's movements and manipulations of scene objects. In Section 1.4, we introduce an XML-based format for representing action sequences extracted from user interactions. Section 1.5 describes a method for generating animations of virtual characters from action representations using behavioral animation techniques. Finally, we discuss the proposed action capture method in Section 1.6.

1.2 Action Capture Framework

Action capture is a VR-based method for recording the actions of a human VR user and later reproducing these actions by virtual characters. For the present purposes, with 'actions' we refer to manipulations of scene objects. Actions are decomposable into action primitives which correspond to basic behaviors of the virtual characters.

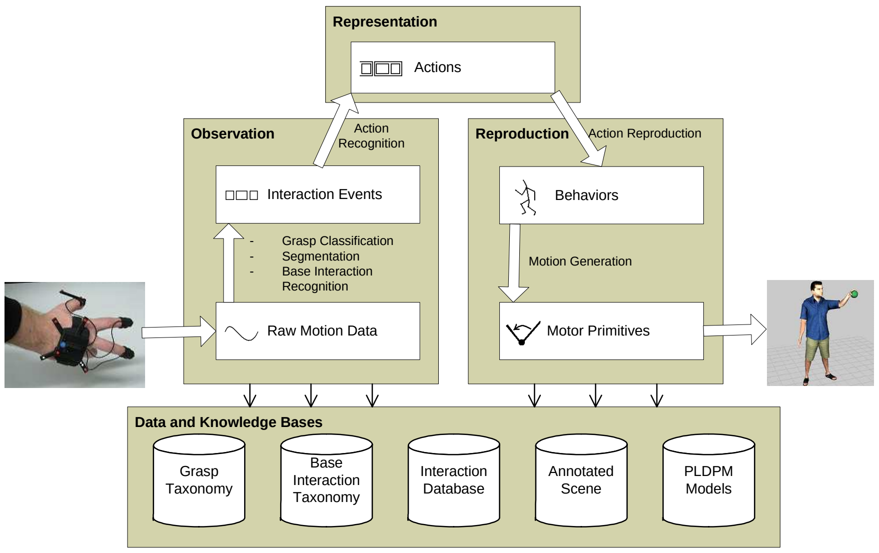


Fig. 1.2 Components of a System Archtecture for Action Capture

1.2.1 Action Capture Setting

The *setting* for action capture consists of:

- Virtual environment: which supports its interactive manipulation by a human user. I.e. the virtual environment contains interactive objects which e.g. can be picked up and displaced, buttons that can be pushed, knobs for turning etc.
- Human demonstrator (teacher): who performs an action or a sequence of actions in the virtual environment. The human teacher's actions are typically tracked using standard VR input devices such as position trackers and data gloves although in principle alternative methods e.g. based on visual input are also possible.
- Virtual character (learner): who observes the teacher's actions and learns to repeat them. The virtual character's body is assumed to be similar to the teacher's body, i.e. humanoid. This assumption ensures a more or less straightforward mapping of the teacher's body parts to the virtual character's body, thus simplifying the solution to the correspondence problem. The virtual character's body size and proportions may however differ from the human VR user.

1.2.2 A prototypical Implementation of Action Capture

The action capture concept presented in the preceding section has been implemented in a prototypical system. Figure 1.2 illustrates the main functional components of the system which are:

1. Action observation and analysis: during which the teacher's movements and interactions with scene objects are tracked, segmented, and classified as action primitives (basic interactions).
2. Action representation: Observed action primitives are combined and stored as high-level representations of the action or action sequence. Actions are represented in symbolic form and are thus amenable to manual postprocessing by a human editor. Action representations may also contain style information.
3. Action reproduction: where the action's representation is mapped to goal-directed behaviors of the virtual character. Behaviors are responsible for calculating contact conditions between the virtual character's hand and scene objects for the animation of object manipulaitions. They are executed by calling the lower-level motor programs which serve to animate the virtual character.

All components in the architecture refer to several common data and knowledge bases, including a grasp taxonomy, interaction databases, semantic object annotations, statistical motion models etc. These main components of the system architecture are described in more detail in the following sections.

Annotated Object Model

An important prerequisite for an Action Capture-ready Virtual Environment are objects that actions can be performed on. The kind of actions referred to here includes amongst others 6DOF displacement (pick and place) and manipulations on control actuators such as pushing/pulling levers, pressing buttons, turning knobs, moving sliders, etc. This exceeds the possibilites that a purely graphical representation of objects offers (in the sense of classical 3D models) by far. To implement the functionalities of these objects, a host of functional components need to be integrated into the VR application. Besides graphical rendering there is a need for collision detection, dynamics simulation, sound generation, etc. Each of these functional componentes often has its own database with possibly incompatible object representations, such as scenegraph vs. flat object collections of dynamics engines.

