Parametric Human Movements

Learning, Synthesis, Recognition, and Tracking

Herzog, Dennis

Publication date:
2011

Document Version
Early version, also known as pre-print

Parametric Human Movements

Learning, Synthesis, Recognition, and Tracking

PhD Thesis
by
Dennis Levin Herzog

Department of Mechanical and Manufacturing Engineering, Aalborg University
Lautrupvang 2B, 2750 Ballerup, Denmark
e-mail: deh@imi.aau.dk

Typeset in LaTeX, July 2011.

Abstract

This thesis aims at the learning of action primitives and their application on the perceptive side (tracking and action recognition) and the generative side (synthesizing movements for robot control). A major motivation is to use a unified primitive representation which represents the primitives in a context-dependent way and can be used for both, perception and generation. A motivation for a unified model comes from neuroscience: certain regions in the human brain are active during recognition and generation of certain actions. Suitable models in such a context are generative stochastic models such as the hidden Markov model (HMM). This thesis considers arm actions in a table-top scenario. The actions are for example: pointing to, reaching for, and relocating an object. These actions are highly context dependent, i.e. they depend on the actual locations of the objects. A promising extension of the commonly used HMM is the parametric HMM (PHMM), which has been introduced previously in the context of parametric gestures. The PHMM can learn and generalize the dependency of a movement trajectory on a set of parameters, as, for example, object locations. This capability is crucial in both cases: the recognition and the generation of arm movements. Besides the training of the PHMMs and the accuracy of this action representation, the following two applications are considered: imitation and intertwined tracking and recognition.

In the robotics imitation application, a humanoid robot is enabled to relocate objects placed on a table-top. An important aspect is to synthesize movements on the robot such that the robot reaches for the right object in order to grasp it. The learned parametric HMM enables the robot to generate an action for specific parameters given by the location of the object for which it should reach. Thus, the robot is able to achieve the desired effect by its actions as required in the current context. An interesting aspect of the imitation application is that the actions and their effects are learned on the basis of another embodiment, i.e. the PHMMs are trained on human performances. Both, the actions and their corresponding effects need to be mapped to the robot's embodiment.

In the tracking and recognition application, the action primitives are used to define a space of possible actions and action sequences, where the sequences are defined by a grammar. The tracking is performed in the action space which reduces the dimensionality of the tracking problem and allows for recognition. From the tracking perspective, it is crucial that the parameters of the action primitive can be used to adapt the primitive to the actual appearance of the tracked motion, since the actions visually appear different when applied to different object locations. From the recognition perspective, it is necessary to recognize that an action has been performed, but in order to understand the full semantic of an action, also the recovery of the action parameters is important. For example, in order to identify an object to which a person is pointing, it is necessary to identify the location to which the person is pointing. This is provided by the estimate of the action parameters.

Findings of the thesis are: a method is developed thus that PHMMs can be utilized to represent parametric actions/primitives not only for recognition but also for the synthesis of accurate trajectories. This is evaluated by experiments. The experiments with the humanoid robot show that the proposed methods of reproducing the actions on the robot and synthesizing actions from the parametric models (trained on demonstrations by a person) enable the robot to accomplish tasks by generating appropriate actions with the desired effect. In a rule-learning application the robot learns what to do with several objects. The implemented framework for recognition and tracking makes use of several action models, which are trained on demonstrations. The framework runs in real time and allows one to recover the pose and recognize actions and their parameters online. The experiments show that different actions can be distinguished and that the primitive actions of complex actions being tracked can be recognized even from a single view in an view invariant manner.

Acknowledgments

I would like to take this opportunity to thank all the people who have supported me during my PhD thesis. First, I want to thank my supervisor Assoc. Prof. Dr. Volker Krüger for all his guidance, fruitful discussions, advice and, especially, his patience over the past 4 years. I also thank my former college Prof. Daniel Grest for his support and collaboration in the beginning of my work. I appreciated very much the collaboration with Prof. Dr. Aleš Ude during my three month residence in Slovenia, where I could use a humanoid robot for research. I also thank my current and past group members Pradeep Kumar Reddy, Baby Sanmohan, Gaurav Sharma and Kasper Broegaard Simonsen for their company and discussions. Special thanks go to my parents, my Aunt Jane, and Uncle Zig for their moral support.

Table of Contents

Chapter 1

Introduction

Over the last decades, one can recognize an increasing interest of the vision, robotics, and computer graphics communities in human motion, actions and activities. Typical vision-based recognition applications for video footage are content-based video annotation, retrieval, summarization, or sign-language recognition. Further recognition applications are: face recognition, vision-based human-computer interaction, or surveillance, either in public, e.g., for theft detection, or at home for the surveillance of elderly people. The applications can be very specific, as, for example, the recognition of certain incidents as fall detection in the home, or complex activity recognition tasks as in comprehensive surveillance systems. On the other hand, the focus of video-based un-intrusive motion capture is mainly the acquisition of the body pose in a sequence of frames without the use of special suits or markers. One motivation is to capture the motion on the basis of usual cameras in unconstrained environments. Since motion capture is a complicated vision task, several novel works on motion capture make use of motion models. Further application of motion capture are human-computer interaction (HCI) and motion analysis. For HCI, motion capture may be used as a preprocessing step for a gesture recognition engine or to control an avatar in a virtual world. Typical application of motion analysis are clinical studies, assisted sports training, and diagnosis of orthopedic patients. The robotics community is interested in HCI to establish an easy way to program robots for certain tasks. An interesting field in robotics is the one of humanoid robots (humanoids). Here one is interested in communicating with a humanoid in a human-like way and enabling the robot to learn from demonstrations. Such a system would require advanced perceptive/cognitives skills to understand the action/activities of other agents, but also requires advanced planning and motor control skills to allow the robot to communicate with people and to perform complex tasks.

What makes human movements so important? The importance becomes obvious if one notices that human movements are ubiquitous in entertainment, communication, work, or at home. Most movies would not be the same without animated computer characters. People who communicate with each other usually support their utterances with gestures. Such gestures transfer often an important part of information of the dialog. Sometimes the dialogs can not be grasped without understanding the gestures. Such gestures are, for example, pointing gestures, which are used to specify an object, or gestures for communicating the size or length of an object. A nice example is the fish size gesture (see Figure 1.1), where a person tells about a fish he has caught: “The fish is that big”. However, this is only one aspect. Movements are of course also important to manipulate and interact with environment. Basic examples are actions such as grasping or pushing an object. These examples are rather simple.

Figure 1.1: Fish Size Gesture. The shown gesture communicates the size of a fish during a speech: "I cought a fish... it's that big". The idea of the drawing and speech stems from [134].

These actions can be understood in isolation. However, the actual meaning of those actions becomes by far harder to grasp if one increases the complexity of a scene. This is, for example, the case when a number of people interact or manipulate a set of objects over a long time, e.g., cleaning a dinner table or repairing a car. It is obvious that the perception and understanding of such activities or tasks is complicated, but this is also true for the execution of such tasks. The humans have to control their embodiment and have to plan and coordinate their activities.

So what makes the applications of computer vision and the tasks of a humanoid robotics so different? An important aspect is that the perception and understanding can be performed on different levels. In applications such as surveillance, automatic video annotation, sign-language recognition, and human-computer interfaces one has the possibility to define specifics aspects which have to be recognized. In sign-language recognition and human-computer interfaces the gestures are limited/predefined. In video annotation the works aim, for example, at recognizing certain actions and scenes. In surveillance one can define certain situations and events which are of interest and need to be detected. Of course, this does not mean that these tasks are simple; and in surveillance one can identify situations which cannot be interpreted easily due to the concurrency of the real world where people interact with other people and objects, see [58]. It only means that one has in most of these applications the possibility to break down the problem to certain aspects which can be addressed in isolation. However, in humanoid robotics [110] one is interested in equipping the robot with the capability to interact with the environment and people, enabling the robot to improve its skills by learning from observations, and enabling the robot to apply this knowledge in unknown situations. One strategy might be to record and label all possible movement sequences the robot might need to perform or recognize. However, this seems to be a rather poor approach, if one considers complex real life situations. The number of scenes that robot might encounter are infinite even for a specific task such as cleaning a dinner table. Here, the number of plates, knifes, forks, and cups may vary. The objects can be placed at different locations and in arbitrarily many constellations. As a consequence, the actions for cleaning the dinner table need to be planned and adapted to the current situation. If the robot ob-

serves now a person who is cleaning the table, then there are obviously different levels of interpreting the scene. One could recognize, for example, low-level actions such as picking up a fork or stacking the plates. In a global view one would rather tend to interpret these actions as cleaning up the dinner table. Both levels of understanding would be important for the robot in order to understand the task as such. Consequently, one needs a more structured hierarchical representation of actions or activities in order to recognize and handle complex activities. This seems to be especially crucial if one considers a planning system on a robot which needs to compose a global goal through a number of subgoals. However, a general representation of human motions seems to be also important in a motion capture and recognition system of such a robot. Otherwise, the system would be applicable only in a very constrained context. Several works on motion capture consider only motion models for a single action or a small unstructured set of actions.

In computer vision and robotics, a number of different notions have been introduced to structure actions and activities. In [85], a hierarchy of change, event, verb, episode, and history has been introduced. In [16], the words motion, action and activity are used to address different abstraction levels. In the context of this thesis, the following hierarchy [80, 65] seems to be more appropriate: action/motor primitives, actions, and activities. Actions are supposed to be composed from action primitives, and activities are composed from actions. In [80] this is exemplified by the use of a tennis game scenario, where the player's activity of playing tennis is composed of actions as returning the ball, which in return is composed from motor primitives as running left/right and performing a forehand/backhand. This taxonomy is basically compatible to the levels of actions and activities that are discussed in the survey [125] on activity recognition. In [125], an action is understood as a simple motion pattern typically lasting for tenth of seconds and is executed by a single person, whereas an activity is characterized as a complex sequence of actions which can involve interaction of several humans.

In this thesis the following notion is used: a movement is a short motion sequence of the body or some body parts of a person. A movement can be an action or a gesture. A gesture is supposed to convey some meaning. An action is associated with some object. Either an action has a certain effect on an object or conveys a certain meaning as, for example, pointing to an object or a supposed object location. A complex action/movement is a composition of basic actions/movements. Actions and movements which can be composed to a complex action/movement are addressed as (action/movement) primitives.

The notion that human motion is composed from primitives is supported by neuroscience [102, 103] and physiology [86]. The composition of human motion is similar to speech being composed from phonemes. The hierarchical notion of complex actions/activities and the similarity of the motion composition process and language speak for the use of grammars to structure the primitives and to define their composition to complex actions/movements. A concrete grammatical framework for human movements is given in the work [38]. The work defines a language for human motion and uses parallel grammars to describe the synergetic composition of body motion on the basis of motor primitives for the different body parts. Another finding in neuroscience is the discovery of the mirror neuron system in the brain of monkeys and humans. This region is thought to encode action primitives [103, 101] and uses the same neural mechanism for the recognition and generation of action primitives. This discovery and the founded attitude that imitation is an expression of higher intelligence [110] are important motivations for the field of imitation learning in robotics. Among other objectives in imitation learning, one is interested in the acquisition/learning of movements from demonstrations, and the generation and recognition of movements. The use of a generative model for the representation of

primitives, which combines motion recognition and generation in one system, is motivated through the mirror system [56]. The mimesis framework proposed in [56] uses hidden Markov models (HMMs) as a representation of primitives. However, usual HMMs lack the capability of modeling primitives in a context dependent way.

Context dependency is an important aspect of many movements. The fish-size gesture mentioned above varies depending on the fish-size. The conveyed meaning of the gesture is the size of the fish. In a similar way actions that are applied on objects vary depending on the object location(s). Such actions are for example: reaching for an object in order to grasp it, pointing to an object, or pushing an object from a location A to a location B. The reaching and pointing actions are examples for simple goal-directed actions which depend (at least) on the location of one object. However, a primitive for pushing an object depends at least on both locations, A and B, in order to achieve the desired effect in a given context. In robotics several representations for goal-directed action primitives exist (see Section 1.1.1). An application in the work [127] is to enable a robot to throw a ball into a basket placed at different locations. Here, the throwing primitive varies depending on the basket location. Contrarily to the fish-size gesture, an important aspect of the throw is the dynamics of the action. All the so far described movements (i.e. actions and gestures) depend on the context (effect, meaning, or goal). This context is assumed to be characterized through some continuous parameters. In the following these context dependent movements are addressed as parametric movements, primitives, actions, or gestures. For a useful representation of such a parametric movement it is important that the model used generalizes over the parameter space even though it is trained only on a small set of examples. Similar to the arguments (mentioned above) for composing actions/activities from primitives, the argument for such a parametric movement model is again that it is impossible to record and store all possible trajectories of a parametric movement, since the parametrization is continuous. The recognition and generation of parametric movements for only certain parameters would be only useful in a predefined and constrained context.

If one looks at the scene context from a more global view, then one can see further relations between actions and context. Assume an environment with two objects including a cup that is placed on a heavy table. A person could point to each of the objects, but a grasping action can be applied only on the cup. One sees that each object has only certain affordances, i.e. only a certain set of actions can be applied on a specific object. (Object affordances are discussed in psychology [53].) In addition, one can see that the context basically constrains the set of actions which are likely to occur. Moreover, the context constrains which parameters of each parametric action are likely (e.g., the grasping action is usually applied on an object). The concept that objects and actions are intertwined is well known to the robotics community. The OAC (object action complex) concept [135] combines the relation between action, object, the likely effect of the action, and a measure of success. This formalization aims at planning and execution at all levels of a cognitive agent.

The discussion above shows that the representation of parametric primitives and the representation of complex movements/actions through hierarchies (as grammars) is an important aspect of a humanoid robot that should operate in a general environment and needs to recognize and generate movements. However, the robot as an autonomous online learning imitation system also needs to segment and categorize the observed motions into meaningful sequences and groups in order to refine existing primitives or to create novel primitives. Another aspect is the transfer of the movements onto the robot's embodiment. For example, the mapping of one movement trajectory demonstrated by

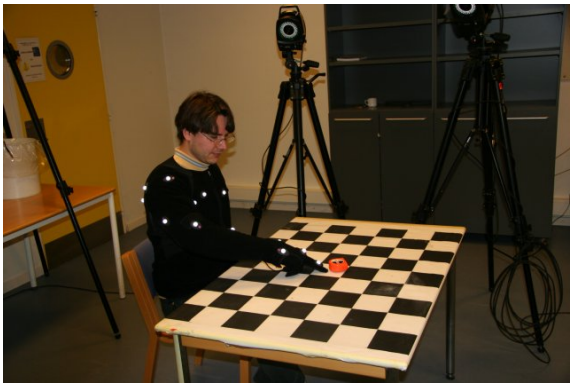


Figure 1.2: Table-Top Scenario. A person is pointing at an object.

a person may look different on the robot's embodiment due to different body sizes and proportions. Hence, an imitated movement may have a different effect/meaning even in the same environment.

This thesis proposes a unified representation of parametric movements/primitives through a model which can be used for both perception and generation. This is motivated by the mirror neuron system and the fact that an imitation system [56] requires both recognition and generative capabilities. The considered applications are imitation of parametric movements by a humanoid and the tracking and recognition on the basis of such representations which are learned on the basis of demonstrations. The learning of the model is based on segmented demonstrations labeled by their parameters (specifying the effect/meaning). However, in a joint work [66] it is shown how the learning of parametric movements can be achieved in an automatic way. The segmentation into meaningful movement sequences is performed based on salient points. However, as described in [109], the movement trajectories of the human demonstrators can be vastly different from each other if the human agent or the involved objects are at different locations even for the same actions. The strategy [109] to cluster the actions applied onto the objects is to take the change of the object state into account. Since the state changes of the objects define the effects of the body movements, the clustering results in a more meaningful set of primitives, where the parameters of the demonstrations could be extracted form the object states.

Many different models for representing movements have been considered in computer vision and robots (see Section 1.1). However, most of these models aim rather at either movement generation or tracking/recognition. Most of these models which are applied in computer vision have not been considered for the generation of precise parametric movements for robot control. A suitable model for a unified representation of movements is the hidden Markov model (HMM), which is a generative stochastic model. This model is suited to handle temporal and spacial variances in the movements during training and recognition. However, the movements, which are supposed to be represented in this thesis, are parametric, as mentioned above. If an HMM is trained on a set of demonstrations of a parametric movement, the HMM would capture the structural variation depending on the movement parameters either as variance or in an internal structure of the state transitions (see Section 2.3). In both cases, the actual meaning of a specific instance of the parametric movement can not be recovered from the HMM. In other words, if one considers a pointing action, a part of the semantic of the action, i.e. the actual pointed to location, cannot be recovered from the HMM. One approach concerning this issue is to use mixture-of-expert models [59]. However, a more promising approach is the parametric HMM (PHMM), which has been introduced in the context of parametric gestures [134] for the recog-

nition of such gestures and their parameters. The PHMMs model explicitly the systematic variation of the movements depending on the parameters. A PHMM can learn and generalize the systematic variation within a set of demonstrations. The training procedure of the PHMM is supervised, i.e. a label containing the parameters is provided for each demonstration. In this way, the meaning of the parameters of the parametric movement is introduced to the PHMM and the parameters of the PHMM can be interpreted in the same way. This is crucial for both, action recognition and generation. In the case of recognition, a PHMM that is trained, for example, on pointing actions can be used to recognize whether a presented movement is a pointing action. In addition, the PHMM framework [134] has a mechanism for estimating the parameters of the action. The estimated parameters concretize then the meaning of the action, i.e. the parameters specify the pointed to location. In the case of action generation, a PHMM can be used to synthesize a specific movement for given parameters. In the case of the synthesis for robot control, a PHMM that is trained on reaching actions, can be used to synthesize reaching actions for arbitrary object locations, thus the robot can grasp an object placed at an arbitrary location in the working space. The utilization of PHMMs for learning a unified model of parametric movements, which enables not only recognition but also the synthesis of accurate movement trajectories (as required for robot control), is investigated. Therefore the notion of prototype movements is established (Section 1.2.1) w.r.t. synthesizing from (parametric) HMMs (Section 2.2.4).

As an example, a table-top scenario is considered, see Figure 1.2. The considered actions are for example: pointing to, reaching for, and relocating an object. All these actions are parametric and depend on the object locations.

In the robotics imitation application, a humanoid robot is enabled to relocate objects placed on a table-top. An important aspect is to synthesize movements on the robot such that the robot reaches for the right object in order to grasp it. The learned parametric HMM enables the robot to generate an action for specific parameters given by the location of the object for which it should reach. Thus, the robot is able to achieve the desired effect by its actions as required in the current context. An interesting aspect of this imitation application is that the actions and their effects are learned on the basis of another embodiment, i.e. the PHMMs are trained on human performances. Both, the actions and their corresponding effects need to be mapped to the robot's embodiment.

In the tracking and recognition application, the primitive representation is used to define a space of possible actions and action sequences, where the sequences are defined by a grammar. The tracking is performed in the action space which reduces the dimensionality of the tracking problem and allows for recognition. From the tracking perspective, it is crucial that the parameters of the action primitives can be used to adapt the primitive to the actual appearance of the tracked motion, since the actions visually appear different when applied to different object locations. From the recognition perspective, it is necessary to recognize that a certain action has been performed, but in order to understand the full semantic of the recognized action, also the recovery of the action parameters is important.

An overview of the thesis is given in Section 1.2.2. The contributions are discussed in Section 1.3. Related work is discussed in the following section.

1.1 Related Work

Related work to parametric HMMs is given in Section 3.2. In the following the fields related to this thesis are discussed.