To facilitate the design and management of virtual reality scenes and to provide a mechanism for declaration of higher-level information for scene objects, the concept of *annotated objects* has been introduced. This XML-based representation structure incorporates all information about types of scene objects in a common database. Such information includes graphical model, type name, component references, grasp affordance information, physical parameters, collision proxies, joint definitions, etc. A central annotated objects management component handles the instantiation and destruction of objects and provides each functional component of the VR application with the information relevant to its specific functionality.

In addition to rigid bodies with physical properties, a system for articulated objects has been implemented. Such objects normally consist of a fixed fitting and one or more actuator components. The actuators are attached to the fitting by joints with varying degrees of freedom, joint constraints and with the support for discrete states. The actuators' behavior is fully simulated by a dynamics engine and thus reacts realistically to forces exerted by the user's virtual hand model as well as to enviroment influences such as object-object collisions, gravity, etc. This object model forms a solid and versatile basis for direct user interactions in realistic virtual prototyping scenarios.

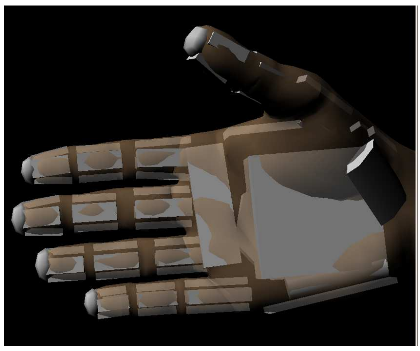
Realistic Virtual Grasping

A central functional component for VR applications involving direct manual object manipulation is the virtual hand model of the user. It forms the bridge between the real world and the virtual scene without which there would be no adequate way for the user to manually interact with scene objects. The virtual hand model has several functions to fulfill:

- *Represent the real hands of the user as accurately as possible.* This is done by employing a skeletal model with bone lengths adjusted to the respective bone lengths of the VR user's real hands. The joint rotations of the VR user's fingers are detected with various tracking devices, e.g. Immersion Cybergloves or A.R.T. Fingertrackers, and mapped to the joint angles of the hand model. The wrist positions, i.e. the root positions of the hands, of the user are tracked via 6DOF trackers and determines translation and orientation of the hand model in the virtual environment accordingly.

- *Detect collisions between the virtual hand model's fingers and the virtual objects.* For this purpose we employ collision sensors which are attached to key locations on the bones of the virtual hand model. These sensors consist of geometrical primitives (spheres, boxes, cylinders etc.) and thus facilitate efficient collision detection. Each sensor is assigned to a specific position on a specific finger or palm segment. Thus, when a collision of a sensor and a scene object occurs it is precisely clear which part of the hand model touched the object. This provides further cues for grasp classification, c.f. section 1.3.2, and drives the grasping heuristics. The grasping heuristics determines when an object has been completely grasped by the user and thus is attached to the user hand movements. The release of objects is determined similarly.
- *Determine the outcome of hand-object collisions.* In grasped state, the grasped object just follows the hand motions. For all other hand-object collisions, forces are determined based on contact points, contact normals and intersection depth of the collision. These forces are applied to the object and allow slight manipulations of objects even in ungrasped state, such as pushing, or manipulations of the articulated control actuators.

For the description of the bone structure of the hand model we use the Cal3D format and a model that adheres to the HANIM standard for humanoid models. For the description of the collision sensors and their placement on the hand model a custom XML formalism has been developed. Figure 1.2.2 shows a screenshot of the hand model with sensors and an excerpt of the XML sensor definition file.



```
<sensor name="right-thumb-sensor3-volar">
  <bone>r_thumb3</bone>
  <rel-translation>0.0 0.0 -1.25</rel-translation>
  <in-between-next-bone>r_thumb_distal_tip
  </in-between-next-bone>
  <geometry>
      <primitive>cube</primitive>
      <scale>3.0 1.5 1.5</scale>
      <rotation-angles>0 -10 0</rotation-angles>
  </geometry>
  <sensor-segment-info>
      <segment>thumb_distal</segment>
      <location>volar</location>
  </sensor-segment-info>
</sensor>

<sensor name="right-hand-back">
  <bone>r_middle0</bone>
  <rel-translation>8.0 -2.0 1.8</rel-translation>
  <geometry>
      <primitive>cylinder</primitive>
      <scale>5.0 5.0 0.6</scale>
```

Fig. 1.3 left: Virtual hand model with collision sensors. right: Example sensor definition in XML.

1.3 Observation: Interaction Analysis

The analysis process of user interactions works through several levels of abstraction. It starts with the raw data collected from the input devices tracking the user. From this raw data atomic basic interactions are extracted by a segmentation and classification process. Information about these basic interactions is passed on in the form of *interaction events* to higher levels of the application, such as the action recognition component. This component detects semantically meaningful actions from the user interactions with scene objects and represents them in a high-level description formalism which in turn links back to the lower data layers for reference. This action representation can be used for persistence and serves as the central interface for exchange between the observation and the action reproduction component.