1.1.1 Imitation Learning

In 1996, Honda, announced a humanoid robot [52]. The research and development was initiated in 1986. However, it was far from satisfying that the machine required tele-operation to perform other tasks besides, for example, locomotion [111]. In [111] it is postulated that the study of imitation learning offers a promising route to gain new insights into the mechanisms of perceptual motor control that could lead to the creation of autonomous humanoid robots. For example, the exploration of new actions through reinforcement learning is rather impossible in the case of large state spaces as of humanoid robots (30 DOFs), even when using only simple motor commands for each DOF [111]. The learning of new actions through imitation of human agents can be seen here as an efficient alternative to self exploration.

Robotics research on imitation started in the early 1990s [65] under names such as teaching by showing, learning by watching, programming by demonstration (task-level learning). The focus was on the extraction of task knowledge by analyzing changes in the environment and observing (tracking) the hand [65]. For example, in the work [67] of learning by watching, the authors stress as a goal for future robots to overcome the requirements such as the need of experts and programming to enable the robot for certain tasks. The approach enables the learning of reusable task plans on a symbolic level by watching a human performing assembly tasks. A plan can then be adapted to a different environment to achieve the same assembly goal. The robot (a parallel-jow gripper supported by a Cartesian-type 6 DOF arm) is controlled in terms of control macros.

The advent of humanoid robots also made the acquisition of motor knowledge through imitation more attractive due to the same kinematic structure [65]. Early works on mapping grasping or whole body movements are [61, 100]. In [100], a kinematic model similar to the robot, but scaled to the performer, is used to calculate joint angles for the robot based on the least-squares optimization of 3D point correspondences. The replicated performances have a similar appearance. In [8] a reference description of motion data based on a kinematic reference model is established. This enables the transfer of different types of recorded motion data (via the intermediate representation based on the reference model) to different humanoid embodiments through the use of converter modules. The problem of appearance-level imitation of actions is that the mapping to the robot may alter the achieved effect.

Typical questions [10] in imitation learning are: what, when, and how to imitate. In [10] a framework is proposed for the imitation of manual tasks which addresses the questions: what and how. The relevance of the features (joint values of the arm and the 3D hand position) of the demonstrated movements is analyzed based on the variance in the movements. This results in a cost metric. The movements are then reproduced by minimizing the cost to imitate the relevant aspects given the further constraints of the robot.

Various different models and approaches are used to represent movements and parametric movements. In [78] via-points are used as control variables of the demonstrated trajectories. A selection scheme of via point for different sub-goals is proposed based on which the trajectories can be adapted to achieve the task goals (as a tennis serve). In the approach [6] to imitation, HMMs are trained on key-points of trajectories in order to generalize the movements demonstrated to the robot. Gaussian Mixture models are used in [23] to represent trajectories. The time is encoded by augmenting the sequence samples by a time variable. In [126] and [127] a regression model is applied on a set of exemplar movements to synthesize goal-directed actions as throwing a ball in a basket at arbitrary

locations. The movements are represented for the regression through key-points. Dynamic movement primitives (DMPs) [55, 111] have been proposed to represent cyclic movements (e.g., walking) and goal directed movements (e.g., reaching). Typically these primitive models use differential equations to describe the state evolution. The model parameters can be learned on the basis of demonstrations. For goal directed movements an attractor is used to assure that the actual goal is reached. The mimesis framework [56] uses HMMs as a representation of "symbols" (primitives). The framework generates symbols based on observations and is used also for recognition. The synthesis process is based on generating and averaging a large set of output sequences.

Discussion: A disadvantage of the regression model [127] is that the key-points have to be detected in order to generate meaningful trajectories. The detecting of proper key-points is crucial. The advantage of PHMMs is here that HMMs have built-in warping capabilities, which make a pre-processing step as aligning the trajectories unnecessary. DMPs are an appealing approach for synthesizing movements. However, they do not provide a statistic framework for recognition.

1.1.2 Motion Capture

The term, human motion capture, usually addresses the process of registering large scale human motions, as the movements of the head, arms, torso, and legs [79]. The capturing of small scale movements are generally addressed in other research fields. Examples therefore are facial feature tracking [25] and hand pose estimation [105, 104]. However, the term human motion capture applies also to the capturing at different resolutions where the subject may be understood as a single entity (low resolution) or as a highly articulated subject with a skeletal structure (high resolution) [79]. Human motion capture is an active research area due to its large number of applications and its inherent complexity [80]. The vision-based recovery of the human motion comprises difficult and ill-posed problems, such as inference of pose for a highly articulated and self-occluding non-rigid 3D object [80]. In contrast to rigid objects, the clothes on the human subject can cause appearance and shape changes, e.g., through lighting changes or the wrinkling of clothes. The clothes may even disguise some parts of the body pose, an example for such clothes are frocks (see [21]).

Current and potential applications can be categorized into control (1.), surveillance (2.), and analysis (3.) [79]. 1. Typical control applications are [19]: character animation for movies, interactive virtual worlds (e.g., in computer games), teleconferencing, or advanced human-computer interfaces. Character animation is an example, where the pose estimates of an articulated human model is the desired output of a capturing system. Contrarily, human-computer interfaces (HCI) require also some understanding of the captured motion. The approach [134] to gesture recognition relies on trajectory data, whereas other works approach the recognition problem on the basis of image features (see Section 1.1.3). 2. In surveillance, typical systems track one or several subjects and monitor the subject(s) [79]. The fall incident detection [72] makes use of the roughly tracked motion of the subject, whereas the cashier activity recognition approach [29] works on the basis of image feature-based event detectors. 3. Examples of applications of motion analysis are [79, 19]: clinical studies, assisted sports training, diagnosis of orthopedic patients.

Due to the different applications and their requirements, different types of technologies exists. Examples for active sensing [79] systems are electromechanical systems from Metamotion, which measure directly the body pose (see [34]), and electromagnetic systems from Motionstar (see [34]),

which use sensors which are attached to the person. Some systems from Vicon make use of reflective markers which are recognized in different camera views. The use of video cameras or IR cameras are typical examples for passive sensing. The use of markers can be seen as a compromise between active and passive sensing [79]. However, the active sensing and the marker-based devices are to some level intrusive and require the cooperation of the subject, which makes them impractical or unusable in certain applications, especially in surveillance.

The use of usual consumer cameras seems to be preferable, due to cost and non intrusiveness, but makes greatest demands on the processing of the data. To simplify the capturing task, most works on vision-based motion capture rely on constraints [79] concerning: the environment (static or uniform background, controlled lighting condition), the subjects (e.g., one subject, specially colored or tight-fitting clothes, slow motions, or known motion pattern), and the view on the subjects (e.g., walking parallel to the camera plane (e.g., [89])). Some works aim particularly at overcoming certain constraints: in the work [32], the body model's surface adapts to the clothes, and the work [43] aims at outdoor motion capturing.

The tasks which a motion capture system usually addresses are [79]: initialization (finding/setting, e.g., thresholds, camera calibration, initial body pose, size of body parts), tracking (establishing coherent relations of the subject(s) between frames), pose estimation (finding the configuration of the body or body parts of the subject(s)). The focus is in the following on works for video-based capturing. Usually these works make at some level of the task (tracking or pose estimation) use of a human body model. Different models are for example [20]: stick figures, where the nodes define some stylized joint positions (defined through 2D/3D locations or defined through local transformations within the built-in hierarchy), or models, where the skeletal structure is fleshed out through basic geometries (cylinders [114], superquadratic [121], meta balls [90], etc.) or a skin mesh [32, 36]. Typically, these models are then articulated based on a set of parameters (e.g., describing joint values) which are defined as a state vector. The state vector defining the body pose is then usually augmented by the global pose [36, 26]. Such an augmented state vector has the advantage that the estimated state vector of the estimated pose can serve directly as a description of the captured motion. The change of the state vector between frames can be used to predict the pose [34] for the next frame (which addresses basically the tracking task). Further model parameters are used, for example, to define the shape of the skin [32] or the appearance [115] in form of texture parameters (of a trained appearance model). Analysis-by-synthesis approaches, which analyzes a scene by comparing its appearance to a model of that scene [19], are common practice in many works (e.g., [114, 26, 36, 90]). Either the model is synthesized in a hypothesized pose and compared directly with the scene [26], or the synthesized model is used to establish correspondence to define some error measure which is then optimized for the pose estimation [36]. Note, not all works make (directly) use of body models and synthesize the body pose in order to make pose estimates, see [80].

An early work on motion capture is for example [20], this work and also [90, 36] make use of gradient based optimization techniques to estimate the pose. A disadvantage of these techniques is that they can get stuck in local minima (e.g., due to the depth disambiguations in a monocular view) from which they can not recover (easily).

In the following, the focus is on newer works which make either use of an articulated body model or which are interesting in the context of this thesis. Three different directions are considered, which aim also at overcoming the pose estimation problems of the gradient-based techniques: 1. *Particle fil-*

tering. A typical technique to overcome the problems of gradient-based techniques is particle filtering. These works approach the pose estimation usually in an analysis-by-synthesis approach, where each hypothesis (given by a particle) is compared with the scene. However, the disadvantage of particle filtering is that the number of required particles scales exponentially with the number of dimensions [77, 26]. Due to the high dimensionality of an articulated body model, the processing time for evaluating all hypotheses becomes large. In order to use the particles more efficiently, different approaches have been investigated: annealing [26] (multi-stage filtering which allows reducing the number of particles required without losing track of local maxima/minima), interval particle filtering [107] (focusing on certain DOF), jump diffusion [118] (approaching the problem of depth disambiguations in monocular views), covariance scaled sampling [117], interacting particle filters [31], shape-encoded particle propagation [81], and partitioned sampling [77]. Other works aim instead at reducing the dimensionality of the state space, e. g., by using motion models. These works are discussed under point three.

2. *Complex Frameworks (Combining Different Techniques).* Novel frameworks use, for example, multistage approaches ([70] considers the stages: coarse tracking of people, body part detection, 2D joint location estimation, and 3D pose inference) or implement various constraints ([41] considers the constraints concerning self-occlusions, the kinematic, and the appearance; and uses belief propagation to infer the pose on the bases of a graphical model). The body model in the work [32] models also deformations of the skin, e. g., in order to adapt to clothes. The framework uses of a mixture of particle filtering and local optimization techniques, which are applied on different levels of the model (refining the pose of limbs, refining the whole body pose, refining the meshes of the skin).

3. *Utilizing Motion Models.* The following works are especially interesting in the context of this thesis. In the works [128, 114] linear subspace models are established, e. g., for cyclic motions as walking. Based on example sequences of the walking cycle, a compact representation is generated through principal component analysis [114]. The representation consists then of a number of eigenmotions. An arbitrary motion is a linear superposition of these and is specified through some scalar parameters. These (latent) parameters and the progress of the motion is part of the state space. The technique used is particle filtering. An extension to multiple types of motions is given in [116], which structures the motions in a tree.

In the work [89], a switching linear dynamic system (SLDS) is described. The SLDS is trained on two different types of motion (running and walking). The model is then used for supporting the tracking and for classifying/identifying motion regimes (walking/running) based on the switching states.

The works [129, 130, 96, 131] make use of Gaussian process models (GPMs). The GPM in [129] is a static model, which learns a low dimensional embedding in a latent space of the high-dimensional pose data and provides a density function over the latent space and the pose space. The model provides also a non-linear probabilistic mapping from the latent space to the pose space. In [130] an extension of the GPM to a dynamic model (GPDM) is used, where the model includes in addition a dynamic model in the latent space. In the experiments of [129, 130], the motions walking and golf swing are considered (independently), where the pose estimations is done based on tracked 2D joint locations. In [96], particle filtering is investigated in combination with GPDMs. In the experiments different actions are considered, but the transition between models of different actions is not considered. An interesting work in this regard is [25], in which a switching layer is introduced in top of the GPDMs that enables the simultaneously tracking/recognition of multiple motion types.

Alternative techniques to GPDM are Laplacian eigenmaps latent variable models [73] and local linear embedding (LLE) [28].

An interesting work concerning motion models for 3D motion capture is [122]. The work introduces the so called implicit mixture of conditional restricted Boltzmann machines. The computational cost of the training procedure is depending linearly on amount of training data (which is not given for the GPMs [69]). It learns coherent models of different movements and can infer transitions between the movements, even when not present in the training data. The dimensionality reduction is only implicit such that large deviations from the training set are possible (e.g., if the model is trained on usual walking the tracking is not condemned to fail when the tracked subjected scratches the nose).

Discussion: The works concerning GPDMs and linear subspace models (generated on the basis of PCA) considered above do not aim at the recovery of the latent parameters. Even though the latent parameters have a certain meaning, the interpretation is not obvious, since the low dimensional embedding is generated in an unsupervised way. When aiming at recognition, the works (mentioned above) concerning GPDMs and SLDS consider only the recognition of the type of motion. A problem of the GPDM [131] learning procedure is that the procedure can produce gaps in the latent space, when the same pose appears several times in the modeled motion, see [131]. An interesting aspect in the context of this thesis is that the learning of NLDS requires a vast amount of training data [131]. The GPMs training is in $O(N^2)$ or $O(N^3)$ [122], where N is the number of training sequences. Whereas the training of parametric HMMs is in $O(N)$. An advantage of the PHMMs is that the meaning of the latent parameters is given through the supervised training procedure. A further advantage of the PHMMs in comparison to the linear subspace models (generated on the basis of PCA) is that the PHMMs have the inbuilt capability to cope with different dynamics of the training examples. Moreover, the PHMMs model also the variance of each instance of parametric movements, which is preferable in the case of recognition.

1.1.3 Action Recognition

The recognition of human actions and activities has several applications as content-based video annotation, retrieval, and summarization, human-computer interaction, and surveillance. Comprehensive review articles on action and activity recognition are [80, 125]. Typically, authors [80, 65] make use of terms as action, activities, simple/complex action or behaviors to characterize the complexity of the recognition task.

Important aspects of action recognition are view invariance and identity invariance. As an example, the point trajectories in the image plane of a subject are sensitive to translations, rotations, and scaling [125] and are highly depended on the orientation of the subject to the camera due to the perspective projection onto the image plane. Therefore alternative representations as speed and spatio-temporal curvature are more useful [95]. The work [95] aims at view-invariant action recognition and presents a representation which captures dramatic changes in speed and direction of 2D point trajectories. However, already the extraction of accurate point trajectories is complicated due to occlusions, noise, and background clutter [125].

A typical hierarchical structure of a comprehensive activity recognition systems is [125]: modules such as segmentation, tracking, and object recognition at the lower-levels, action recognition modules at mid-level, and reasoning engines at the high-level. In the following, different categories of works

on action recognition are discussed.

Template Based Approaches. In [91] the periodicity of (sufficiently) periodic movements of subjects is detected to generate a sequence of 2D templates describing an action cycle. For template generation, the subject is tracked and segmented, spacial/temporal scale changes and the subject's translation are compensated. Each 2D template consists of a grid aligned in x- and y-direction, where each bin contains some flow-based statistics of the corresponding region over the corresponding temporal segments of the cycles. The recognition is performed by template matching. In [17], the regions of the subject's motion in an image frame are segmented by using difference images. The segmented image sequence of a performed action is aggregated in two types of template images: MEI and MHI. The motion energy image (MEI) is a binary valued image, where a foreground pixel indicates where motion has occurred in the image sequence; the motion history image (MHI) is a scalar-valued image generated by summing up the segmented images with increasing weights. The MEI and MHI are then used for recognition. The matching is performed by using the statistics of the scale, translation, and orientation invariant Hu moments [17]. A proposed backward-looking algorithm is used to account for different expansions in time. View invariance is established by using templates acquired for different viewing directions. An extension of the 2D templates in [17] to 3D templates of visual hulls is considered in [133].

Volumetric Approaches. A video sequence can be understood as spatio-temporal volume. The segmented silhouettes [15] or contours [137] of a person in a sequence of frames is then a volume or a surface. Such a volume/surface can be interpreted as an object, which allows one to use object recognition approaches. In [137], the stacked contours of an image sequence of an action are basically an "action" surface. Action "surface" descriptors are generated for important interest points (ridges, valleys, etc.) of an action "surface". The action recognition task becomes then a matching problem of these interest points. In [15], the silhouettes form an object volume. From the solution of the Poisson equation a variety of local shape features are computed within the volume. A set of moments of these features form then a global shape/action descriptor of an action. The recognition is performed by descriptor matching. The advantage in comparison to [137] is that the approach does not rely on a mechanism for matching points. In order to establish a better view point invariance (e. g., in comparison to [15, 137]) the work [119] makes use of manifold learning in order to learn how the appearance of an action varies when the viewpoint changes. The normalized R transform (translation and scale invariant) is used as a shape descriptor of silhouette images. A time-normalized sequence of silhouette images of a performed action is then described as a 2D surface by stacking the 1D shape descriptors of each image. For a demonstrated action, the surfaces are generated for 64 views by visual hull animation. The 1D cyclic embedding of the 64 surfaces is generated by the Isomap algorithm which results in a functional description of the surface depending on the viewing angle.

Space-Time Features. The space-time (ST) volume is here understood in comparison to the approaches above rather as a 3D intensity image, which allows one to use image feature detectors and descriptors specialized for detecting/describing salient points of the ST volume [68]. In [112], an SVM approach is used for recognizing different actions. In [87], a graphical model is used to discover action categories in an unsupervised manner. In contrast to the local approaches (based on salient points), the tensor-based approach in [63] is a holistic approach on the basis of the ST volumes.

Statistical Models. Various different models are applied for action and activity recognition, only some examples for different models are given: HMMs are used, e. g., in [4] and [120] for human action

and sign language recognition. Dynamic Bayesian networks are used in [99] for gesture recognition. A switching hidden semi-Markov model is used in [27] for activity recognition. In [58] stochastic context-free grammars are used for the detection and recognition of temporal extended activities and interactions between subjects.

1.2 Overview

As mentioned in the introduction, this thesis considers a table-top scenario as shown in Figure 1.2. A person sits in front of a table-top and performs actions on different objects. The considered actions are, for example, reaching or pointing which are parametrized by the object locations. In the introduction and the related work section it is argued that HMM/PHMMs are very appealing movement models. HMMs are an appealing movement model, since HMMs enable both recognition and generation of movements, which suits the concept of imitation and mirror neurons. HMMs are able to handle and model the *natural* variances of human movements. These variances are the spacial variances and different dynamics. The parametric extension of HMMs (PHMM) can model explicitly the *systematic* variation within parametric movements. This enables the recognition of the full semantic of a specific performance of a parametric action and the synthesis of a specific action with the desired effect/meaning.

However, in order to make the chapter overview (Section 1.2.2) and the contribution of the thesis (Section 1.3) easier to grasp, it makes sense to: a) give some example for the *natural* and *systematic* variances of human movements, b) discuss the notion of prototype movements/actions that is used in this thesis, and, c) discuss how one can compare movement trajectories.

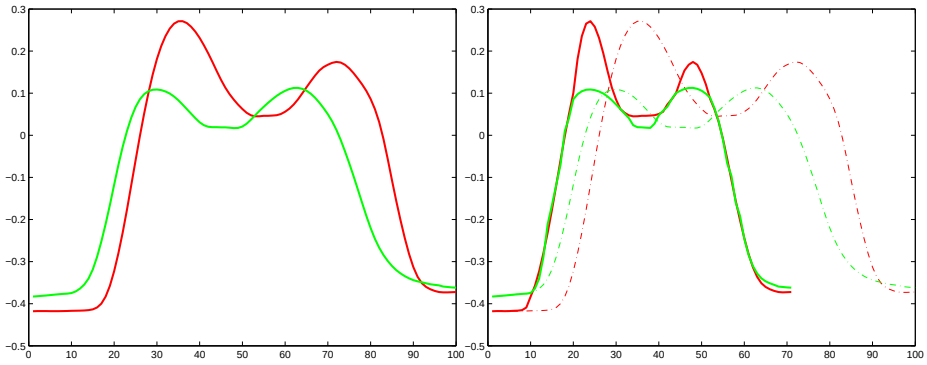


Figure 1.3: Dynamics of Human Movements and Dynamic Time Warping. Each function shows one dimension of a recorded 3D finger-tip trajectory over time. ***Left:*** *Two Recorded Sequences.* — ***Right:*** *Time-Warped Sequences.*

1.2.1 Parametric Movements

In Figure 1.3 (left) and Figure 1.4 (left) show performances of a pointing action, where a human points repetitively in the same way at a single target location. However, each performance misses this location slightly. In Figure 1.4 (left), the x, and y components of the 3D finger-tip trajectories are displayed. Only the first part of the performances are shown here, where the finger approaches the target. In Figure 1.3 (left), the z component of the finger-tip trajectories are displayed as functions

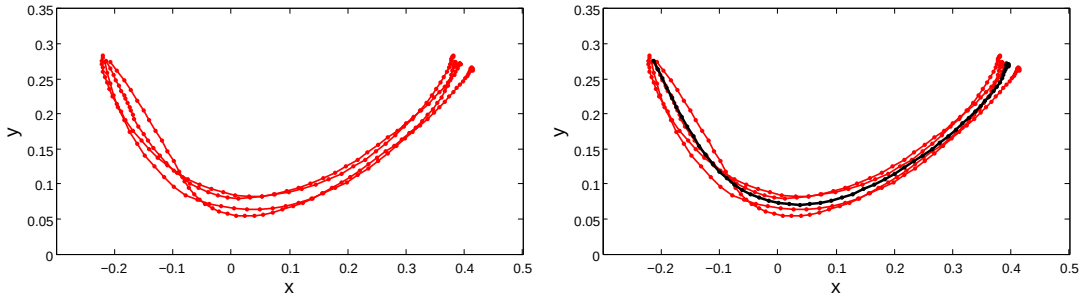


Figure 1.4: Non-Parametric Human Movements: Spacial Variance and Prototypes.
Left: *Recorded Trajectories.* The plots show 4 recorded repetitions of the same pointing action performed by a human. Only the 2D projection of the 3D finger-tip trajectories is shown. The person had to point always at one location on the right side of the plot. Each trajectory shows some spatial variation and also the target has been missed slightly. — ***Right:*** *Prototype Movement.* A possible prototype of the action is shown in black. Here, it is calculated by averaging.

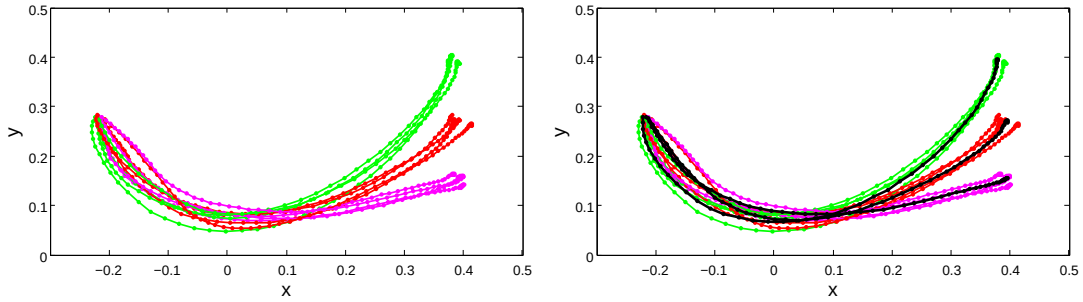


Figure 1.5: Parametric Human Movements: Spacial Variance and Prototypes.
Left: *Recorded Trajectories.* The plot shows several performances of a parametric pointing action. Here, three pointing targets are considered. For each target 4 repetitions are displayed. — ***Right:*** *Parametric Prototype.* For each target a possible prototype is shown in black. These prototypes can be understood as three different instances of a parametric prototype.

over time. The approaching and the withdrawing parts of two performances of the pointing action are shown here. The small plateau in the middle belongs to the phase where the finger rests for a short while at the target. One can see in both figures that the finger-tip trajectories differ in their location in 3D space, even though the recorded person tried to perform the same movement. In Figure 1.3 (left) one can also see that the execution speed varies over the performances. Since the person tried to perform the repetitions in the same way, it makes sense to assume that there exists an underlying prototype of the movement. Each performance is then a natural variation of the prototype. A possible prototype is shown in Figure 1.4 (right). Figure 1.5 (left) shows repetitions of pointing actions for three different target locations. Still, it makes sense to assume that there is some underlying prototype for each target location (see Figure 1.5 (right)), where each performance is a *natural* variation of the corresponding prototype. However, here it makes sense to assume that there is some prototype movement which varies in a *systematic* way depending on the target. Such a prototype movement

that varies in a systematic way depending on some parameters is called in the following *parametric prototype (movement/action/primitive)*. As mentioned above, HMMs are able to model and handle the natural variance of movements. The parametric HMM (PHMM) is able to model explicitly the systematic variation of the underlying parametric prototype.

Comparing and Aligning Movements. Another important aspect is to measure the similarity of movements. The two trajectories Figure 1.3 (left) appear to be similar. However, the area between the graphs is rather large. This is partially based on the different dynamics of the movements. Another problem may be that the recorded movements can differ in their length or number of samples. Dynamic time warping (DTW) is a method which allows one to align the movements to some degree with respect to their dynamics. In Figure 1.3 (right), the two movements are warped by a DTW algorithm. This works also when the movement sequences differ in length. The area between the warped sequences in Figure 1.3 (right) is much smaller and seems to reflect the actual similarity of the sequences. DTW can be also applied on multivariate sequences. A reasonable similarity measure is then, for example, the root-mean-square of the Euclidean distances of the corresponding samples of two sequences aligned by time warping. DTW is explained in Appendix B. Another aspect is the interpolation or averaging of movement sequences. The sample-wise interpolation of the aligned sequences in Figure 1.3 (right) seems to be reasonable to get a sequence with a similar shape. Contrarily, the sample-wise interpolation of the un-aligned sequences (Figure 1.3 (left)) seems to be questionable, since one would interpolate between parts of the sequences with different semantics. Here, one would interpolate, for example, samples which belong to the parts of the movements, where the finger approaches the target and where the finger is withdrawn from the target. This situation would become even severe if the movements differ more in their dynamics.

1.2.2 Overview and Chapter Outline

The contributions of this thesis are discussed in the following Section 1.3. In the appendix, an introduction to DTW (Appendix B), particle filtering (Appendix C), modeling of 3D scenes (Appendix D), and Gaussian distributions (Appendix E) is given. An overview of the notation used in this thesis is given in Appendix A. An overview of the main chapters of the thesis is given below. A final conclusion of the thesis is given in Chapter 7. Publications which are related with this thesis are: [47, 49, 48, 50, 45, 46, 66, 51]. D. Herzog is first author of the works [47, 49, 48, 50, 45, 46, 51], and second author of the work[66]. The works [47], [48], [50], [51] are associated with the Chapters 3, 4, 5, and 6, respectively. However, the thesis is only partially based on the these works and provides further aspects and experiments.

- **Chapter 2**: This chapter provides an introduction to the hidden Markov model (HMM) and investigates the utilization of HMMs for representing (non-parametric) prototype movements w.r.t. recognition, training, and synthesis of prototype movements.

  The introduction to HMMs is given in Section 2.1.1, which provides: an overview of the notation that is used in this thesis w.r.t. HMMs (Section 2.1.2), a discussion of HMM-related problems (Section 2.1.3), and approaches to these problems (Sections 2.1.4.2 and 2.1.5). Short notes are given about the implementation of the HMM framework (Section 2.1.6) and its run time behavior (Section 2.1.7).

Section 2.2 discusses different types of HMMs, problems of the HMMs training procedure (Section 2.2.2), and extensions of HMMs (HMMs with time restrictions (Section 2.2.3.2) and a finish state (Section 2.2.3.1)). Different synthesis and recognition methods are discussed in Section 2.2.4 and Section 2.2.5, respectively. A method of synthesizing prototype movements is proposed (Section 2.2.4, Method D).

The training of HMMs on movements is then investigated in Section 2.3 with a view to synthesizing prototype movements by Method D. The preliminary findings (summarized in Section 2.4) are that the training of left-right model with time restrictions and a finish state is very robust and converges quickly. Such a trained HMM is suitable for the synthesis of good prototype movements by Method D.

- **Chapter 3**: This chapter introduces parametric HMMs (PHMMs) in the context of parametric human movements.

  Section 3.1 establishes the notion of PHMMs as it is used in this thesis. Section 3.2 provides PHMM-related work which utilizes PHMMs for the recognition and synthesis of human movements. Section 3.3 gives an overview of the linear PHMM (LPHMM) and the non-linear PHMMs (NLPHMM) which are introduced in the work [134] (Wilson and Bobick). Another interpolative approach to PHMMs (IPHMM) introduced in [47] (Herzog et al.) is discussed in depth in Section 3.4. This approach has the advantage that it makes only use of the usual HMM framework, whereas the LPHMM and NLPHMM training procedures require some extensions of the HMM framework. However, the training procedures of the LPHMM and NLPHMM are more flexible, and the NLPHMM is more general. Final remarks on PHMMs are given in Section 3.5.

- **Chapter 4**: This chapter discusses the representation of parametric human movements and the representation of these through PHMMs (Section 4.1), also the representation of bi-parametric movements is considered which can be used as primitives to build complex movements. The PHMMs are then evaluated for the synthesis and recognition of parametric human actions.

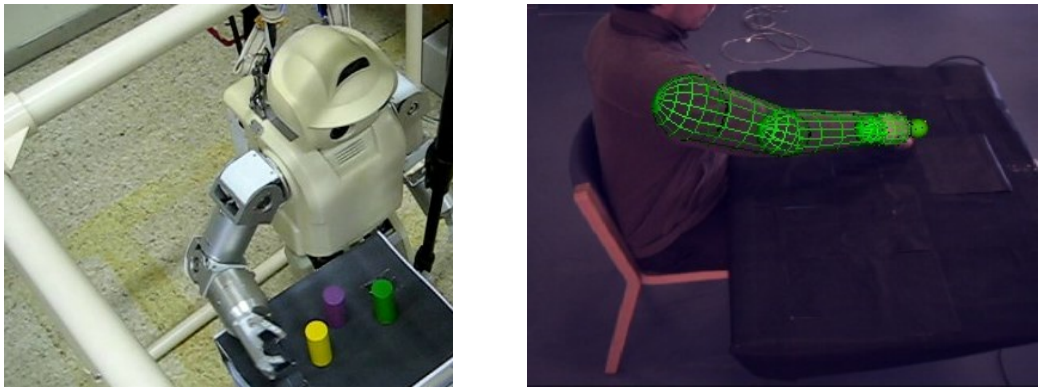


Figure 1.6: Motion Imitation and Tracking in Action Space. ***Left:*** *Motion Imitation.* The humanoid robot imitates a parametric reaching action in order to grasp the yellow object. — ***Right:*** *Action Tracking.* A person is performing a pointing action. The arm pose is tracked. The pointed to location is estimated online. The estimate is indicated by the green ball (close to the hand). The color of the ball (green) indicates that a pointing action is recognized.

The training of PHMMs for utilizing recognition but also the synthesis of parametric human movements is discussed in Section 4.1.2. The proposed approach is to use no-skip left-right HMMs with time restrictions, a finish state, and a large number of states in order allow one to use a trained model also for the synthesis of accurate prototype movements.

In Section 4.2 such PHMMs are evaluated for the synthesis of basic parametric human arm actions for arbitrary action parameters, where the parameters specify the table-top locations. The actions are pointing and reaching. The PHMMs with a large number of states are still suitable for classifying (Section 4.2) actions and recognizing their parameters. In these experiments IPHMMs and LPHMMs are evaluated. The recognition and synthesis experiments are an updated version of [48] (Herzog et. al), since the HMM framework used in [48] makes implicitly use of a covariance prior. Extensions of this section are: an quadratic extension of the linear PHMM is proposed in Section 4.2.3. The quadratic PHMM (QPHMM) represents the systematic variation of the considered action very accurately (which is evaluated). In addition, the time durations of PHMM states are considered Section 4.2.4.2

In Section 4.3 it is shown that a PHMM can be also used to learn and synthesize more complicated bi-parametric arm movements. The considered action is parametrized by two object locations, where the hand approaches the objects in sequence.

- **Chapter 5**: This chapter concerns the imitation of parametric arm actions by humanoid robots, see Figure 1.6 (left). The effect of the actions is modeled by the parameters of the actions.

  The chapter begins with a discussion of imitating (parametric) actions which have a certain meaning/effect. The meaning/effect can become quite different when the embodiment of the imitator differs from the embodiment of the imitated demonstrator.

  Approaches to mapping of parametric actions on and converting parametric actions models for another embodiment is discussed in Section 5.1, an important aspect is to preserve/map the meaning/effect of the action parameters.

  The imitation of parametric arm action by humanoid robots is discussed in Section 5.2, see Figure 1.6 (left). The action models (PHMMs), which are trained on demonstrations by a person, are used to synthesize prototype movements for robot control. The implementation enables the robot to reach for objects which are arbitrarily located on a table-top. The robot can then grasp and place the object somewhere else (Section 5.2.5). Note, the robot's embodiment differs significantly from the demonstrator. The new capabilities of the robot are evaluated in a rule-learning task (Section 5.2.6), where the robot has to relocate several arbitrarily located objects as demonstrated by an advisor. The robot implementation is published in [50] (Herzog et. al). However, this section is extended with details of the robot used. An evaluation of the imitation error is added in Section 5.2.7.

- **Chapter 6**: This chapter provides the theory, implementations, and experiments for the novel approach (Tracking in Action Space) to 3D online-tracking and -recognition of parametric movements/actions and their parameters, see Figure 1.6 (right). The approach allows the online-capturing of the arm pose and the online-recognition of parametric arm actions and their parameters and this in the case of multiple actions and complex actions. The approach is published first in [51] (Herzog et. al). However, the extension which makes the implementation ready for

online-tracking and more robust is provided in this chapter. The experiments are completely new (except one).

The chapter begins with a general discussion of motion capture and action recognition, where the action conveys a certain meaning (parametric actions). The conveyed basic idea is to see the tracking and recognition as an intertwined problem. Related work is given in Section 6.2. The basic concept of Tracking in Action Space, is to perform the tracking in the space of the parametric actions and concatenations thereof (Section 6.3). Section 6.4 explains the details: utilizing the PHMMs for modeling the action space, the application of particle filtering to establish the tracking in this space, and the basics of recognizing the actions and their parameters. The observation model including the scene and human body model are given in Section 6.6. This includes a matching approach which makes the tracking more robust.

In the experiments (Section 6.7) the following aspects are evaluated: Section 6.7.1: The recovery of the body pose when using a high or a low frame rate and when using single or multiple views. The recognition of a single action and the recognition accuracy of the parameters of the action while tracking online. Therefore, different actions are investigated. Section 6.7.2: The recognition of actions and their parameters is investigated when the scenario consists of different actions. Section 6.7.3: The tracking and recognition of complex actions, which are a concatenation of several basic parametric actions (action primitives), are considered.

Final remarks are given in Section 6.8.

1.3 Contribution

The contributions of this thesis are in short: a) establishing PHMM as a movement and primitive representation which allows one besides recognition also precise synthesis of prototype actions in an effect/meaning dependent way as it is required for the imitation by a humanoid robot. b) imitation of parametric actions on a humanoid robot in order to achieve the desired effect/meaning. c) establishing a tracking/recognition framework which makes use of the representation of parametric actions/primitives and enables tracking and recognition in an intertwined way, where the action primitives can be used to model complex actions through grammars.

PHMMs as Representation of Parametric Movements and Action Primitives.

- A notion of (parametric) prototype movements/actions is established.
- The HMM/PHMM learning procedure is tuned w.r.t. this notion, thus that the trained (parametric) HMMs can be used to synthesize prototype movements/actions in an un-complicated way.
- A main goal is to establish the PHMM as a unified framework for action recognition and synthesis. For a precise representation of the actions, (parametric) left-right HMMs with a large number of states and time-restrictions are used. It is shown that the training of such (parametric) HMMs is uncomplicated and successful with the proposed training strategy. It is evaluated that the representation of an parametric action through such a single PHMM allows one to synthesize accurate prototype actions. Such a PHMM is evaluated as being appropriate for recognition. The

evaluation for synthesis and recognition considers here the whole parameter range of the learned parametric actions.

- It is shown that PHMMs can be trained in order to represent bi-parametric actions. The bi-parametric actions can then be used as primitives which can be concatenated in a meaningful way to represent complex actions.
- Additional contributions are the proposed quadratic extension of the linear PHMM and the interpolative approach to PHMMs.

Imitation of Parametric Actions on a Humanoid Robot.

- PHMMs have not been considered so far in robotics as a representation of parametric or goal-directed actions.
- It is shown that the PHMM representation can be successfully applied to imitate learned actions on a humanoid robot, where the action parameters are successfully used to achieve the desired effect/meaning on the robot.
  - A contribution is here the proposed way of converting the parametric representation of an action which is learned based on demonstrations of a human (with a certain embodiment) to a representation which suits the embodiment of the human robot used (note, the embodiment differs in its proportions and the size). It is worthwhile to note that the effect/meaning of a demonstration is only provided (in terms of parameters) for the embodiment of the demonstrator not when reproduced on the robot. As a consequence, the conversion of the body motion and the meaning of the parameters have to be considered.
  - In addition, other approaches for a scenario with demonstrations from several demonstrators and several imitators are proposed.
- A minor contribution is the extended application of an action primitive (reaching for) for a more complex task (relocating objects) without introducing poses that are not part of the primitive.

Tracking and Recognition.

- One contribution is the proposed view of the tracking and recognition of action primitives and complex actions as an intertwined problem, where the tracking is performed within the proposed space of possible action which can be modeled through action primitives:
  - The so called *action space* uses PHMMs to model action primitives and has a lower dimensionality than the pose space.
  - The concept is to use only a basic set of action primitives which can model then many different complex actions.
  - The primitives are parametric, which generalizes their applicability in different contexts.
  - The tracking is performed here within the proposed space of possible actions or action sequences such that the recognition becomes a part of the tracking.

  - The action primitives and their parameters can be recognized online, where the recognized parameters provide an important part of the semantic of a performed action.
  - The approach allows for further argumentation and extensions: a) One could argue that it is not necessary to have perfect pose estimates in order to accomplish the recognition of the actions. b) The approach allows one to incorporate knowledge of the context into the tracking. The argument is that objects of the context provide a prior knowledge of actions and their parameters (this is discussed in the introduction). The notion of the action space allows one to incorporate this knowledge into the tracking simply by adjusting the particle propagation within the action space.

- the implementation and the recognition of actions and their parameters works in real time (online) on the basis of a single camera view
- the experiments investigate and answer fundamental questions:
  - the recovery of the body pose
  - multiple views and single views
  - different viewing directions (in the case of single views)
  - recognition of actions and their parameters in the case of single and multiple actions
  - the recognition/tracking of complex actions which are concatenations of action primitives
- The application of motion models is not new to motion capturing community. Two contributions can be seen here:
  - The latent variable models generate the embedding in the latent space in an unsupervised way, where the meaning of the latent parameters is unknown. Consequently, the recognition is then constrained to recognizing the used motion model during the tracking. However, in the case of parametric actions/gestures also the recognition of the specific instance of the action and its parameters are important to understand the full semantic. The PHMM as motion model allows one here to recognize both, the action type and their parameters.
  - For generality of the tracking it is required to allow transitions between the motions that are modeled by a set of motion models. A simple switching between models is not always meaningful, for example, when the pose after the switching has nothing to do with the pose before. Since the PHMM models the systematic variation explicitly, it is easy to define proper transitions between movements. As a consequence, one can use a basic set of movement primitives. A complex movement/action is then only a concatenation of these primitives. In a complicated scenario with arbitrary movements/actions it is crucial to use primitives since the number of complex movements can be arbitrarily large.