1.3.1 Motion Level

On this level, data from the various tracking hardware is collected in a continuous fashion. In a typical scenario of the Virtual Workers project, the user is head tracked via stereo vision goggles with markers. The arms are tracked at several key locations, such as shoulders, elbows and wrists through 6DOF-markers. And the finger movements are tracked through either Cyberglove or optical fingertracking systems. From all the collected data, a virtual representation of the user's posture over time is generated (currently limited to the upper body), therefore this level of analysis can also be referred to as the *motion capture* level.

All input devices together produce a continuous and extensive stream of heterogeneous data. To enable persistence and to facilitate the recording and playback process, an interaction database module has been implemented. This database collects all data created in the motion capture process in the form of various channels and stores it in a central datastore. All data is explicitly assigned to its specific recording session. These recording sessions can further be annotated with meta information, such as the interacting user, the virtual scene, optional video footage, etc. This allows for reproduction of individual recording sessions as well as analysis on a larger scale, across session boundaries, such as training data collection for classification algorithms, principal component analysis, etc.

1.3.2 Basic Interaction Level

The goal of this level is to detect a specified set of basic interactions from the continuous stream of data coming from the motion level. All basic interactions have one specific aspect that is modified by the respective type of interaction, such as hand-object distance, hand-object contact, forces, prehension, and object position or orientation. Currently, the following types of basic interactions are distinguished:

- **reach** - The movement of the user hand towards a scene object. This can be along a relatively straight line as well as via a complex approach trajectory. The modified aspect is the distance between the user hand and the object and the outcome is that the user hand is able to touch the object.
- **grasp** - Refers to the full prehensile enclosure of the object by the user hand. Grasping can happen by various different grasp types which are detected through a classification process w.r.t. a given grasp taxonomy. The modified aspect is the contact between the user hand and the object with the outcome that the user firmly holds the object and is able to move it to another location, or, in the case of actuators, manipulate the actuator components according to its degrees of freedom.
- **touch** - The same as grasp, but does not result in object prehension. Also modifies contact between the user hand and the object with the outcome that a light non-prehensile contact has been established that allows, e.g., pushing.
- **release** - The counterpart to *grasp*. Also modifies hand-object contact with the outcome that the object is released from prehension by the user hand. It is then again subject to other influential factors, such as gravity or reset forces.
- **push** - This includes any exertion of forces on the object in a non-grasped state. The outcome of the forces depends on the nature of the object. In the case of a freely movable rigid body it normally leads to a position change. In the case of articulated objects it depends on the specific degrees of freedom of the actuator. Normally it leads to a translation of the actuator component along one of the actuator axes without moving its fitting. The pressing of a button would be a classic example.
- **pull** - This is similar to push but requires object prehension and normally is directed opposite to the shoulder-hand vector. The pulling of a lever or the opening of a drawer is a typical example.
- **displace** - Refers to the movement of the object as a whole and is thus restricted to freely movable rigid bodies. Also requires object prehension. The modified aspect is the object's location and orientation in space.
- **turn** - This is a special case of pulling or pushing, restricted to control actuators with rotational degrees of freedom. The modified aspect is the orientation of the actuator component relative to its fitting.

A first step in the detection of basic interactions is the segmentation of hand movement data. Hand trajectories are recorded and segmented when pauses in movement occur. These pauses normally denote the end of one interaction and the possible beginning of a new one. Another step is the reaction to collision information from the hand sensors. Whenever a collision of a hand sensor with an annotated object has been detected, contact has been made with the object and possibly a prehensile grasp has occured. A grasp heuristics is applied to determine this, based on which points of the hand make contact with the object. For example, contact with the thumb and the index finger at their volar aspects are a strong indicator for a prehensile grasp. When a prehensile grasp occurs, its grasp type is classified based on the hand posture information from the tracking devices; see [HBAJ08] or [HBAWJ07] for details on the classification process.

Further, forces are generated based on the intersection depth of the hand sensors with the objects in direction of their contact normals. These forces can lead to either pushes, pulls or turns, depending on the type of contact and the type of object involved. Displacement occurs when the whole hand is moved while an object is grasped.

Another important functionality is to make the information about detected basic interactions available to other components of the application in a flexible way. For this reason, the concept of *interaction events* has been introduced. These events are posted via a dispatcher/subscriber system (observer pattern). Any system component can send interaction events to the dispatcher, e.g. when one of the basic interactions has been detected. The dispatcher in turn passes the events on to the registered listener components which can react accordingly based on their respective functionality. This way, senders and receivers of interaction events need not know of each other in software engineering terms. Example functionalities for event receivers are visualization, persistence (to file or to database) and most importantly action recognition.