Chapter 2

Learning Movement Trajectories with Hidden Markov Models

This chapter discusses the application of the HMM framework for the training of human movement trajectories. It discusses, beside different hidden Markov models (HMMs), the recognition and different methods for synthesizing trajectories from the model (Section 2.2). The preliminary findings are that the left-right model (with time restrictions (Section 2.2.3.2) and a finish state (Section 2.2.3.1)) is well suited to generate good prototype movements (Section 2.3). However, without the parametric extension (parametric HMMs, Chapter 3), the HMMs are not well-suited to model parametric movement classes for the purpose of recognition and synthesis when one is interested in recognizing and synthesizing particular instances of the movement class.

The chapter begins with an introduction to HMMs in Section 2.1 which includes a description of the implementation of the HMM framework (Section 2.1.6) and its run time behavior (Section 2.1.7).

2.1 The Hidden Markov Model

This section provides an introduction to the hidden Markov model (HMM) in Section 2.1.1, an overview of the notation that is used in this thesis w.r.t. HMMs (Section 2.1.2), and a discussion of HMM related problems (Section 2.1.3) and approaches to these problems (Sections 2.1.4.2 and 2.1.5). Short notes are given about the implementation of the HMM framework (Section 2.1.6) and its run time behavior (Section 2.1.7). Some references for HMMs are [54, 93, 92, 14, 11, 18].

2.1.1 The Model

The Hidden Markov Model (HMM) is a finite state machine that is described in a probabilistic manner. This concerns the state transitions and the outputs of each state. The output distributions are either probability mass function (pmfs) in the discrete case of an HMM, or, in the continuous case, probability density functions (pdfs). A simple example of a discrete HMM is shown in Figure 2.1.

An HMM has a finite set $\mathcal{S}$ of states. For convenience, the states are addressed in this thesis through integer numbers, where the set of states is assumed to be of the form $\mathcal{S} = \{1, \ldots, N\}$, where N denotes the number of states. Generally, an HMM is denoted as a triple $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$. The stochastic matrix $\boldsymbol{A} = (a_{ij})$ defines here the probabilities of transitions between the states, where a_{ij}

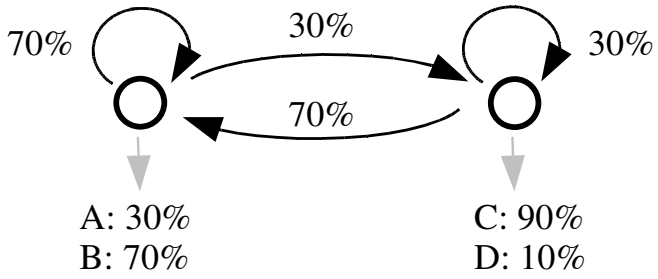


Figure 2.1: Two-State HMM. The figure shows an HMM with discrete output symbols. Both, the transition arcs and the output symbols are labeled with probabilities. Possible output sequences are for example: ACBBC, or ACBBAD. The sequence DDDDD is a rather unlikely output sequence due to the probabilities of the self transition and the output symbol D of the state on the right side.

is the probability of an outgoing transition from a state $i \in \mathcal{S}$ to a state $j \in \mathcal{S}$. As a consequence, it is $\sum_j a_{ij} = 1$ for each state i. The output distributions are specified through $\boldsymbol{B} = (b_i)$ for each state i. In this thesis, only continuous HMM are of interest, where the HMM outputs are modeled through multivariate Gaussian density function $b_i(\boldsymbol{x}) = N(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. An output sequence is then of the form $\boldsymbol{X} = \boldsymbol{x}_1\boldsymbol{x}_2 \cdots \boldsymbol{x}_t \cdots \boldsymbol{x}_T$, where $\boldsymbol{x}_t$ denotes that the output is generated in the (discrete) time step t. The sequence length T is (generally) not predefined through the HMM, and can be different for each output sequence $\boldsymbol{X}$. The prior distribution of an initial state in an output sequence is defined by the pmf $\boldsymbol{\pi} = (\pi_i)$, i.e., $\pi_i = P(q_1 = i)$, where $q_{t=1}$ denotes the "current" state at the time step $t = 1$.

As mentioned in the introduction, an HMM is a generative model. An output sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$ can be drawn from the model [92] by generating step-by-step a state sequence $\boldsymbol{Q} = q_1 \cdots q_T$ with respect to the initial probabilities π_i and the transition probabilities a_{ij} and drawing for each state q_t an output $\boldsymbol{x}_t$ from the corresponding observation density $b_i(\boldsymbol{x})$. Generally, there is no unique correspondence between a given output sequence $\boldsymbol{X}$ and a state sequence, since different hidden state sequences $\boldsymbol{Q} = q_1 \cdots q_T$ can generate the same output sequence $\boldsymbol{X}$. Hence, the state variable q_t is latent (hidden), and only the posterior probability $P(q_t = i|\boldsymbol{\lambda}, \boldsymbol{X})$ that a state i is responsible for generating the output $\boldsymbol{x}_t$ can be inferred.

2.1.2 Notation

- $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ denotes an HMM, where
  - $\boldsymbol{A} = (a_{ij})$ is the transition matrix, where a_{ij} defines the probability of the state transition $i \to j$ (from state i to state j)
  - $\boldsymbol{B} = (b_i(\boldsymbol{x}))$ defines the output densities $b_i(\boldsymbol{x})$ of the states $i \in \mathcal{S}$
  - $\boldsymbol{\pi} = (\pi_i)$ defines the prior state distribution of the initial states ($\pi_i = p(q_1 = i)$)
- $\mathcal{S} = \{1, \ldots, N\}$ denotes the set of the states of an HMM with N state
- $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T = \boldsymbol{x}_1 \cdots \boldsymbol{x}_t \cdots \boldsymbol{x}_T$ denotes a multivariate output/observation sequence of length T, where each output $\boldsymbol{x}_t$ at time step t is a (multidimensional) vector

- $\mathcal{X} = \{\boldsymbol{X}^k | k = 1, \ldots, K\}$ a set of K output sequences $\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k$
- q_t denotes a latent state $q_t \in \mathcal{S}$ at the time step t
- $\boldsymbol{Q} = q_1 \ldots q_T$ a (hidden) state sequence

2.1.3 Related Problems

In the section above, only the definition of a (usual) HMM and the notation are given. This section discusses the three fundamental problems [92] and efficient algorithms that are an important part of the HMM framework. The implementation is discussed in the Sections 2.1.6 and 2.1.7. The three problems are the evaluation problem, the decoding problem, and the parameter estimation problem as stated below. The algorithmic approaches to the estimation problem and the evaluation problem are discussed in the Sections 2.1.5 and 2.1.4. Both problems are important in this thesis.

1. **Evaluation Problem.** *Given an observation sequence $\boldsymbol{X}$ and a model $\boldsymbol{\lambda}$, compute the likelihood/probability $\mathcal{L} = p(\boldsymbol{X}|\boldsymbol{\lambda})$, efficiently.* The value $\mathcal{L}$ can be interpreted in two different ways [92]. First, this can be seen from a generative viewpoint, where the probability $\mathcal{L}$ measures how likely the observation sequence $\boldsymbol{X}$ is produced by the model $\boldsymbol{\lambda}$. In the second viewpoint, the likelihood $\mathcal{L}$ is a score for how well the model $\boldsymbol{\lambda}$ matches the sequence $\boldsymbol{X}$. This interpretation is very useful if one has to choose among several models $\boldsymbol{\lambda}_i$. A method of evaluating the likelihood allows one to chose the model $\boldsymbol{\lambda}_i$ which matches the observation $\boldsymbol{X}$ best. *An efficient solution to the evaluation problem is given by the forward-backward algorithm discussed Section 2.1.4.2.*
2. **Decoding Problem.** *Given an observation sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$ and a model $\boldsymbol{\lambda}$, find a corresponding state sequence $\boldsymbol{Q} = q_1 \cdots q_T$ which is optimal in some sense.* For example, a state sequence that explains the observation best (i.e., a sequence which maximizes $p(\boldsymbol{Q}|\boldsymbol{\lambda}, \boldsymbol{X})$ [92]).

   It should be clear that there are generally many possible hidden state sequences that explain an output sequence. Thus, it is not possible to refer to "the" corresponding sequence. In some applications, a "decoded" state sequence $\boldsymbol{Q}$ has a meaningful interpretation. For example, in a text recognition application [39], where certain states or parts of the decoded sequence $\boldsymbol{Q}$ identify the glyphs.
3. **Parameters Estimation.** *Given a set of sequences $\mathcal{X} = \{\boldsymbol{X}^k\}$, find the model parameters of an HMM $\boldsymbol{\lambda}$ which maximize the the liklihood function $p(\mathcal{X}|\boldsymbol{\lambda})$.* The meaning of this is to find the parameters of the HMM such that the HMM matches the sequences best. The number of model states and the type of the output distributions are usually assumed to be predefined. The parameter estimation problem is stated above as a maximum likelihood (ML) problem. There exists no known method to calculate the optimal parameters directly [92]. An efficient iterative approach is the Baum-Welch expectation maximization algorithm, which is discussed in Section 2.1.5. The observation sequences used for the parameter estimation are also called training sequences, since they are used to adjust the model parameters, or in other words, to train the model. An alternative to the ML criterion is the MAP (maximum posterior) criterion [74]. Further criteria are discussed in [92].

2.1.4 Evaluation Problem

The forward-backward algorithm given in Section 2.1.4.2 is a means to efficiently evaluate $p(\boldsymbol{X}|\boldsymbol{\lambda})$ for a sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$ and an HMM $\boldsymbol{\lambda}$. The algorithm is also required by the Baum-Welch algorithm (Section 2.1.5). The notation introduced in Section 2.1.2 is used in the following. It is assumed that an HMM $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ and a sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$ are given. The purpose of the following section is to show that it is necessary to overcome the enumeration of state sequences in order to solve the evaluation problem efficiently.

2.1.4.1 Enumeration

A straight forward approach to the evaluation problem is the enumeration of all hidden state sequences $\boldsymbol{Q} = q_1 \cdots q_T$ of length T:

$$\begin{aligned}
\mathrm{P}(\boldsymbol{X}|\boldsymbol{\lambda}) &= \sum_{\boldsymbol{Q}} \mathrm{P}(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda}) && (2.1)\\
&= \sum_{\boldsymbol{Q}} \mathrm{P}(\boldsymbol{X}|\boldsymbol{Q}, \boldsymbol{\lambda})\, \mathrm{P}(\boldsymbol{Q}|\boldsymbol{\lambda}) && (2.2)\\
&= \sum_{\boldsymbol{Q}} \big(\pi_{q_1} \cdot \prod_{t=1}^{T} \mathrm{P}(\boldsymbol{x}_t|q_t, \boldsymbol{\lambda})\big)\big(\prod_{i=1}^{T-1} a_{q_i q_{i+1}}\big) && (2.3)\\
&= \sum_{\boldsymbol{Q}} \pi_{q_1} b_{q_1}(\boldsymbol{x}_1) a_{q_1 q_2} b_{q_2}(\boldsymbol{x}_2) a_{q_2 q_3} \ldots b_{q_T}(\boldsymbol{x}_T) && (2.4)
\end{aligned}$$

This enumeration is a rather poor approach, since it is, in principal, computationally infeasible for problems of reasonable size. For example, for $N \geq 10$ states and a sequence length $T \geq 40$, the number of state sequences $\boldsymbol{Q}$ is $\#\boldsymbol{Q} = N^T \geq 10^{20}$, and for each state sequence there is one summand in (2.4) that has to be evaluated. If one neglects the evaluation of the output distributions $b_{q_i}(\boldsymbol{x}_t)$ the enumerative approach results in an $\mathcal{O}(TN^T)$ algorithm.

2.1.4.2 Forward-Backward Procedure

The forward-backward procedure is a dynamic programming approach based on the so called forward and backward variables $\alpha_i(t)$ and $\beta_i(t)$, where i and t are a state number and a discrete time step. The calculation of $p(\boldsymbol{X}|\boldsymbol{\lambda})$ can be performed either on the forward variables $\alpha_i(t)$ or the backward variables $\beta_i(t)$, but both variables are required in the EM algorithm in Section 2.1.5. The variables are defined as

$$\alpha_i(t) \equiv p(\boldsymbol{x}_1 \cdots \boldsymbol{x}_t, q_t = i|\boldsymbol{\lambda}) = \left(\sum_{j=1}^{N} \alpha_j(t-1) a_{ji}\right) b_i(\boldsymbol{x}_t), \quad (2.5)$$

$$\beta_i(t) \equiv p(\boldsymbol{x}_{t+1} \cdots \boldsymbol{x}_T|q_t = i, \boldsymbol{\lambda}) = \sum_{j=1}^{N} a_{ij} b_j(\boldsymbol{x}_{t+1}) \beta_j(t+1). \quad (2.6)$$

The left-hand side of these equation defines the meaning of the variables [92], the right-hand side shows how these variables can be calculated recursively [92]. The recurrence on the right-hand side

seems to be quite evident, but the proof [14] requires a large set of conditional independence properties. The following values [92] can be used as a starting point to compute the variables:

$$\alpha_i(1) = p(\boldsymbol{x}_1, q_t = 1|\boldsymbol{\lambda}) = \pi_i b_1(\boldsymbol{x}_1), \tag{2.7}$$

$$\beta_i(T) = p(\sim |q_t = i, \boldsymbol{\lambda}) = 1. \tag{2.8}$$

Using $\alpha_i(1)$ and $\beta_i(T)$ as a starting point, all forward and backward variables can be calculated iteratively with the aid of the recurrences (2.5) and (2.6). The values of $\alpha_i(t)$ can be evaluated in the order $\alpha_1(t), \ldots, \alpha_N(t)$ for $t = 2, 3, \ldots, T$. Of course, the backward variables have to be evaluated in reverse order $t = T-1, T-2, \ldots, 1$. Once the forward variables have been evaluated, $p(\boldsymbol{X}|\boldsymbol{\lambda})$ can be calculated from

$$p(\boldsymbol{X}|\boldsymbol{\lambda}) = \sum_{i=1}^{N} \alpha_i(T), \tag{2.9}$$

which is basically marginalizing over the states $i = 1, \ldots, N$: $p(\boldsymbol{X}|\boldsymbol{\lambda}) = \sum_i p(\boldsymbol{X}, q_T = i|\boldsymbol{\lambda}) = \sum_i \alpha_i(T)$. The evaluation of $p(\boldsymbol{X}|\boldsymbol{\lambda})$ through the forward/backward procedure requires $\mathcal{O}(TN^2)$ operations [92], and additionally, $\mathcal{O}(NT)$ evaluations of the output density functions $b_i(\boldsymbol{x}_t)$. The $\mathcal{O}(TN^2)$ operations are required to evaluate the sums in Equation (2.5)/(2.6) for the forward/backward variables (there are $\mathcal{O}(TN)$ forward/backward variables).

2.1.5 Baum-Welch Algorithm

The Baum-Welch algorithm (Baum et al. 1970 [9]) is an iterative method that allows one to estimate the parameters of an hidden Markov model given a (set of) observation sequence(s). In [92], the update formulas are derived primarily based on the classic work of Baum and colleagues. The update formulas can also be derived within the framework of the EM algorithm [92]. The Baum-Welch algorithm is equivalent with the EM algorithm for HMMs [12].

The variables are chosen in the following notes on the Expectation Maximization (EM) algorithm on the basis of the notation which was introduced for HMMs. However, the EM algorithm is, of course, a general concept. The EM algorithm is an iterative method of [18] finding a maximum likelihood estimate

$$\boldsymbol{\lambda}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\lambda}} p(\boldsymbol{X}|\boldsymbol{\lambda}) \tag{2.10}$$

for some parameter set $\boldsymbol{\lambda}$ of a statistical model and some data set $\boldsymbol{X}$. The EM algorithm becomes particularly useful in the case of missing data or when the ML estimate can not be solved analytically [11]. In the latter case, it can be helpful to introduce some latent/hidden/missing variables and to assume that some values or data $\boldsymbol{Q}$ is given for these variables. The complete data likelihood $p(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda})$ can then become tractable (as for HMMs). The data $\boldsymbol{X}$ together with $\boldsymbol{Q}$ is called the complete data set.

The EM algorithm is an iterative method. In each iteration the model parameters of the previous iteration $\boldsymbol{\lambda}'$ are refined to new values $\boldsymbol{\lambda}$. Each iteration of the EM algorithm is a two step procedure; namely, the expectation (E) step, and maximization (M) step. In the E step [18], the latent variables are estimated based on the model parameters $\boldsymbol{\lambda}'$ and the data $\boldsymbol{X}$. In the M step [18], the maximization

of the likelihood is performed under the assumption that the latent parameters are known. The new model parameters $\boldsymbol{\lambda}$ are computed by maximizing the $\mathcal{Q}$-function [14]

$$\mathcal{Q}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{\boldsymbol{Q}} p(\boldsymbol{Q}|\boldsymbol{X}, \boldsymbol{\lambda}') \ln p(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda}) \tag{2.11}$$

with respect to $\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}'$ is assumed to be fixed. Comprehensive explanations of the EM algorithm are given in [14, 11, 18].

In the following, a sketch of the derivation [13] of the Baum-Welch update formulas in the EM framework is given. The hidden Markov model (HMM) $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ is here assumed to be an HMM with continuous outputs, where the output density $b_i(\boldsymbol{x})$ of each state i is modeled by a single multivariate Gaussian density (see Appendix E) with a covariance matrix $\boldsymbol{\Sigma}_i$ and mean $\boldsymbol{\mu}_i$, i.e., $b_i(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\Sigma}_i, \boldsymbol{\mu}_i)$. For the training it is assumed that a single observation sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$ is given. In addition, initial values $\boldsymbol{\lambda}'$ of the HMM parameters need to be given. This includes the values $(\pi_1, \ldots, \pi_N) = \boldsymbol{\pi}$, $(a_{ij}) = \boldsymbol{A}$, and the values for the matrices $\boldsymbol{\Sigma}_i$ and vectors $\boldsymbol{\mu}_i$. It is worthwhile to note that an iteration of the EM algorithm does not change the structure of the HMM, e.g., the number N of states is not changed.

2.1.5.1 The Expectation Step

In the E step, the following posterior distributions are calculated for the hidden (latent) state sequence $\boldsymbol{Q} = q_1 \cdots q_T$:

$$\gamma_i(t) = p(q_t = i|\boldsymbol{X}, \boldsymbol{\lambda}'), \tag{2.12}$$

$$\xi_{ij}(t) = p(q_t = i, q_{t+1} = j|\boldsymbol{X}, \boldsymbol{\lambda}'). \tag{2.13}$$

The value $\gamma_i(t)$ is the posterior probability that the state i is active at time step t given the state sequence $\boldsymbol{X}$ and the model $\boldsymbol{\lambda}'$. The value $\gamma_i(t)$ is called in the following *responsibility* [14] (of state i at step t). Similarly, $\xi_{ij}(t)$ is the posterior probability that the state transition $i \to j$ is used at the time step $t \to t+1$.

The values $\gamma_i(t)$ and $\xi_{ij}(t)$ can be computed by using the forward and backward variables $\alpha_i(t)$ and $\beta_i(t)$ (see Section 2.1.4.2), which need to be computed in advance for the observation sequence $\boldsymbol{X}$ and the current model parameters $\boldsymbol{\lambda}'$. By the use of the Markovian conditional independence [13]

$$p(\boldsymbol{X}, q_t = i|\boldsymbol{\lambda}') = p(\boldsymbol{x}_1 \cdots \boldsymbol{x}_t, q_t = i|\boldsymbol{\lambda}')p(\boldsymbol{x}_{t+1} \cdots \boldsymbol{x}_T|q_t = i, \boldsymbol{\lambda}'), = \alpha_i(t)\beta_i(t)$$

the responsibility $\gamma_i(t)$ can be written as

$$\gamma_i(t) = \frac{p(\boldsymbol{X}, q_t = i|\boldsymbol{\lambda}')}{p(\boldsymbol{X}|\boldsymbol{\lambda}')} = \frac{p(q_t = i|\boldsymbol{X}, \boldsymbol{\lambda}')}{\sum_{i=1}^{N} p(q_t = i|\boldsymbol{X}, \boldsymbol{\lambda}')} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{i=1}^{N} \alpha_i(t)\beta_i(t)}. \tag{2.14}$$

Similarly, the posterior transition probabilities $\xi_{ij}(t)$ can be written (see [13]) as

$$\xi_{ij}(t) = \frac{\gamma_i(t) a_{ij} b_j(\boldsymbol{x}_{t+1}) \beta_j(t+1)}{\beta_i(t)}.$$

2.1.5.2 The Maximization Step

The first concern is to calculate the log-likelihood function ($\ln p(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda})$) of the $\mathcal{Q}$-function. For $p(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda})$, one gets [14]:

$$\begin{aligned} p(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda}) &= p(q_1|\boldsymbol{\pi}) \left(\prod_{t=1}^{T-1} p(q_{t+1}|q_t, \boldsymbol{A}) \right) \left(\prod_{t=1}^{T} p(\boldsymbol{x}_t|q_t, \boldsymbol{B}) \right) \\ &= \pi_{q_1} \left(\prod_{t=1}^{T-1} a_{q_t, q_{t+1}} \right) \left(\prod_{t=1}^{T} b_{q_t}(\boldsymbol{x}_t) \right). \end{aligned}$$

The log-likelihood function is then

$$\ln p(\boldsymbol{X}, \boldsymbol{Q}|\boldsymbol{\lambda}) = \ln \pi_{q_1} + \left(\sum_{t=1}^{T-1} \ln a_{q_t, q_{t+1}} \right) + \left(\sum_{t'=1}^{T} \ln b_{q_t}(\boldsymbol{x}_t) \right).$$

The $\mathcal{Q}$-function becomes then

$$\begin{aligned} \mathcal{Q}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = & \sum_{\boldsymbol{Q}} p(\boldsymbol{Q}|\boldsymbol{X}, \boldsymbol{\lambda}') \ln \pi_{q_1} \\ & + \sum_{\boldsymbol{Q}} p(\boldsymbol{Q}|\boldsymbol{X}, \boldsymbol{\lambda}') \sum_{t=1}^{T-1} \ln a_{q_{t-1}, q_t} \\ & + \sum_{\boldsymbol{Q}} p(\boldsymbol{Q}|\boldsymbol{X}, \boldsymbol{\lambda}') \sum_{t=1}^{T} \ln b_{q_t}(\boldsymbol{x}). \end{aligned}$$

By the use of following two ways for marginalizing

$$\sum_{\boldsymbol{Q}} p(\boldsymbol{Q}|\boldsymbol{X}, \boldsymbol{\lambda}') = \sum_{i=1}^{N} p(q_1 = i|\boldsymbol{X}, \boldsymbol{\lambda}') = \sum_{i=1}^{N} \gamma_i(t), \tag{2.15}$$

$$\sum_{\boldsymbol{Q}} p(\boldsymbol{Q}|\boldsymbol{X}, \boldsymbol{\lambda}') = \sum_{i=1}^{N} \sum_{i=1}^{N} p(q_t = i, q_{t+1} = j|\boldsymbol{X}, \boldsymbol{\lambda}') = \sum_{i=1}^{N} \sum_{j=1}^{N} \xi_{ij}(t), \tag{2.16}$$

the $\mathcal{Q}$-function can be written as (see [14]):

$$\mathcal{Q}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{i=1}^{N} \gamma_i(t) \ln \pi_i \tag{2.17}$$

$$+ \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_{ij}(t) \ln a_{ij} \tag{2.18}$$

$$+ \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma_i(t) \ln b_i(\boldsymbol{x}_t). \tag{2.19}$$

In this formulation of the $\mathcal{Q}$-function the quantities $\gamma_i(t)$ and $\xi_{ij}(t)$ are constant, since they depend only on the model parameters $\boldsymbol{\lambda}'$ and the observation sequence $\boldsymbol{X}$. The model parameters $\boldsymbol{\lambda}'$ are assumed to be constant in the M step of the EM algorithm. Thus, the maximization has to be carried out only with

respect to the new parameters $\boldsymbol{\lambda} = (\boldsymbol{A} = (a_{ij}), \boldsymbol{B} = (b_i(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \boldsymbol{\pi} = (\pi_i))$. Since each of the summands (2.17), (2.18), and (2.19) acts only either on π_i, or a_{ij}, or $(\mu_i, \boldsymbol{\Sigma}_i)$, each summand can be maximized separately. The maximization of each term is a constrained optimization problem. The constraints $\sum_i \pi_i = 1$ and $\sum_j a_{ij} = 1$ have to be preserved in (2.17) and (2.18), since π and $\boldsymbol{A}$ are a stochastic vector and stochastic matrix. For the first term (2.17), the precondition of an optimal value is:

$$\boldsymbol{\nabla}_{\binom{\boldsymbol{\pi}}{\delta}} \left(\sum_{i=1}^{N} \gamma_i(1) \ln \pi_i + \delta(1 - \sum_{i=1}^{N} \pi_i) \right) = \mathbf{0},$$

where $\delta \in \mathbb{R}$ is the Lagrange multiplier [14]. It follows

$$\gamma_i(1)\frac{1}{\pi_i} - \delta = 0, \qquad \sum_{i=1}^{N} \pi_i - 1 = 0.$$

The left equation can be resolved to $\pi_i = \gamma_i(1)/\delta$ which can then be substituted in the right equation:

$$\sum_{i=1}^{N} \frac{\gamma_i(1)}{\delta} - 1 = 0,$$

This gives the value of delta: $\delta = \sum_i \gamma_i(1)$. By substituting δ in the equation $\pi_i = \gamma_i(1)/\delta$, the update formula for π_i is derived:

$$\pi_i = \frac{\gamma_i(1)}{\sum_{i=1}^{N} \gamma_i(1)}.$$

The term (2.18) can be regarded as N summands ($i = 1, \ldots, N$). Each summand is:

$$\sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_{ij}(t) \ln a_{ij} = \sum_{j=1}^{N} \underbrace{\left(\sum_{t=1}^{T-1} \xi_{ij}(t) \right)}_{\hat{\gamma}_j^i} \ln a_{ij} = \sum_{j=1}^{N} \hat{\gamma}_j^i \ln a_{ij}.$$

The term on the right has the same structure as the term (2.17) (with j in the place of i) and has to comply to the same constraint: $\sum_j a_{ij} = 1$ (with j in the place of i). By re-using the update formula (2.20), one gets the update formula for a_{ij}:

$$a_{ij} = \frac{\hat{\gamma}_j^i}{\sum_{j=1}^{N} \hat{\gamma}_j^i} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_{ij}(t)} \equiv \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)},$$

where the last equation step uses the equality $\sum_{j=1}^{N} \xi_{ij}(t) = \gamma_i(t)$, which can be seen from the definitions of $\gamma_i(t)$ and $\xi_{ij}(t)$.

The third term (2.19) has the same form as the terms which one gets in the derivation for estimating the parameters of a Gaussian mixture distribution for an i.i.d. data set [13]. Derivations are given in [13, 14]. The update formulas [14] for the Gaussians' means $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$ are:

$$\mu_i = \frac{\sum_{t=1}^{T} \gamma_i(t) \boldsymbol{x}_t}{\sum_{t=1}^{T} \gamma_i(t)},$$
$$\Sigma_i = \frac{\sum_{t=1}^{T} \gamma_i(t) (\boldsymbol{x}_t - \mu_i)(\boldsymbol{x}_t - \mu_i)^\top}{\sum_{t=1}^{T} \gamma_i(t)}.$$

2.1.5.3 Multiple Observation Sequences

In this thesis the training of the HMMs is performed on the basis of a set $\mathcal{X} = \{\boldsymbol{X}^k\}_{k=1}^K$ of multiple observation sequences $\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k$, which are assumed to be independent. The likelihood function that is maximized by the EM algorithm becomes then:

$$p(\mathcal{X}|\boldsymbol{\lambda}) = \prod_{k=1}^{K} p(\boldsymbol{X}^k|\boldsymbol{\lambda}) \equiv \prod_{k=1}^{K} p_k. \tag{2.20}$$

In the E-step of the EM algorithm, all the quantities (as the $\alpha_i(t)$, $\beta_i(t)$, $\gamma_i(t)$, and $\xi_{ij}(t)$) have to be computed for each observation sequence $\boldsymbol{X}^k$. These quantities are therefore denoted as $\alpha_i^k(t)$, $\beta_i^k(t)$, $\gamma_i^k(t)$, and $\xi_{ij}^k(t)$. The following update formulas of the M-step are given on the basis of [92].

$$\pi_i = \frac{\sum_{k=1}^K \frac{1}{p_k} \gamma_i^k(1)}{\sum_{k=1}^K \frac{1}{p_k} \sum_{i=1}^N \gamma_i^k(1)}, \tag{2.21}$$

$$a_{ij} = \frac{\sum_{k=1}^K \frac{1}{p_k} \sum_{t=1}^{T_k-1} \xi_{ij}^k(t)}{\sum_{k=1}^K \frac{1}{p_k} \sum_{t=1}^{T_k-1} \gamma_i^k(t)}, \tag{2.22}$$

$$\mu_i = \frac{\sum_{k=1}^K \frac{1}{p_k} \sum_{t=1}^{T_k} \gamma_i^k(t) \boldsymbol{x}_t^k}{\sum_{k=1}^K \frac{1}{p_k} \sum_{t=1}^{T_k} \gamma_i^k(t)}, \tag{2.23}$$

$$\Sigma_i = \frac{\sum_{k=1}^K \frac{1}{p_k} \sum_{t=1}^{T_k} \gamma_i^k(t) (\boldsymbol{x}_t^k - \mu_i)(\boldsymbol{x}_t^k - \mu_i)^\top}{\sum_{k=1}^K \frac{1}{p_k} \sum_{t=1}^{T_k} \gamma_i^k(t)} \tag{2.24}$$

2.1.6 Implementing the HMM Framework

It is assumed that a set $\mathcal{X} = \{\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k\}_{k=1}^K$ of training sequences and initial values for a HMM $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B} = (b_i), \boldsymbol{\pi})$ are given, where each Gaussian density $b_i(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is defined by its mean $\boldsymbol{\mu}_i \in \mathbb{R}^D$ and covariance matrix $\boldsymbol{\Sigma}_i \in \mathbb{R}^{D \times D}$. The outputs have to be vectors $\boldsymbol{x}_t^k \in \mathbb{R}^D$.

In the following, the implementation of the EM steps for the training of the model is discussed. Also the computation of the likelihoods $p(\boldsymbol{X}^k|\lambda)$ is discussed. The likelihoods need to be computed in the E-Step. The likelihood $p(\mathcal{X}|\lambda)$ can be used, for example, as a stopping criterion for the iteration process.

E-Step. To compute the forward and backward variables, the probabilities $b_i(\boldsymbol{x}_t^k) = \mathcal{N}(\boldsymbol{x}_t^k|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ for each $k \in \{1, \ldots, K\}$, $k \in \{1, \ldots, K\}$ and each Gaussian density $i \in \{1, \ldots, N\}$ need to be evaluated. The evaluation of $\mathcal{N}(\boldsymbol{x}_t^k|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ requires (see Section E.1) the computation of the determinant $\lambda_i = \det(\boldsymbol{\Sigma}_i)$ and the computation of the Mahalanobis distance (Equation (E.2)) which includes the computation of inverse matrix $\boldsymbol{B}_i = {\boldsymbol{\Sigma}_i}^{-1}$ of the covariance matrix $\boldsymbol{\Sigma}_i$ (besides some usual arithmetic operations). This is performed by using the Cholesky decomposition $\boldsymbol{\Sigma}_i = \boldsymbol{L}_i \boldsymbol{L}_i^\top$ which is discussed in Section E.2. The Cholesky decomposition and the computation of the inverse $\boldsymbol{S}_i = \boldsymbol{L}_i^{-1}$ (Section E.2) are efficient tools to compute the inverse of a symmetric positive definite matrix (as $\boldsymbol{\Sigma}_i$). The inverse matrix is $\boldsymbol{B}_i = \boldsymbol{\Sigma}_i^{-1} = \boldsymbol{S}_i^\top \boldsymbol{S}_i$ (see Equation (E.10)). However, the inverse need not be computed explicitly as shown in the following.

For each Gaussian $i = 1, \ldots, N$ the followings steps are performed: the matrix $\boldsymbol{L}_i = (l^i_{jk})$ is computed (Section E.2.1), the determinant $\lambda_i = \det(\boldsymbol{\Sigma}_i) = (\prod_j l^i_{jj})^2$ is computed (see Equation (E.4)), and finally $\boldsymbol{S}_i$ is computed (Section E.2.2).

For each k and t the value of $b_i(\boldsymbol{x}^k_t) = \mathcal{N}(\boldsymbol{x}^k_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is computed by: a) computing the Mahalanobis distance

$$\hat{d}^k_i(t) \equiv d_{\boldsymbol{\Sigma}_i}(\boldsymbol{x}^k_t, \boldsymbol{\mu}_i) = \sqrt{(\boldsymbol{x}^k_t - \boldsymbol{\mu}_i)^\top (\boldsymbol{S}_i^\top \boldsymbol{S}_i)(\boldsymbol{x}^k_t - \boldsymbol{\mu}_i)} = \sqrt{(\boldsymbol{S}_i(\boldsymbol{x}^k_t - \boldsymbol{\mu}_i))^2}, \tag{2.25}$$

and, b) computing $b_i(\boldsymbol{x}^k_t) = \mathcal{N}(\boldsymbol{x}^k_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ on the basis of $\hat{d}^k_i(t)$ and the determinant λ_i.

Based on the values $b_i(\boldsymbol{x}^k_t)$ the forward variables $\alpha^k_i(t)$ can be computed (Section 2.1.4.2 and Section 2.1.5.3). However, an important issue is that the values $b_i(\boldsymbol{x}^k_t)$ can be very small or very large. As a consequence the values $\alpha^k_i(t)$ can become really small or large, since $\alpha^k_i(t)$ is basically a product of the values $b_i(\boldsymbol{x}^k_t)$ (a discussion is given in [92]). The values $\alpha^k_i(t)$ can easily overflow or underflow the range of a floating point number (`double`) of a computer. The values $\alpha^k_i(t)$ can become then 0 or ∞, which can result in a devision by zero when the posterior distributions are computed (e.g., in Equation (2.14)). One approach to handling this problem is to introduce scaling values [92]. However, this approach can still have issues [82]. Another approach is to use an implementation of floating point numbers with arbitrary precision. In this thesis an implementation of numbers is used which represent a number through a `double` and an integer, where the integer captures large and small exponents. This number type is used for all critical operations.

Based on the forward variables $\alpha^k_i(t)$ the likelihoods $p_k = p(\boldsymbol{X}_k|\boldsymbol{\lambda})$ can be computed (Equation (2.9)) if necessary. Small changes of the log-likelihood $\ln p(\mathcal{X}|\boldsymbol{\lambda}) = \ln \prod_k p_k$ during the iterations of the EM algorithm can be used as a stopping criterion. However, the procedure (as discussed so far) is a means to solve the evaluation problem, where one is interested in the probabilities of some sequence(s) given the HMM λ.

For the M-Step, also the backward variables $\beta^k_i(t)$ need to be computed (Section 2.1.4.2 and Section 2.1.5.3).

M-Step. Based on the forward and backward variables, the posterior probabilities $\gamma^k_i(t)$ and $\xi^k_i(t)$ are computed (Section 2.1.5.2 and Section 2.1.5.3). Finally, all model parameters π_i, a_{ij}, μ_i and $\boldsymbol{\Sigma}_i$ are updated based on the update formulas given in Section 2.1.5.3.

2.1.7 Run Time

In the following, the run time requirements are analyzed for the implementation in the section above, where each state has a single Gaussian output density. It is assumed that a model $\boldsymbol{\lambda}$ and K observation sequences $\boldsymbol{X} = \boldsymbol{x}^k_1 \cdots \boldsymbol{x}^k_{T_k}$ are given. The dimensionality of the outputs $\boldsymbol{x}^k_t$ is D. The maximal length of the output sequences is denoted as $T = \max_k T_k$.

The ***Evaluation Problem***, i.e. computing the probabilities $p_k = p(\boldsymbol{X}|\boldsymbol{\lambda})$ and $p(\mathcal{X}|\boldsymbol{\lambda})$, requires the following operations: The matrix inversions of the covariance matrices requires $\mathcal{O}(D^3N)$ operations. The evaluation of the values $b_i(\boldsymbol{x}^k_t)$ requires $\mathcal{O}(KTND^2)$ operations. The evaluation of the forward variables $\alpha^k_i(t)$ requires $\mathcal{O}(KTN^2)$ operations. The whole evaluation problem requires then

$$\mathcal{O}(D^3N + KTND^2 + KTN^2) \tag{2.26}$$

operations. If one assumes that $D \leq T$, then one gets

$$\mathcal{O}(KTN(N + D^2)). \tag{2.27}$$

One ***EM Iteration***, requires all operations of the evaluation problem. However, in the $\mathcal{O}$-notation one get the same results as for the evaluation problem. The reason is that the computation of the values $\beta_i^k(t)$, $\gamma_i^k(t)$ and ξ_{ij}^k and the updates of π_i and a_{ij} are all bound by $\mathcal{O}(KTN^2)$. The updates of μ_i and $\boldsymbol{\Sigma}_i$ are bound by $\mathcal{O}(KTND^2)$.

2.2 Utilizing the Hidden Markov Model

The hidden Markov model is introduced in Section 2.1 as a finite state machine which is extended in a probabilistic way. It is denoted as a triple $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ and has a fixed number of states $i = 1, \ldots, N$. The vector $\boldsymbol{\pi} = (\pi_i)_{i=1}^N$ defines the prior distribution of the initial states, and the matrix $\boldsymbol{A} = (a_{ij})_{i,j=1}^N$ defines the transition probabilities a_{ij} from a state i to a state j. In the following, the model is assumed to generate and represent continuous sequences, where each output is a D-dimensional continuous vector $\boldsymbol{x} \in \mathbb{R}^D$. The output distributions $\boldsymbol{B} = (b_i)$ are assumed to be continuous multivariate Gaussian distributions, where each density $b_i(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is defined by its mean $\boldsymbol{\mu}_i \in \mathbb{R}^D$ and covariance matrix $\boldsymbol{\Sigma}_i \in \mathbb{R}^{D \times D}$.