Interaction events have a type, based on the basic interaction encapsulated. In terms of structure, interaction events consist of a type independent header and type dependent contents. The header contains type identifier and timing information (timestamp and duration). The type dependent part contains details about the basic interactions described in the event. A *grasp* event, for instance, contains hand configuration, hand position, an identifier of the grasped object, contact points between hand and object, detected grasp type, etc. A *reach* event contains the hand motion trajectory, end position, etc. In addition to hand-related events, control actuators send information about changes of their internal state. These can be of varying degree of detail and reach from just the final state via discrete intermediate states to a complete state history for every frame. The latter allows for an exact in-effect reproduction of the performed manipulation even when the playback component does not have a dynamics simulation.

For persistence and to allow for manual analysis, an XML format for interaction events has been developed. This format can be stored and retrieved to file or database and contains all details in human-readable form. See figure 1.4 for an example *grasp* event.

1.4 Representation of Actions

One possibility to animate a virtual human would be to use interaction events generated from raw tracking data directly as animation input. However, especially for testing purposes it should also be possible to author an animation description quickly with an appropriate editor, e. g. an XML editor. A basis interaction description would not be a good choice to solve this task since it tends to become too complex to be human-readable for long interaction event sequences. For this reason, an abstraction from the interaction event level is required. We provide this abstraction

```
<event timestamp="9.7555" type="grasp" duration="0.92">
<low-level>
<sensor-data numsensors="22">124 116 146 93 150 175 111 151 140 125
178 105 145 89 174 87 163 114 95 85 148 61</sensor-data>
<joint-angle joint-id="r_index1">73.27 -0.04 0.99 0.06</joint-angle>
...
<joint-angle joint-id="r_thumb3">42.75 0 1 0</joint-angle>
<object-ids>cockpit_steering_wheel-1</object-ids>
<hand-transform>0.720147 -0.39539 -0.570134 0
                0.616856 0.741028 0.265258 0
                0.317605 -0.542716 0.77755 0
               -0.228399 -0.400599 1.2676 1</hand-transform>
<hand-side>right</hand-side>
</low-level>
<high-level>
<taxonomy>Schlesinger</taxonomy>
<category>Cylindrical</category>
</high-level>
</event>
```

Fig. 1.4 Example XML representation of an interaction event (grasp).

in the form of an action description language. Moreover, such a language offers the possibility to easily rework or change recorded interaction event sequences after transforming them (automatically) into a high-level action description.

Regarding the fact that we want to automatically generate natural-looking animations from an action description, the inclusion of references to underlying interaction event sequences should be allowed (see section 1.3). In this way, basic interaction data can also be used by an animation synthesis tool. However, the description must retain the power to enable the derivation of plausible animations even if links to the underlying interaction description layer are not included.

The specification of the action format includes different aspects such as (manipulated) objects, different action types, action composition, synchronization and timing. The action description language is defined by an XML-Schema. In the following, single aspects of our action description language will be explained in more detail.

Objects

Objects can be divided into several classes: *fixed objects*, *movable objects* and *articulated objects* (e.g. control actuators). Fixed objects cannot be moved. However, they can be touched. Movable objects can be moved arbitrarily (e.g. a ball). Articulated objects have specific movement constraints. For example, a *slidable object* is an articulated object which can only be moved along one axis and has a maximum and a minimum position. Another special kind of an articulated object is a so called *discrete state object* which represents an articulated object with defined discrete states.

Properties, constraints and discrete states of objects (such as the minimum and maximum position of a slider or the *ON* and *OFF* states of a toggle button) can be defined in annotated object documents (see section 1.2.2) which are referenced by the corresponding objects. The XML code below shows two example object definitions.

```
<MovableObject id="Hammer" annotation="hammer.xso"/>
<DiscreteStateObject id="CarRadioOnOffButton" annotation="button.xso"/>
```

Actions

An *action* describes the interaction between the user hand and a particular object. Conceptually, an action consists of different phases: *reaching* (approaching the object), *grasping* or *touching*, *object manipulation* (optional) and *releasing* the object. Different action types can be distinguished: *constrained move actions*, *unconstrained move actions* and actions which don't result in an object displacement (*touch actions*). Some action types can be only performed using a special kind of object. For instance, the *constrained action* subtype *shift* can be only performed using a *slidable object* (a subtype of an *articulated object*). Unconstrained move actions can result in a change of the position and/or orientation of a *movable object*. Specialized *unconstrained move* action types are *translate*, *pick and place* and *turn*. The example code below illustrates the definition of a *pick and place* action.

```
<PickAndPlace
    targetPosition="5 5 2"  targetObject="Hammer"
    reachDuration="0.5"     interactionDuration="2"
    graspType="CYLINDRICAL"/>
```

As mentioned, actions can be derived from a sequence of interaction events. For instance, a *touch action* simply consists of a reach, a grasp or touch and a release event while a *pick and place* action comprises a reach and a grasp event, some displace events and a release event.