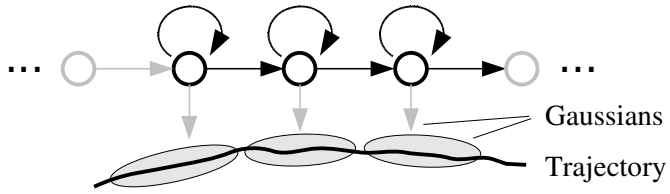


Figure 2.2: Left-Right HMM. The figure shows a left-right HMM with continuous outputs. A possible output of this model could be a sampled version of the depicted trajectory. Since a hidden sequence has to pass the states in the order from left to right, each state of the model (trained for the depicted trajectory) is basically responsible for a certain part of the trajectory.

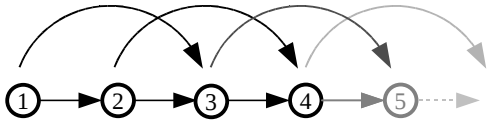


Figure 2.3: Left-Right HMM. The figure shows the transition structure of a certain left-right HMM. This left-right model has no self-transitions but skip-transitions.

2.2.1 Ergodic and Left-Right HMMs

Depending on the state transition structure (given by the transition matrix $\boldsymbol{A} = (a_{ij})$), HMMs are classified, e. g., as ergodic or fully connected, or as left-right HMM. In a fully connected HMM [92] each state could be reached in a single step from every other state, i. e. each transition probability is

greater than zero ($a_{ij} > 0$). In a strict definition [92], an ergodic HMM requires only that each state can be reached in a finite number of steps from every other state.

In a left-right model [93], the transition matrix has the property that $a_{ij} = 0$ for $j < i$. This means that all the state sequences $\boldsymbol{Q} = q_1 \cdots q_N$ (with a nonzero probability)[1] have the property that the state number along the sequence can only increase or stay in the same state ($q_{t+1} \geq q_t$).

Of course, one can define very different left-right HMMs concerning the transition structure. Two distinct left-right HMM structures are shown in the Figures 2.2 and 2.3. The left-right model in Figure 2.2 has only self-transitions (transitions to the state itself ($a_{ii} > 0$)) and transitions to the direct successors of each state ($a_{i(i+1)} > 0$), and all other transitions $i \rightarrow j$ have a probability $a_{ij} = 0$. The model with a transition structure, shown in Figure 2.3, has no self-transitions. In addition to the transitions to the direct successor of each state, the model has skip-transitions, which permit a state sequence (with nonzero probability) to leave some states out. The model in Figure 2.2 is referred to as no-skip (left-right) HMM. Other examples of left-right HMMs are given in [92].

In the case of a time sequence $\boldsymbol{X}$, the left-right structure allows one to understand more easily how the underlying state sequences $\boldsymbol{Q}_i$ of the sequence $\boldsymbol{X}$ with a high likelihood $p(\boldsymbol{X}|\boldsymbol{Q}_i, \boldsymbol{\lambda})$ may look. This is especially the case for the no-skip HMM as illustrated in Figure 2.2. An underlying state sequence has to pass the states from left to right. In order to have a high likelihood the sequence has to use the states with appropriate output distributions for a sample of the sequence.

2.2.2 Utilizing the HMM Training Procedure

The training of the model can be performed based on a training set $\mathcal{X} = \{\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k\}$ via the Baum-Welch algorithm (Sections 2.1.5 and 2.1.6). The Baum-Welch algorithm is an application of the expectation maximization framework (Section 2.1.5), and maximizes the likelihood function $\mathcal{L}(\boldsymbol{\lambda}) = P(\mathcal{X}|\boldsymbol{\lambda})$. The algorithm requires that the number N of states is predefined and requires furthermore initial values for all the parameters π_i, a_{ij}, $\boldsymbol{\mu}_i$, and $\boldsymbol{\Sigma}_i$. The algorithm reestimates then the parameters in each iteration to increase the likelihood $\mathcal{L}(\boldsymbol{\lambda})$. This procedure is guaranteed to converge, but not guaranteed to find the global maximum of the likelihood $\mathcal{L}(\boldsymbol{\lambda})$ [92]. This means that the estimated parameters π_i, a_{ij}, $\boldsymbol{\mu}_i$, and $\boldsymbol{\Sigma}_i$ are not always the optimal solution.

As a consequence, the initialization of the parameters is crucial. In [92] (Section C), it is stated that there is no straight forward answer to this question; and experiments have shown that for either uniform or randomly chosen initial values for $\boldsymbol{A}$ and π lead to useful reestimates of these values. However, the initial values for the output distributions are essential in the case of continuous output distributions [92]. A proposed method in [92] is the manual segmentation of the data and assigning the data to the states. Another problem mentioned in [92] (Section D) is insufficient data, which could be solved by increasing the number of training examples or by decreasing the number of states.

A common problem known from the maximum likelihood estimation of Gaussian mixtures is that a Gaussian component of the mixture has after each iteration the responsibility for less and less data points and tends to model only a single or too few data points [14]. Since the responsibilities of the Gaussian component for the other data points are in this case basically zero, this can result (due to the finite machine precision) in a covariance matrix of the mixture component that is numerically

[1] Obviously, all state sequences are considered in the HMM framework. However, those state sequence $\boldsymbol{Q}$ with a probability $p(\boldsymbol{Q}|\boldsymbol{A}) = 0$ are unimportant. Thus, it makes sense to address the important sequences with a probability greater than zero as "the state sequences".

singular. An approach [83] to handling the problem of singular covariance matrices is to put a prior on the Gaussian distributions. A simple approach [83] is to add a matrix $\boldsymbol{\Lambda} = \sigma\mathbf{I}$ to the estimated covariance matrices. In the HMM framework [84], this is approached by adding the matrix $\boldsymbol{\Lambda} = \sigma^2\mathbf{I}$ (with $\sigma = \textit{constant}$) to each covariance matrix after each iteration of the Baum-Welch algorithm. A covariance matrix $\boldsymbol{\Sigma}$ is then replaced by $\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} + \sigma^2\mathbf{I}$. For the training of the HMMs/PHMMs in this thesis, it turned out (for the used motion data) that the singularities occurred only in the first iterations of the Baum-Welch algorithm when $\sigma = 0$. Thus, the training is performed in this thesis with a covariance prior σ, but the value of σ is decreased after some iterations and finally set to zero.

The usual training procedure for left-right HMMs/PHMMs in this thesis (Section 2.2.3.1) makes also use of a finish state $N+1$ (Section 2.2.3.1) and time restrictions of the number of state outputs of each state (Section 2.2.3.2).

2.2.3 Training of Left-Right HMMs

A left-right HMM can be trained by using an initial transition matrix $\boldsymbol{A} = (a_{ij})$, where the transitions $i \to j$, which are supposed to be zero in the finally trained model, are initialized with zero ($a_{ij} = 0$). The probabilities a_{ij} of these transitions stay zero in each iteration of the Baum-Welch training procedure [92].

The following consideration regards the initialization of the prior distribution $(\pi_1, \ldots, \pi_N)$ of the states. It is assumed that a set of trajectories $\{\boldsymbol{x}^k(t)\}$ (represented through sequences) is given, where each trajectory is a repetition of basically the same movement of a person. In the left-right model with no skip-transitions (as shown in Figure 2.2) it is reasonable to assume that consecutive states model basically consecutive (temporal) parts of each trajectory $\boldsymbol{x}^k(t)$ (as shown in Figure 2.2). Since the trajectories start in basically the same body pose, it seems to be reasonable to assume that the first part of the movements is modeled by the first state $i = 1$. In this case, it is reasonable to have a model with a prior state distribution $\boldsymbol{\pi}$ with $\pi_1 = 1$ and $\pi_i = 0$ for $i > 1$. This can be accomplished by using an initial vector $\boldsymbol{\pi} = (1, 0, \ldots, 0)$ for the training procedure. These values are not changed by the Baum-Welch algorithm, since the values $\boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_N$ initialized with zero.

2.2.3.1 Training with a Finish State

Following the above argument that the first state of a state sequence (with nonzero probability) has to begin in the first state $i = 1$, it seems reasonable (if one assumes that the movement trajectories also end in the same body pose) to enforce each state sequence to end in the last state $i = N$. To ensure this in the implemented HMM framework, an additional *dummy/finish* state $i = N+1$ is added to the HMMs and each training sequence $\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k$ is extended by an additional dummy sample $\boldsymbol{x}_{T_k+1} \equiv \mathbf{0}$. In the training procedure and the forward-backward procedure of the HMM framework, the output "density" of the state $i = T+1$ is defined to be $b_{N+1}(\boldsymbol{x}_t^k) \equiv 0$ for $t \leq T_k$ and $b_{N+1}(\boldsymbol{x}_t^k) \equiv 1$ for $t = T_k + 1$. For all the other states $i < N+1$ the output density is defined as $b_i(\boldsymbol{x}_{T_k+1}) \equiv 0$ for the dummy sample $\boldsymbol{x}_{T+1}$. This has the effect that only such a hidden state sequence $Q = q_1 \cdots q_{T_k+1}$ is important (i.e., $p(\boldsymbol{Q}|\boldsymbol{\lambda}, \boldsymbol{X}^k) > 0$) when a) the state sequence ends in the dummy state (i.e., $q_{T+1} = T+1$) and b) q_{T_k} is a state for which the transition probability $a_{q_{T_k},(N+1)}$ to the additional dummy/finish state $i = N+1$ is nonzero. By initializing, e.g., $a_{j,N+1} = 0$ for $j < N$ and $a_{j,N+1} = 1$ for $j = N$, each important hidden state sequence "has to end in the state N", i.e., $q_T = N$

and $q_{T+1} = N + 1$.

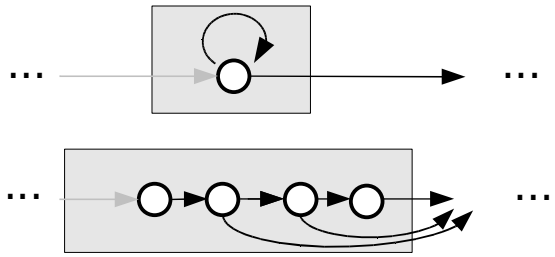


Figure 2.4: Explicit Time Restrictions. The figure shows a simple way to introduce time restrictions to a state of a left-right HMM. Therefore each state of the HMM is replaced by several sub-states which share the same output distribution. In the figure this procedure is shown for a single state of a left-right HMM without skip-transitions (as shown Figure 2.2). On the top, a state of the left-right HMM is shown. The state is replaced here by 4 sub-states with the transition structure as shown by the gray box at the bottom. These 4 sub-states share the same Gaussian output distribution. The single state of the left-right HMM (at the top) has to generate at least one output (for a hidden state sequence) and can generate also many outputs. The 4 sub-states can only generate either 2, 3, or 4 outputs from their shared output distribution. The structure (of the last 3 sub-states) is known as Fergusson topology [106].

2.2.3.2 Training with Explicit Time Restrictions

The no-skip left-right HMM (Figure 2.2) permits a state to generate many consecutive outputs but also only a single output. If the model is used to generate sequences (Method A, Section 2.2.4), then some states can contribute several outputs to one sequence and many outputs to other sequences. Since the training procedure maximizes the likelihood of the model for a given set of sequence, the maximization can then rely on this capability of the HMM. A possibility to restrict the number of outputs which a state can generate in a state sequence (with nonzero probability) is given by an extension of the hidden Markov model, which is known as expanded state HMM (ESHMM).

In the ESHMM [106], a state is identified with a sub-HMM which has only a single output distribution. To achieve this in the HMM framework of this thesis, each state of a no-skip left-right HMM (Figure 2.2) is replaced by a number of sub-states which share the same output distribution. The transition structure of the sub-states of a replaced state constraints the number of outputs which can be generated from the shared distribution. The Figure 2.4 gives an example of how a single state is replaced, so that the sub-states have to generate (in a state sequence with nonzero probability) at least 2 and maximal 4 outputs.

In the following, it is explained how the time (or output) restrictions of the ESHMM are established in this thesis for a given no-skip left-right HMM $\boldsymbol{\lambda}$ with N states. The number of outputs of each state is constrained in the following to a minimal number d_{min} and maximal number d_{max} of outputs. Of course, one could use different constrains for each state, but this is not considered here.

Each state i of the HMM $\boldsymbol{\lambda}$ is replaced by $R = d_{\text{max}}$ sub-states $i'_1, \ldots, i'_R$, which are supposed to share a single output distribution. This results in an HMM $\boldsymbol{\lambda}'$ with $N' = N \cdot R$ (sub-)states. The identifier $i'_r \in \{1, \ldots, N'\}$ of the r-th sub-state of the replaced state i is $i'_r = (i-1)R + r$.

The transition structure of the sub-states $i'_1, \ldots, i'_R$ of the replaced state i is then chosen such that a state sequence (with nonzero probability) through the sub-states has to generate at least d_{min} and at most d_{max} outputs. The transition matrix $\boldsymbol{A}' = (a'_{i'j'})$ is initialized for the sub-states i'_r of a state i as follows: $a'_{i'_r,i'_{r+1}} = 1$ for $r = 1, \ldots, d_{\text{min}} - 1$, and $a'_{i'_r,i'_{(r+1)}} = 0.5$ and $a'_{i'_r,(i'_R+1)} = 0.5$ for $r = d_{\text{min}}, \ldots, d_{\text{max}} - 1$, and $a'_{i'_r,(i'_R+1)} = 1$ for $r = d_{\text{max}} = R$. All the other outgoing transitions $i'_r \to j'$ from the sub-states of the replaced state i are initialized with zero ($a_{i'_r j'} = 0$). Figure 2.4 shows the transitions (with nonzero probability) given by the initialization with $d_{\text{min}} = 2$ and $d_{\text{max}} = 4$.

To accomplish that the sub-states $i'_1, \ldots, i'_R$ of a replaced state i share the same output distribution, the usual HMM framework (where each state has a single multi-variate Gaussian output distribution) is used, but after each iteration of the Baum-Welch training procedure the means $\boldsymbol{\mu}'_{i'_j}$ and covariance matrices $\boldsymbol{\Sigma}'_{i'_j}$ of the sub-states $i'_1, \ldots, i'_R$ are unified. Therefore, the averages $\bar{\boldsymbol{\mu}}_i = \sum_{r=1}^{R} \boldsymbol{\mu}'_{i'_r}/R$ and $\bar{\boldsymbol{\Sigma}}_i = \sum_{r=1}^{R} \boldsymbol{\mu}'_{i'_r}/R$ are calculated and used to replace the means $\boldsymbol{\mu}'_{i'_r}$ and covariance matrices $\boldsymbol{\Sigma}'_{i'_r}$ of the sub-states, i.e. $\boldsymbol{\mu}'_{i'_r} := \bar{\boldsymbol{\mu}}_i$ and $\boldsymbol{\Sigma}'_{i'_r} := \bar{\boldsymbol{\Sigma}}_i$ for $r = 1, \ldots, R$. This is performed for the sub-states of each replaced state i. In the next training iteration, each set $\{i'_1, \ldots, i'_R\}$ of sub-states shares then the same output distribution.

In the following such an HMM will be addressed simply as an N state HMM, with explicit time durations or (output) restrictions (with minimal and maximal output durations d_{min} and d_{max}), where each set $\{i'_1, \ldots, i'_R\}$ of sub-states (with $R = d_{\text{max}}$) is identified as a single state i. Each identified state i has a single Gaussian output distribution $b_i(\boldsymbol{x})$ which is shared by the sub-states $i'_1, \ldots, i'_R$. The true number of states of the underlying HMM is of course $N' = N \cdot R$.

2.2.4 Synthesis

In order to synthesize or generate a sequence from an HMM one can use different methods which suit different purposes. The Method A is the basic method to generate an output sequence. This method follows the notion of the HMM as a generative model. If one uses the HMM to model smooth movements, then this method does not seem to be very suited to synthesize a smooth movement sequence (see [56]). The subsequent methods B, C, and D try to overcome this drawback of Method A. The method D is used to generate prototype movements in this thesis.

Method A. An output sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$ of length T can be drawn from the model [92] by generating a state sequence $\boldsymbol{Q} = q_1 \cdots q_T$ with respect to the initial probabilities π_i and the transition probabilities a_{ij} and drawing for each state q_t the output $\boldsymbol{x}_t$ from the corresponding output distribution $b_{q_t}(\boldsymbol{x})$. In detail, first $q_1 \in \{1, \ldots, N\}$ is chosen with respect to the probabilities $\pi_1, \ldots, \pi_N$, and $\boldsymbol{x}_1$ is drawn from the distribution $b_{q_1}(\boldsymbol{x})$. For $t = 2, \ldots, T$, a state $q_t \in \{1, \ldots, N\}$ is chosen with respect to the outgoing transition probabilities $a_{q_{t-1},1}, \ldots, a_{q_{t-1},N}$ from the previous state $t - 1$, and the output $\boldsymbol{x}_t$ is drawn from $b_{q_t}(\boldsymbol{x})$.

Method B. Another approach to synthesize a smooth output sequence from an HMM is to generate a large set of output sequences (by applying the Method A) and to use an averaging scheme as it is proposed, for example, in [56]. The Method B has been proposed in [56] in order to generate movements on humanoid robots.

Method C. First a state sequence $\boldsymbol{Q} = q_1 \cdots q_T$ is generated in the same way as in Method A.

State repetitions are then removed from the sequence, which results in a possibly shorter sequence $\boldsymbol{Q}' = q'_1 \cdots q'_{T'}$ with $T' \leq T$. For this sequence, an output sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_{T'}$ is generated by drawing the outputs $\boldsymbol{x}_t$ from the corresponding state distributions $b_{q'_t}(\boldsymbol{x})$. The outputs are then used as control points for spline interpolation to generate a new sequence. To contribute the state repetitions in the original state sequence $\boldsymbol{Q}$, the parameterization of the spline is chosen such that the number of samples that are generated close to a control point $\boldsymbol{x}_t$ are related to the state repetitions which have been collapsed for q'_t. The Method C has been proposed in [132] for the synthesis human movements from a parametric HMM (Chapter 3). However, a parametric HMM is for fixed parameters a usual HMM (Chapter 3).

Method D. For this method it is assumed that the HMM has a left-right transition structure as shown in Figure 2.2 with self-transitions but without skip-transitions and a prior state distribution $\pi = (1, 0, \ldots, 0)$. If one is now interested in generating a prototype movement from the left-right HMM, then it is seems to be reasonable to use the sequence of the state means $\boldsymbol{\mu}_1 \cdots \boldsymbol{\mu}_N = \boldsymbol{X}$. It is worthwhile to note that in the considered left-right HMM a state sequence (of nonzero probability) has to pass the states in the order $1, 2, 3, \ldots$ (of course, a state can be visited several times in repetition). In the sense of a prototype movement the choice of the mean $\boldsymbol{\mu}_i$ for each state i is reasonable, since this is the expectation of the outputs generated by the state distribution $b_i(\boldsymbol{x})$. The sequence $\boldsymbol{X}$ could then be interpolated by using splines.

In order to account for state repetitions that a hidden state sequence can include, one could follow, e.g., the approach of Method C, where the spline parameterization accounts for the state repetitions in a generated state sequence. However, in the sense of prototype movements, it would be rather inappropriate to use a single generated state sequence to calculate the number of repetitions of a state. As for the outputs, it seems to be reasonable to use the expected number of repetitions/outputs of each state in a state sequence.

Here, one can consider two different approaches: 1. One generates a large set of state sequences and computes the average number of outputs of each state. 2. One calculates the expected number of outputs of each state. (3. One uses another approach to training.)

1. The usual procedure (see Method A) to produce a state sequence is to define a sequence length T. Then, one generates the sequence based on the transition probabilities starting from the state $i = 1$ (it is assumed that $\pi = (1, 0, \ldots, 0)$). However, this has two strange aspects. First, one predefines the length T. Second, for a left-right topology without skip transitions, one would not reach always the last state. If one assumes that a finish state $N + 1$ is used, then it seems to be rational to use the following approach to generate a state sequence. One starts with the state $q_1 = 1$ and selects the following states $q_2, q_3, \ldots$ with respect to the transition probabilities and stops when the finish state $N + 1$ has been reached. Since the HMM is assumed to have no skip transitions, the generated sequence passes each state once (with a certain number of repetitions) in the order $1, 2, 3, \ldots$. However, the number of repetitions of a state i depends then only on the transition probabilities a_{ij} of the state i, i.e. the self-transition probability a_{ii} and the probability $a_{i(i+1)} = 1 - a_{ii}$. Consequently, it makes sense to calculate directly the expected number of repetitions of a state rather than approximating it through generating a large number of state sequences.

2. *Expected Number of Outputs of a State.* The probability of generating a single output from the state i is obviously $p_i(1) = a_{i(i+1)} = 1 - a_{ii}$. The probability of generating (exactly) two outputs is

$p_i(2) = a_{ii}(1 - a_{ii})$. The probability of generating d outputs is $p_i(d) = (a_{ii})^{n-1}(1 - a_{ii})$ [93]. The expected number of outputs of state i is then [93]:

$$\bar{d}_i \equiv \mathbb{E}[p_i(d)] = \sum_{d=1}^{\infty} d\, p_i(d) = \sum_{1}^{\infty} d\, (a_{ii})^{n-1}(1 - a_{ii}) = \frac{1}{(1 - a_{ii})}.$$

The value $\bar{d}_i$ could be also understood as the effective time duration of state i.

In the case of a no-skip N state left-right HMM with explicit time restrictions d_{min} and d_{max}, the number of outputs of a state i is restricted by the transition structure of the sub-states $i'_1, \ldots, i'_R$ ($R = d_{\text{max}}$), as discussed in Section 2.2.3.2. The transition structure is stored in a matrix $(a'_{i'j'}) = \boldsymbol{A}' \in \mathbb{R}^{(NR)\times(NR)}$, where $N \cdot R$ is the number of sub-states. The probability that the state i generates d outputs is $p_i(d) = 0$ for $d < d_{\text{min}}$ or $d > d_{\text{max}}$, and $p_i(d) = a_{i'_d,(i'_R+1)} \prod_{r=d_{\text{min}}}^{d-1} a'_{i'_r i'_{(r+1)}}$ for $d = d_{\text{min}}, \ldots, d_{\text{max}}$, where i'_r denotes the corresponding row/column index of the r-th sub-state of state i in the matrix $(a_{i'j'})$. The expected number of outputs of state i is then:

$$\bar{d}_i = \sum_{d=d_{\text{min}}}^{d_{\text{max}}} d\, p_i(d). \tag{2.28}$$

3. An alternative approach to reproduce the "time durations" of each state would be to expand the samples of each training sequence by a component which includes a time stamp. The state means of the trained HMM would provide then also time labels.

2.2.5 Recognition

In HMM-based recognition, one is interested in calculating the likelihood $\mathcal{L} = p(\boldsymbol{X}|\boldsymbol{\lambda})$ for an observation $\boldsymbol{X}$ and a model $\boldsymbol{\lambda}$ as this gives a hint whether the sequence belongs to the class represented by the model $\boldsymbol{\lambda}$. This so called evaluation problem is discussed in Section 2.1.3. An efficient algorithm for calculating the likelihood $\mathcal{L}$ is given by the forward-backward procedure (Section 2.1.4.2).

A usual scenario for HMM-based recognition or classification is the following: A certain number of sequence classes is given, and for each sequence class k a representative data set $\mathcal{X}^k$ is given. In the training phase, one trains for each class k an HMM $\boldsymbol{\lambda}^k$ by using the Baum-Welch training procedure. In the recognition phase, a novel output sequence $\boldsymbol{X}$ can then be classified by identifying the model $\boldsymbol{\lambda}^k$ which yields the highest likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^k)$. The sequence is then classified as belonging to the class k.

If also a movement $\boldsymbol{X}$ can be given which belongs to none of the classes, it seems to be reasonable to use some threshold which must be reached for a model in order to accept the movement. A more sophisticated acceptance test is given in [88], which is explained in the following for a single HMM $\boldsymbol{\lambda}$: First, the probability distribution $p_{\boldsymbol{\lambda}}(\mathcal{L})$ of the log-likelihood for sequences generated from the model is estimated. Therefore, a large sample set of output sequences $\boldsymbol{X}$ is generated from the model $\boldsymbol{\lambda}$, and is used to calculate the empirical distribution of the log-likelihood $\mathcal{L} = \ln p(\boldsymbol{X}|\boldsymbol{\lambda})$. A novel sequence $\boldsymbol{X}'$ with $L' = \ln p(\boldsymbol{X}'|\boldsymbol{\lambda})$ is then rejected to be drawn from the model if $p_{\boldsymbol{\lambda}}(\mathcal{L} \leq L') < \textit{threshold}$.

However, in any case, it seems to be important that the model is a good model for the data set it is trained on (measured in terms of the likelihood) and that the model is not a good model for sequences which do not belong to the modeled class, since otherwise such a sequence can be falsely classified as belonging to the class.

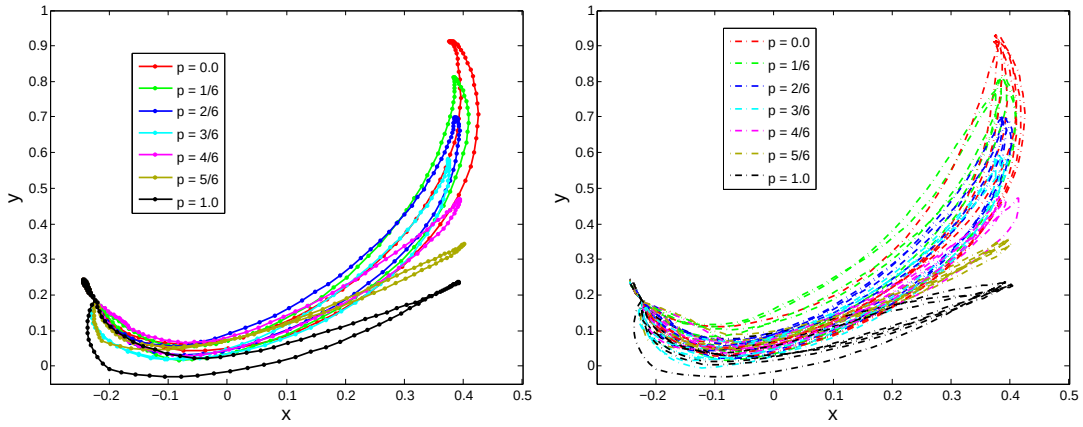


Figure 2.5: Motion Data of a Parametric Human Movement. The plots show trajectories of a recorded parametric human movement that is performed for 7 different values of the parametrization. The recorded person sits in front of a table-top and performs pointing actions to different table-top locations. The plots show the 2D projection of the right hand's index-finger tip. Each finger trajectory is starting from a point $(x, y) \approx (-0.2, 0.2)$, and approaches a target point at the table-top (where $x \approx 0.4$) and then returns to starting point. Since the target points are on a line, one can chose a single parameter $p \in \mathbb{R}$ to parameterize this point. In the plots, the value $p = 0.0$ belongs to the target point of the red trajectories. The value $p = 1.0$ belongs to the target of the black trajectories. As discussed in Section 1.2.1, it makes sense to assume that there exists a parametric prototype movement $\boldsymbol{x}^p(t)$ ($\boldsymbol{x} = (x, y)$), where each movement of the person is an imprecise performance of the prototype movement. The plot on the left is showing one performance of the parametric movement $\boldsymbol{x}^p(t)$ for each parameter $p \in \{0.0, 1/6, 2/6, \ldots, 1.0\}$. The plot on the right is showing 4 repetitions for each parameter.

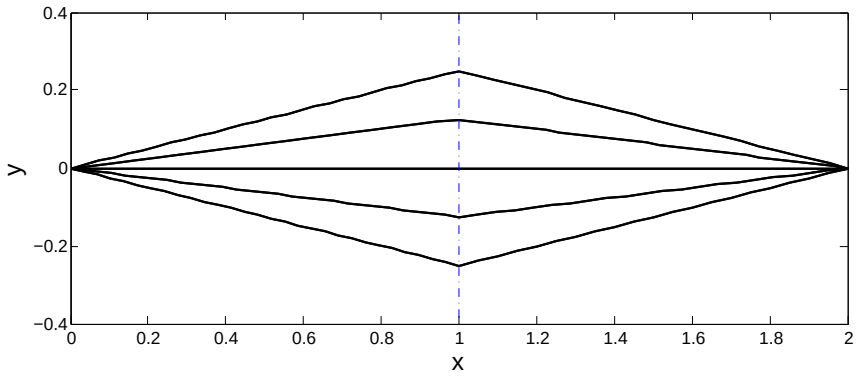


Figure 2.6: Parametric Prototype Movement. The plot shows the (synthetic) parametric movement $\hat{\boldsymbol{x}}^p(t)$ for different parameters $p = -0.25, -0.125, 0, 0.125, 0.25$.

2.3 Training on a Synthetic Movement Set

In this section, different types of hidden Markov models are trained on a synthetic movement set. The movement set is kept simple but exhibits some important features of the human movements that are

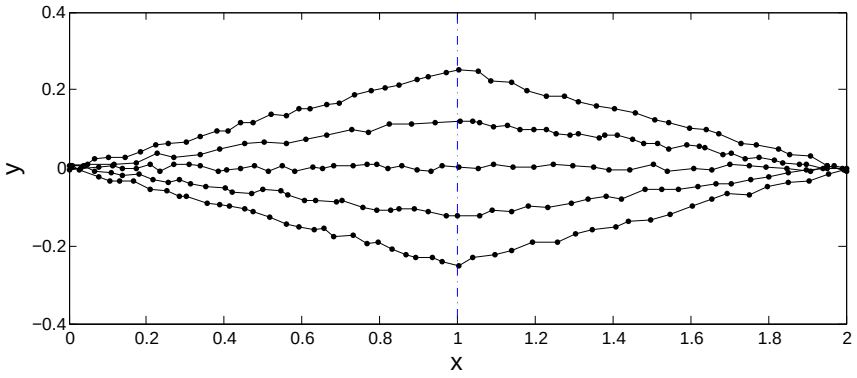


Figure 2.7: Parametric Movement. The plot shows the sampled parametric movement $\hat{\boldsymbol{x}}^p(t)$ for parameters $p = -0.25, -0.125, 0, 0.125, 0.25$, each sample $\boldsymbol{x} = (x, y)$ is disturbed by noise. For the components x and y the noise is drawn from the interval $[-0.005, 0.005]$ of length 0.01.

considered in this thesis.

In Figure 2.5, some recorded performances of a parametric human movement are depicted. The parametric movement is parameterized by a single parameter $p \in \mathbb{R}$. Depending on this parameter, the performances vary spatially. However, each repetition of the movement for a fixed parameter p still varies slightly. It is worthwhile to note that all the performances of the movement are somehow similar in shape, and the trajectories are always very smooth. An aspect which can not be seen in the figure is that the trajectories vary also in their progression rates as discussed in Section 1.2.1

For the generation of the synthetic movement data, a (synthetic) parametric prototype movement is used. The 2D movement trajectory is given by

$$\hat{\boldsymbol{x}}^p(t) = (x^p(t), y^p(t)) = \begin{cases} (t, p \cdot t) & : \quad t \leq 1 \\ (t, p \cdot (2 - t)) & : \quad t > 1, \end{cases}$$

where the time $t \in [0, 2]$ and the parameterization $p \in \mathbb{R}$ are scalars. The class of the parametric movement is defined as $\mathcal{X} = \{\hat{\boldsymbol{x}}^p(t) : p \in [a, b]\}$. In the following experiments the interval is chosen as $[a, b] = [-0.25, 0.25]$. Figure 2.6 shows some instances of the parametric prototype movement class $\mathcal{X}$. Each instance of the movement class is smooth (as discussed for the human movement trajectories above), but the movement class shows some variation. This variation can be understood in two different ways: First, one can understand this movement class as a set of different performances of a given movement prototype (which is, e.g., $\hat{\boldsymbol{x}}^{p=0}(t)$), where the variation is the inaccuracy of the human performer. In this case, an interesting point would be to find some prototype of the performances, where the variation is modeled as noise. Second, one can regard this class as a set of instances of a parametric movement, where one would be interested in modeling the variation depending on the parameter(s). This is particularly important when the parameter(s) defines the meaning/effect of the movement.

For the HMM training (performed in the following), a training set is generated which comprises some sampled versions of the parametric movement $\hat{\boldsymbol{x}}^p(t)$. The samples are disturbed slightly by noise, as it is shown and discussed in Figure 2.7. To contribute different progression rates of human

movements, the trajectories are not sampled uniformly in time. For a given number T of samples, a number of (randomly chosen) $T_1 \in [T/2(1-\alpha), T/2(1+\alpha)] \cap \mathbb{N}$ samples is used for the first part ($t \in [0,1]$). The remaining samples $T_2 = T - T_1$ are used for the second part.

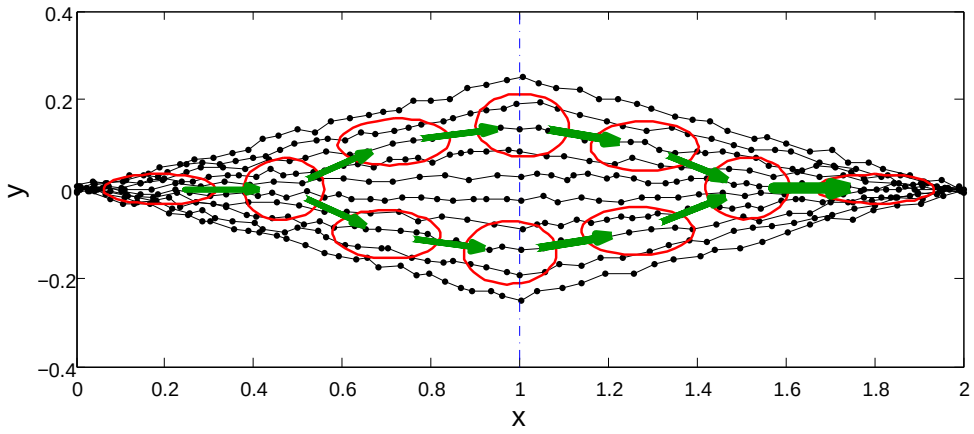


Figure 2.8: Ergodic 10-State HMM: Training Result.

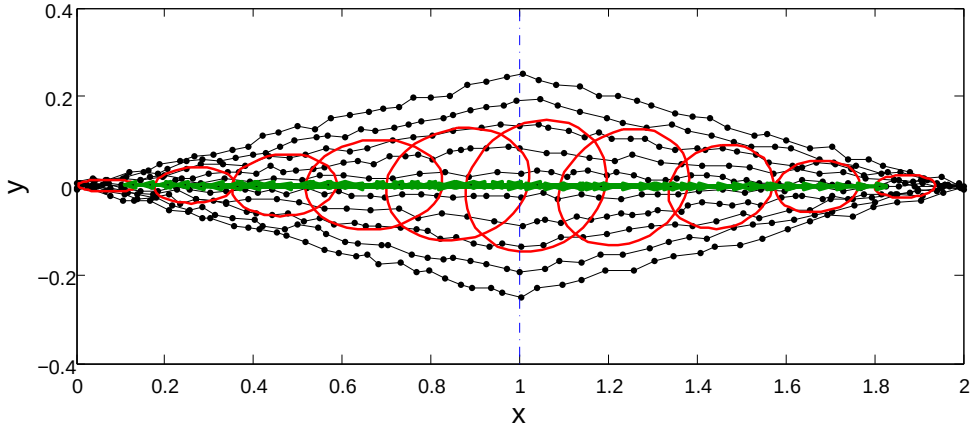


Figure 2.9: Ergodic 10-State HMM: Initialization.

2.3.1 Ergodic HMM

The transition structure of the ergodic HMM is the most general. Such an HMM should be, due to its generality, the best usual HMM model for learning the structure within the parametric movement set.

Figure 2.8 shows the result of a training trial for the ergodic HMM. The covariance matrices and means of the Gaussian output distributions are shown by the red ellipsoids. To display the Gaussians, the method discussed in Section E.2.3 is used. The transitions $i \to j$ with a probability $a_{ij} > 0.01 \cdot p_{\max}$ are shown, where $p_{\max}$ is the maximal probability of the transitions excluding self-transitions. The thickness of a line indicates the probability. The thickness scales (as for all plots in the following) basically with $a_{ij}/p_{\max}$. However, in the plot of Figure 2.8, the probabilities of the transitions between the states are very similar. Self-transitions are not shown in this plot and the following plots. The

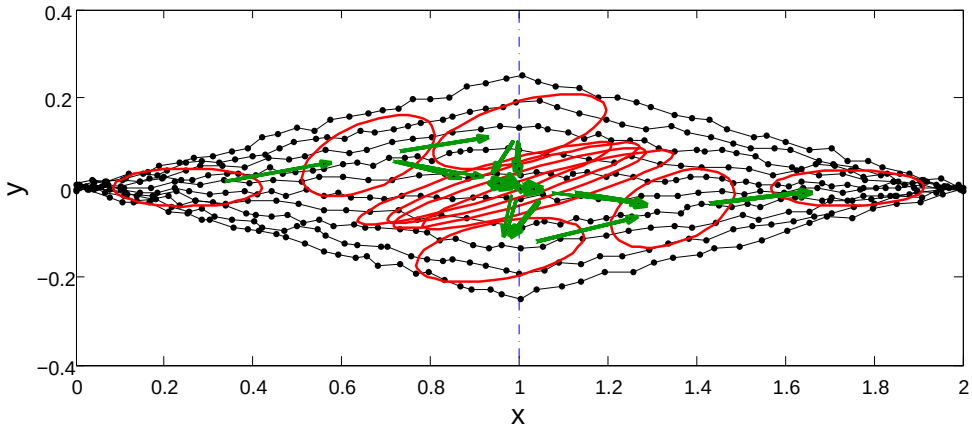


Figure 2.10: Ergodic 10-State HMM: 20 Iterations.

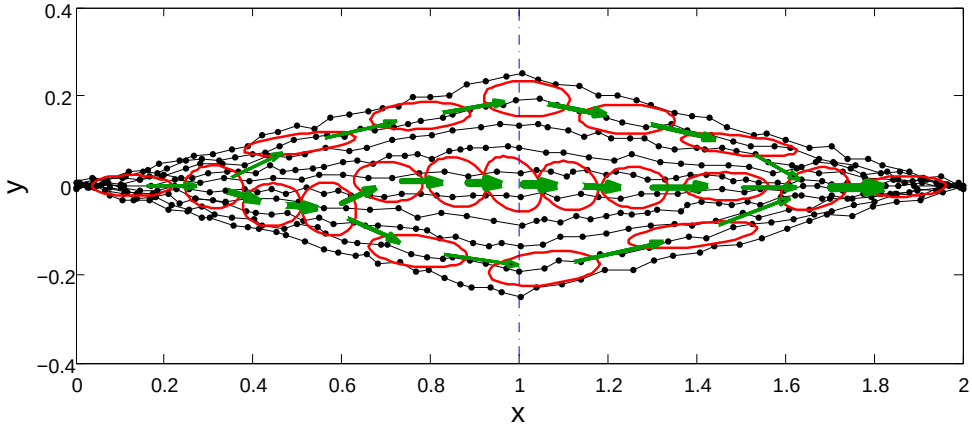


Figure 2.11: Ergodic 20-State HMM: Training Result.