Action Composition

Actions can be grouped together using an *action unit*. An action unit contains a sequence of actions which are executed with a single hand or two hands cooperatively. In order to enable an appropriate description of action units, three different kinds of action units were predefined: *right hand*, *left hand* and *bimanual*. The term action unit was inspired by Kendon [Ken04] who analogously uses the term *gesture unit* to describe a sequence of gestures.

```
<RightHand startTime="2" relaxDuration="2">
    <Touch targetObject="CarRadioOnOffButton"
           reachDuration="1" interactionDuration="0.5"
           graspType="TIP"/>
    <Touch targetObject="CarRadioChannelSeekButton"
           reachDuration="0.5" interactionDuration="1"
           graspType="TIP"/>
</RightHand>
```

Fig. 1.5 Example XML representation of an action unit containing two consecutive actions.

Timing

Actions and action units are so called *time containers*. A time container has its own internal relative timeline. Properties related to timing of an action unit are *start time* and *relaxation duration*. The *start time* is the point of time when the first action of the unit starts after the unit was entered. All actions of the action unit are then executed consecutively in the order defined in the corresponding action language instance document. After the last action has completed, the action unit enters the relaxation phase. The *relaxation duration* of an action unit therefore describes the time required to return to the hand's relaxation position.

An action has two timing related attributes: *reach duration* and *interaction duration*. The *reach duration* represents the time required to position the hand on the target object (reaching phase). During the reaching phase the hand is also preshaped to perform a grasp. The reaching phase has finished if a stable grasp has been established. The grasp type can be defined explicitly by using the action property *grasp kind*. If no grasp type was specified, the animation player has to decide which grasp type can be applied in order to generate a plausible animation.

The *interaction duration* is the time required for the actual hand-object-interaction (interaction phase). To model the interaction process especially of constrained move actions more precisely, target object states and corresponding fractions of the interaction duration can be defined (e.g. moving a gear shift lever through different gears). The interaction phase ends with releasing the object.

A right hand type action unit containing a sequence of two *touch* actions is described by the XML example code in figure 1.5.

Synchronization

Synchronization aspects in our action description language were inspired by the Synchronized Multimedia Integration Language (SMIL [Wor08]). Several action units can be grouped together using an *action unit composite*. Like actions and action units, action unit composites are also time containers.

There are two different types of action unit composites: *parallel* and *sequential*. In contrast to the execution of action units contained in a sequential composite, in a parallel composite all action units are, just as the name says, executed in parallel. The processing of an action unit composite ends if the last (in sequential composites) or longest action unit (in parallel composites) is completed.

1.5 Reproduction of Actions

So far, we described how interactions of a real human are analysed and how the performed actions are stored in a XML-representation. For the intended application, it is vital that stored actions can also be replayed by virtual humans. This calls for animation synthesis algorithms which can generate convincing human-like animations. Synthesis algorithms need to take the environmental context into account in which a particular action in performed. For example, if the position of a button to be pressed has changed since recording the data, then, of course, the animation needs to be adapted to the new position of the button. The approach for animation synthesis in Action Capture also follows the imitation learning methodology introduced earlier. Grasp shapes, kinematic configurations or trajectories recorded from the human user are taken as input data to machine learning algorithms resulting in statistical models of postures and motions. The learning algorithms can be trained on-line using the data of the current user, or off-line by querying the data of various users from the interaction database. Models are later used to control the virtual humans by imitating the learned behavior. In the following we summarize the main results from [BAHJV08], [BADV$^+$08], [BAWHJ07].

1.5.1 Learning Behaviors with PLDPM

Creating a repertoire of motor skills for a virtual human is a challenging and often labour intensive task. Modern machine learning techniques can help to overcome this problem. In Action Capture, machine learning is used to extract important information about kinematic synergies and constraints of the human body, which are stored in a so-called *Probabilistic Low-Dimensional Posture Model* (PLDPM). Figure 1.6 shows an example of PLDPM-Learning for a grasp behavior. First, data about human grasping is acquired using an optical “fingertracking” system. The hand poses are stored as rotations of finger joints (3 ball joints per finger, i.e. 45 degrees of freedom in total). This data is then processed by a manifold learning technique, such as PCA, ISOMAP [TdSL00] or LLE [RS00] in order to get a low-dimensional subspace representing the recorded grasps. Manifold learning refers to a set of dimensionality reduction techniques that can project high-dimensional data onto low-dimensional manifolds. This is particularly helpful when working with human postures, due to the high number of degrees of freedom and the interdependency between joints. Each point in a low-dimensional manifold represents a human grasp and can, hence, be projected back into the original space of joint rotations. Notice, that a finite set of demonstrated postures allows us to extract a continous space with an unlimited number of possible interpolations and extrapolations. In addition to dimensionality reduction, we need a model of the anatomical constraints of the human hand. Such a model is needed in order to discriminate between anatomically feasible and unfeasible postures. This can be achieved by learning a Gaussian Mixture Model (GMM) based on the projected grasps in the low-dimensional manifold. The GMM estimates the probability density function of the grasps. This function can later be used to determine the probability of a grasp being similiar to the demonstrated grasps. Grasps that have low probability are likely to be anatomically infeasible.

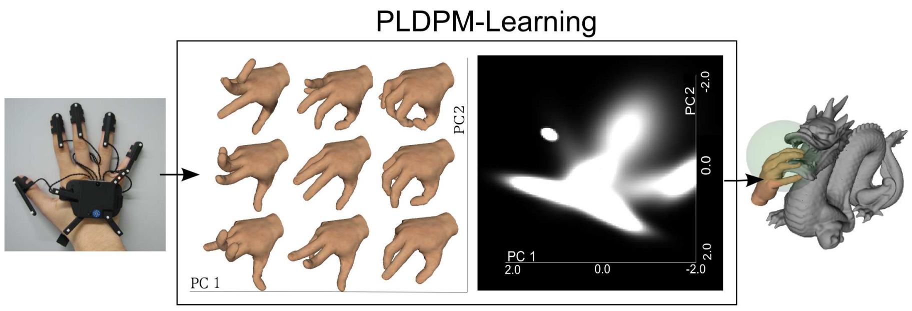


Fig. 1.6 Fingertracking data is used to learn a Probabilistic Low-Dimensional Posture Model for grasping. The learned model can later be used to synthesize realistic grasps for arbitrary 3D objects.

Learned PLDPMs are used to synthesize character postures according to the intended goal and by taking into account the environment. In the above example PLDPMs are used for grasp optimization. The goal of grasp optimization is to find a natural looking hand shape leading to a stable grasp on a user-provided 3D object. This is realized by searching for a point in the lower-dimensional posture model, which optimizes a provided *grasp metric*. Various metrics and quality measures for grasps have been proposed in the robotics literature [MF96], many of which are based on physical properties of the object and the performed grasp. In previous research, we showed that even simple metrics can produce realistic grasps [BAHJV08].

PLDPMs are not confined to the synthesis of postures, but can also synthesize full animations. For this, it is important to notice, that an animation corresponds to a trajectory in a PLDPM. Therefore, for synthesizing animations for virtual humans, we need to specify a trajectory in a PLDPM and project each point along the trajectory back into the original space of joint rotations. This procedure can, for instance, be used to dynamically synthesize a grasping motion for a new object. After a realistic grasp is optimized using the technique described above, a trajectory is created starting at the current position of the hand in the PLDPM and ending at the optimized position. Each point along this trajectory corresponds to a hand shape at a given time of the animation. The described approach can be used to synthesize a variety of complex multi-joint animations from learned examples. Crucial parts of this approach are the input data and the metric used for optimization. While the input data is typically some kind of motion capture, the optimization metric is a mathematical function describing the quality of a posture or animation with respect to the intended behavior.

1.5.2 Learning Goal-Directed Trajectories

Trajectories are important tools for the animation of virtual humans. For example, they can be used to represent the motion of the agent's wrist position during a reaching task. But how should the trajectory be changed, if the object the agent is trying to reach is displaced? Also, can we dynamically add slight changes to the trajectories, so they always look a little bit different and thus more lifelike ?

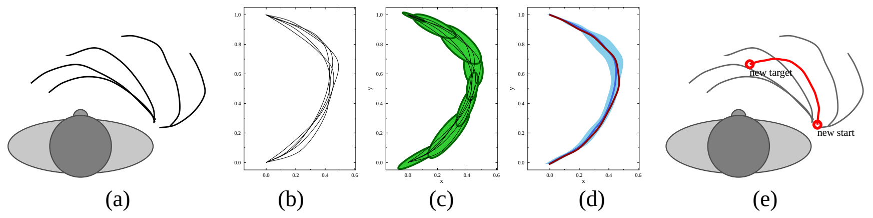


Fig. 1.7 (a) Trajectories in global-space recorded from a human test-subject. (b) Trajectories in local-space after coordinate system transformation. (c) A GMM is learned by fitting a set of Gaussians. (d) A GMR is learned and new trajectory is synthesized (red). The synthesized trajectory is retargeted based on the new start- and end-position.

A computationally efficient way to tackle this problem is the use of a dynamic coordinate system which is spanned between the hand of the virtual human and the position of the target. The idea is based on recent behavioral and neurophysiolgical findings which suggest that humans make use of different coordinate systems (CS) for planning and executing goal-directed behaviors, such as reaching for an object[HS98]. Although the nature of such CS transformations is not yet fully understood, there is empirical support for the critical role of eye-centered, shoulder-centered and hand-centered CS. These are used for transforming a sensory stimulus into motor commands (visuomotor transformations). For retargeting we use a hand-centered CS which is oriented towards the target object.

In Figure 1.7 we see the effect of transforming a set of global-space trajectories into a hand-object CS. The variance which is due to different goal positions of the reach motion was removed and the projected trajectories have higher similarity. The new space can be regarded as a end-position invariant space of trajectories. Next, a statistical model of the trajectories is learned. This is done using Gaussian Mixture

Regression (GMR). The learned GMR model can be queried for a new trajectory having similiar shape to the training trajectories. Finally, the synthesized trajectory can be retargeted to a new start- and end-position by applying a CS transformation from local- to global-space.

1.5.3 Generating Animations from Actions

For animating virtual humans, we need to translate the high-level actions into motions, as can be seen in Figure 1.2. This is done by translating each action into a sequence of behaviors from the virtual human's repertoire of skills. In turn, each behavior uses one or several motor primitives. Motor primitives are low-level programs, which modify the joint parameters of a virtual human. Basic behaviors include:

- **turn head** - Turns the head towards a given position or object.
- **follow trajectory** - Moves the wrist along a recorded or synthesized trajectory.
- **grasp close** - Grasps an object by closing the hand.
- **grasp open** - Opens the hand and brings it into an idle position.

In addition, secondary behaviors such as an 'idle motion' behavior are used, in order to increase the believability of the virtual human. All behaviors can be combined to create complex motions, where, for instance, the virtual human fixates the object and opens his hand while reaching for it. Motion trajectories and hand shapes are synthesized according to the environmental configuration. For example, if an object to be grasped has been displaced, a new reach-trajectory is synthesized using GMR, which takes the new position into account. Further, a new hand shape for grasping the object is optimized using a learned PLDPM. Figure 1.8 shows the reproduction of actions under different environmental configurations. During the recording of actions (left), a real user grasped the gear-shift, the steering wheel and later a radio button. The action reproduction subsystem translates these actions into a set of 'follow trajectory', 'grasp open' and 'grasp close' behaviors. The synthesized animations are robust against changes in the environment. As can be seen in Figure 1.8 (right), the recorded actions can be reproduced in VR, even if the position of the gear-shift and the body proportions of the virtual human change.

1.6 Conclusion

Action capture is a VR-based extension of motion capture that takes advantage of interactive virtual environments: Whereas traditional motion capture just aims to replay the body movements of an actor, action capture further aims to replicate the actor's interactions with the objects of a virtual environment. As an advantage, valid

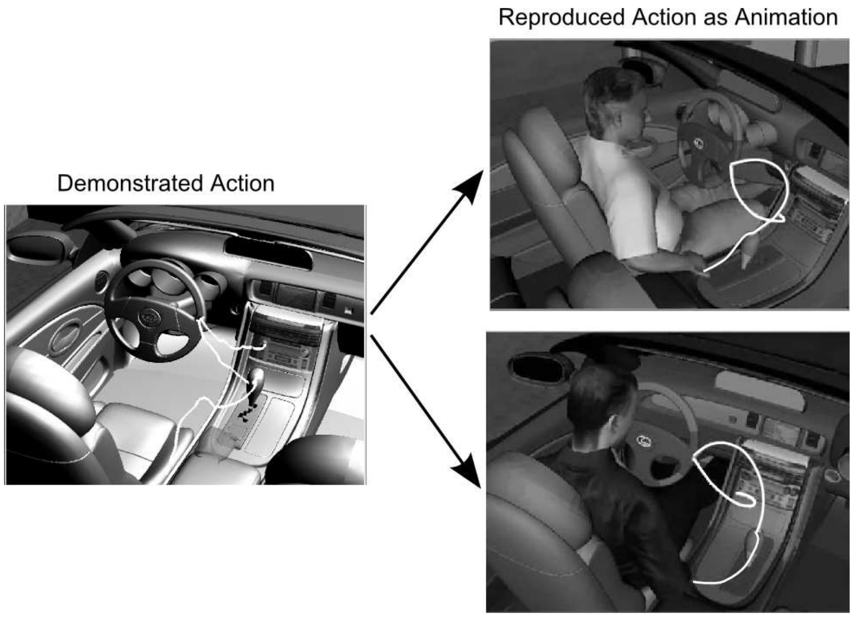


Fig. 1.8 Left: A user performs several actions in a cockpit scenario. The motion is shown as a white trajectory. Right: Recorded action files are used to animate virtual humans of different size and body proportions. Animations are robust against changes in the virtual prototype; e.g. to the right, the gear-shift has been repositioned. White lines indicate the trajectories of the right hand's wrist while interaction with several control elements in a car.

animations of interactions with scene objects are generated, even if the situation is changed.

A potential drawback of generating virtual human animations from recorded VR interactions results from the often overly cautious interactions in VR, e.g. due to missing haptic feedback. This problem is addressed in two ways: (a) During analysis of the interaction, observed movements are generalized to action representations that rely e.g. on hand shape classifications during grasps instead of joint angle recordings. And (b), during animation synthesis, pre-learnt statistical models of arm trajectories and hand shapes are applied and adapted to the current situation. From these models, goal-directed yet natural looking animations can be generated even if the original movement in VR is somewhat jittery.

Recorded actions can be reproduced using virtual humans of different sizes and body proportions. The resulting animations give us important insights about important aspects of a virtual prototype, such as design or ergonomy. We believe that action capture will prove particularly beneficial in virtual prototyping settings that require the automated generation of animations for many variants of prototypes and virtual humans.

The action capture method can be conceptualized as an application of imitation learning. In analogy to a distinction made in developmental psychology between the

the stages of movement and action imitation, action capture can be seen as a next stage in the synthesis of life-like character animations, accomplished by placing the actor in an interactive VR environment.

Acknowledgements

The research described in this contribution is partially supported by the DFG (Deutsche Forschungsgemeinschaft) in the Virtual Workers project.

References

[Arb02] M. A. Arbib. The mirror system, imitation, and the evolution of language. In Dautenhahn and Nehaniv [DN02].

[BADV$^+$08] H. Ben Amor, M. Deininger, A. Vitzthum, B. Jung, and G. Heumer. Example-based synthesis of goal-directed motion trajectories for virtual humans. In *5. Workshop Virtuelle und Erweiterte Realität*. GI-Fachgruppe VR/AR, 2008.

[BAHJV08] H. Ben Amor, G. Heumer, B. Jung, and A. Vitzthum. Grasp synthesis from low-dimensional probabilistic grasp models. *Computer Animation and Virtual Worlds*, 19, 2008.

[BAWHJ07] H. Ben Amor, M. Weber, G. Heumer, and B. Jung. Coordinate system transformations for imitation of goal-directed trajectories in virtual humans. In *Virtual Environments 2007. IPT EGVE 2007. 13th Eurographics Symposium on Virtual Environments. Short Papers and Posters*, 2007.

[BK96] P. Bakker and Y. Kuniyoshi. Robot see, Robot do: An Overview of Robot Imitation. In *AISB96 Workshop: Learning in Robots and Animals*, pages 3–11, 1996.

[BS04] A. Billard and R. Siegwart, editors. *Special Issue on Robot Learning from Demonstration*, volume 47 of *Robotics and Autonomous Systems*, 2004.

[DN02] K. Dautenhahn and C. Nehaniv, editors. *Imitation in Animals and Artifacts*. MIT Press, 2002.

[Gle98] M. Gleicher. Retargetting Motion to New Characters. In *SIGGRAPH'98 Conference Proceedings*, Computer Graphics Annual Conference Series, pages 33–42. ACM, 1998.

[HBAJ08] G. Heumer, H. Ben Amor, and B. Jung. Grasp recognition for uncalibrated data gloves: A machine learning approach. *Presence: Teleoperators* & *Virtual Environments*, 17(2):121–142, 2008.

[HBAWJ07] G. Heumer, H. Ben Amor, M. Weber, and B. Jung. Grasp Recognition with Uncalibrated Data Gloves - A Comparison of Classification Methods. In *Proceedings of IEEE Virtual Reality Conference, VR '07*, pages 19–26, March 2007.

[HS98] H. Heuer and J. Sangals. Task-dependent mixtures of coordinate systems in visuomotor transformations. *Experimental Brain Research*, 119(2), 1998.

[Ken04] Adam Kendon. *Gesture: Visible Action as Utterance*. Cambridge University Press, October 2004.

[Mel96] A. N. Meltzoff. The Human Infant as Imitative Generalist: A 20-year Progress Report on Infant Imitation with Implications for Comparative Psychology. In *Social Learning in Animals: The Roots of Culture*, pages 347–370, 1996.

[MF96] A. Moon and M. Farsi. Grasp Quality Measures in the Control of Dextrous Robot Hands. *Physical Modelling as a Basis for Control (Digest No: 1996/042), IEE Colloquium on*, pages 6/1–6/4, 1996.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[RSM04] R. Rao, A. P. Shon, and A. N. Meltzoff. A Bayesian Model of Imitation in Infants and Robots. In *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*, 2004.

[TdSL00] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[Tho98] E. L. Thorndike. Animal Intelligence: An Experimental Study of the Associative Processes in Animals. *Psychological Review Monographs*, 8, 1898.

[Tom05] B. Tomlinson. From Linear to Interactive Animation: How Autonomous Characters Change the Process and Product of Animating. *ACM Computers In Entertainment*, 3(1), 2005.

[Wor08] World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL 3.0)*, 2008.