HMM has $N = 10$ states, and is trained on 10 training sequences of the parametric movement, where the length of each sequence is $T = 50$ (with $\alpha = 0.3$). The initialization of the HMM is shown in Figure 2.9, where the prior distribution and the outgoing state transition probabilities a_{ij} (of each state i) are chosen uniformly and no finish state $N + 1$ (as discussed in Section 2.2.3.1) is used. For the initialization of output distributions, the training sequences are cut into N segments. The mean and covariance matrix of each state i are then calculated based on the i-th segment of the sequences. The convergence of the Baum-Welch algorithm is for this initialization very slow. About 70 iterations are required. The Figure 2.8 shows the result after 100 iterations. After 20 iterations (see Figure 2.10) the model still has not much in common with the final result. In Figure 2.11, the training result of an ergodic HMM with $N = 20$ states is shown after 100 iterations. The initialization of the 20 state HMM was done in the same way as for the 10 state HMM. However, the training results with a random initialization of the means and covariance matrices were for the 10 state and the 20 state HMM very similar.

To some extent, the 20 state HMM in Figure 2.11 is capable of generating certain sequences which are similar to the sequences of the training data. However, it is easy to generate a zig-zag sequence

$\boldsymbol{X}$ by following the branch in the middle and sampling from the Gaussians. This sequence is not very similar to the training data but would result in a high likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda})$ of the HMM. This also means that the model would recognize such a sequence as a sequence of the parametric movement class $\mathcal{X}$. To overcome this problem, one could then try to increase the number of states to a huge number in order to allow the model to learn the intrinsic systematic of the movement class. But finally one would achieve for a huge number of states and a small training set that the model simply learns each sequence of the training data (over fitting). This has the effect that the model becomes an inappropriate model for the underlying parametric movement class $\mathcal{X}$, since it does not generalize to sequences which belong to the movement class but not to the training set. The only approach is then to increase also the number of training sequences. However, this is somehow pointless, since the parametric movement varies continuously depending on the parameter $p \in [a, b]$ and a usual HMM has no mechanism to learn the systematic of this variation. (The parametric HMM (Chapter 3) has a mechanism to learn the systematic.) In the case that one is capable of training an HMM (provided a huge training set and computation time), the model suffers still in three points: 1. It is (without other means) not possible to recognize the parameter p of an instance of the parametric movement class. 2. There is no means to synthesize a sequence for a given parameter p. 3. For the synthesis of a single prototype movement of the whole class, only an averaging scheme as Method B seems to be suitable. Even if one takes, for example, the means of the Gaussians of the middle branch in Figure 2.11, the first part of the sequence would not be very similar to a training sequence. However, instead of using the Method B, the synthesis of a single prototype movement can be also achieved on the basis of a simple left-right model. Experiments for the left-right model are provided in the following section.

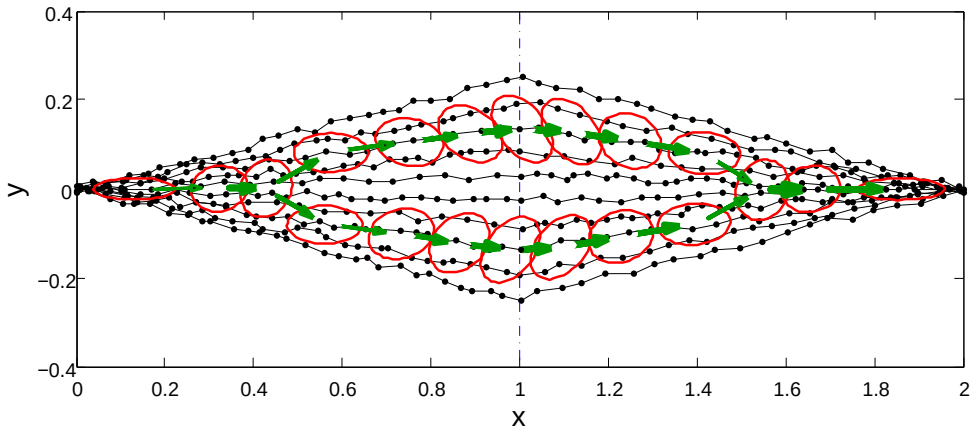


Figure 2.12: Left-Right 20-State HMM with Skip-Transitions: Result.

2.3.2 Left-Right HMM

In this section, left-right HMMs are trained again for the synthetic movement class $\mathcal{X}$. The intention is here to train the HMMs for the movement class in order to synthesize a single prototype of the class (as by Method D, Section 2.2.4), where the variation within this class is considered as noise. The intention is not to learn all the specific instances of the class. The training settings are very similar to the section above: There are 10 training sequences. Each sequence has $T = 50$ samples (if not stated) and has different progression rates ($\alpha = 0.3$). The initialization of the covariance matrices

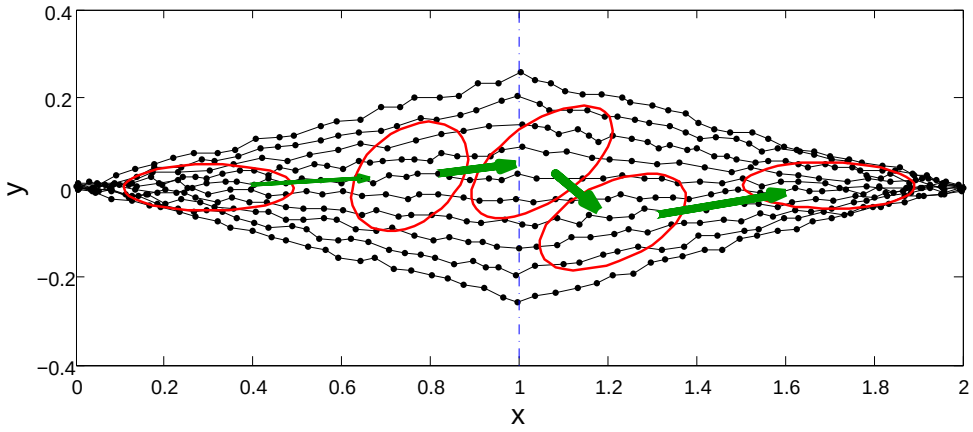


Figure 2.13: Left-Right 5-State HMM: Result.

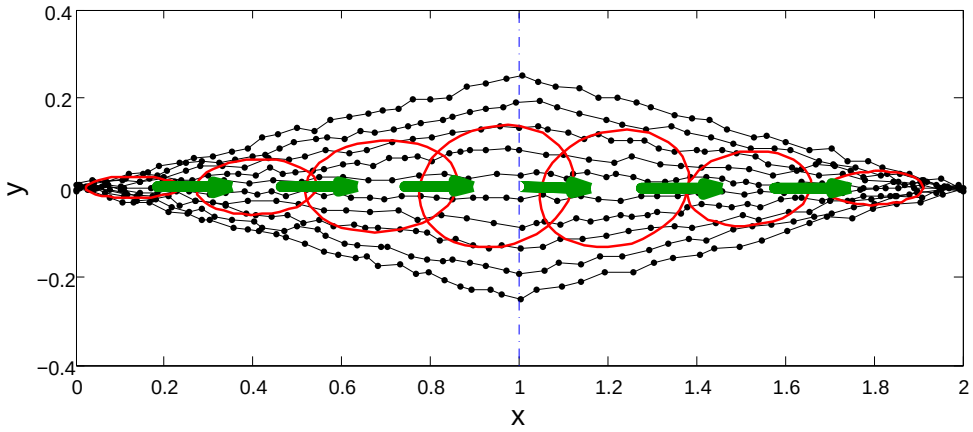


Figure 2.14: Left-Right 7-State HMM: Initialization.

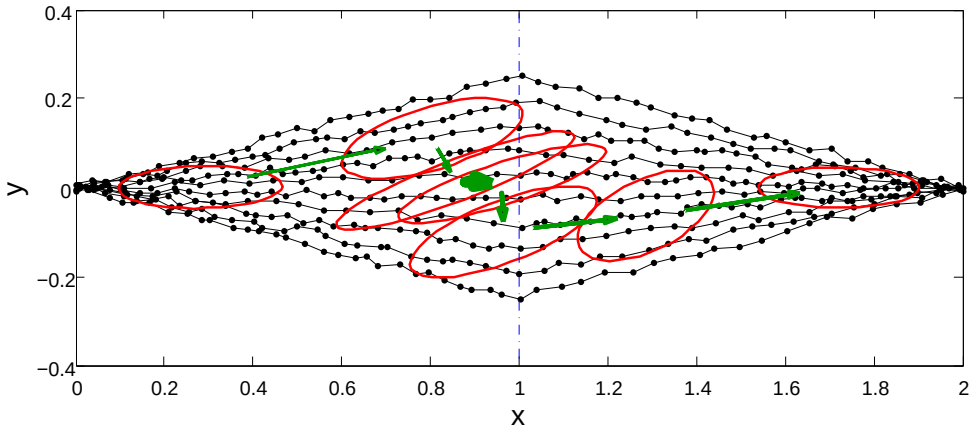


Figure 2.15: Left-Right 7-State HMM: Result.

and means of the HMMs are performed as in the section above by cutting the sequences into N parts (where N is the number of HMM states) and calculating the covariance matrices and means of the

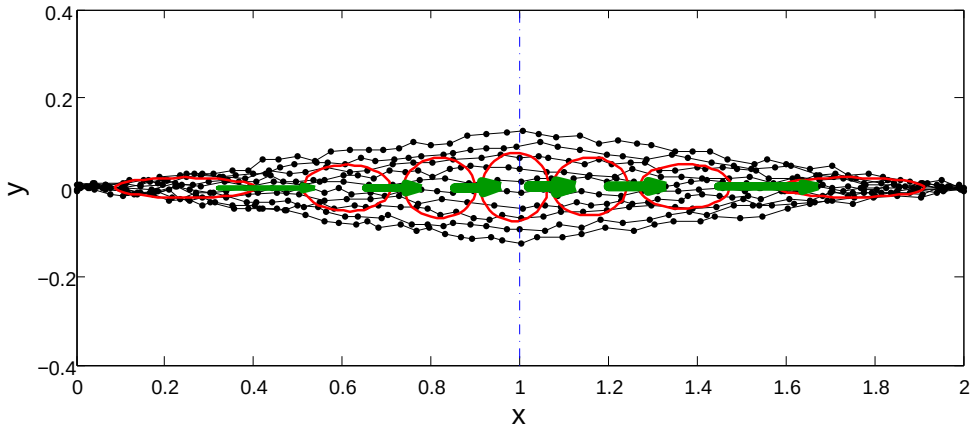


Figure 2.16: **Left-Right 7-State HMM: Result.**

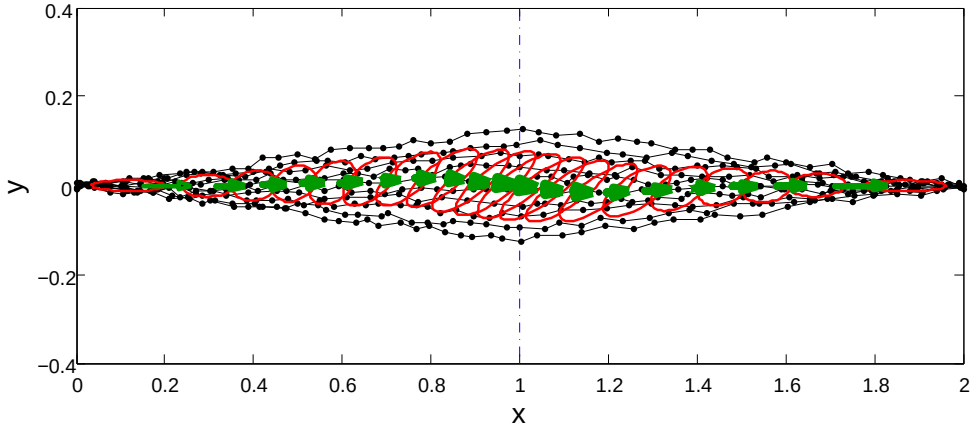


Figure 2.17: **Left-Right 20-State HMM: 10 Iterations.**

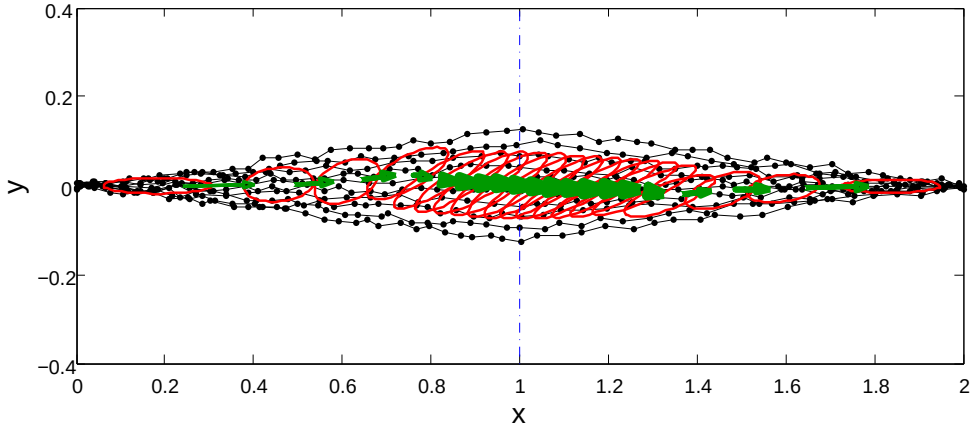


Figure 2.18: **Left-Right 20-State HMM: Result.**

corresponding parts for each state. This seems to be especially reasonable for the left-right HMM, since the spacial variation along the y-axis of the trajectories is considered here as noise. The prior

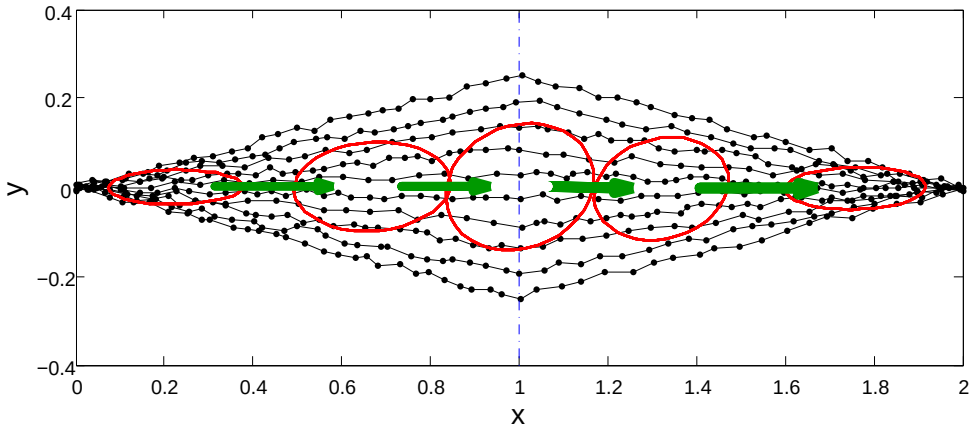


Figure 2.19: Left-Right 5-State HMM with Time Restrictions: Result.

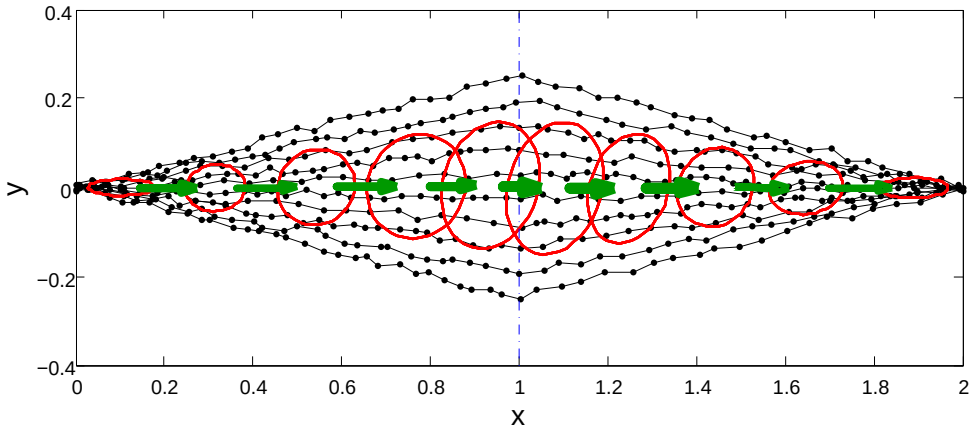


Figure 2.20: Left-Right 10-State HMM with Time Restrictions: Result.

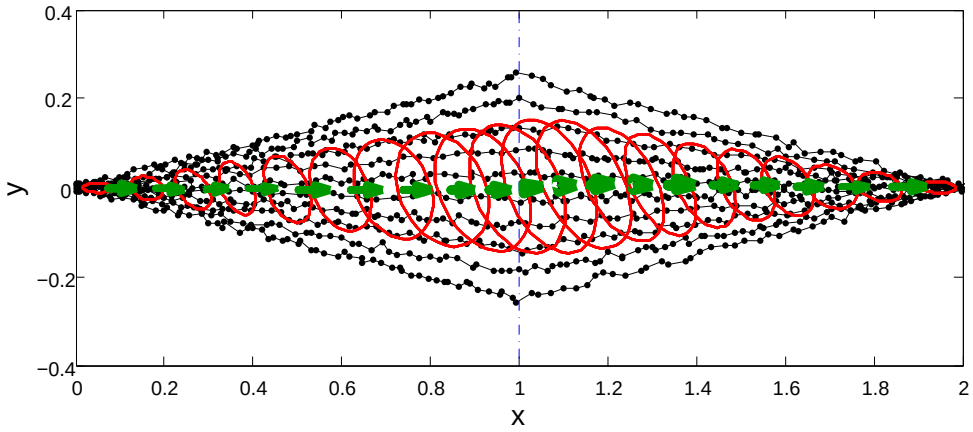


Figure 2.21: Left-Right 20-State HMM with Time Restrictions: Result.

distribution is chosen as $\boldsymbol{\pi} = (1, 0, \ldots, 0)$ and the finish state $N + 1$ is used, so that each hidden state sequence $q_1 \cdots q_T$ (with nonzero probability) begins in the first state ($q_1 = 1$) and has to pass the last

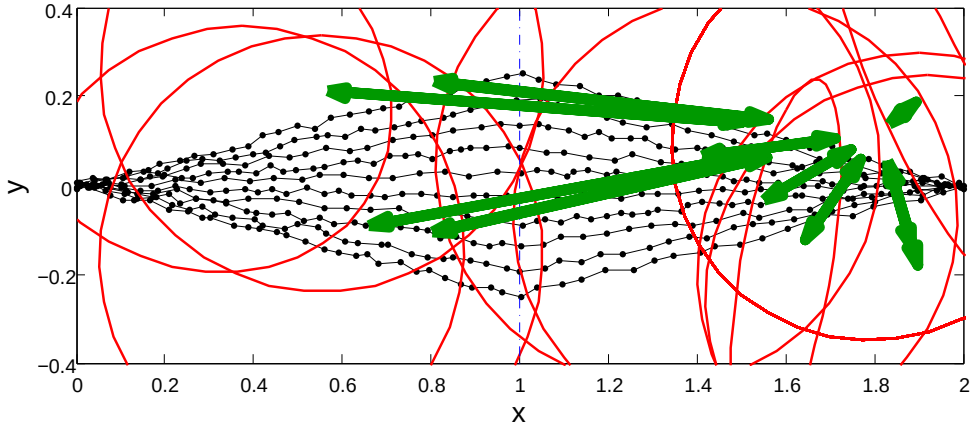


Figure 2.22: **Left-Right 10-State HMM: Random Initialization.**

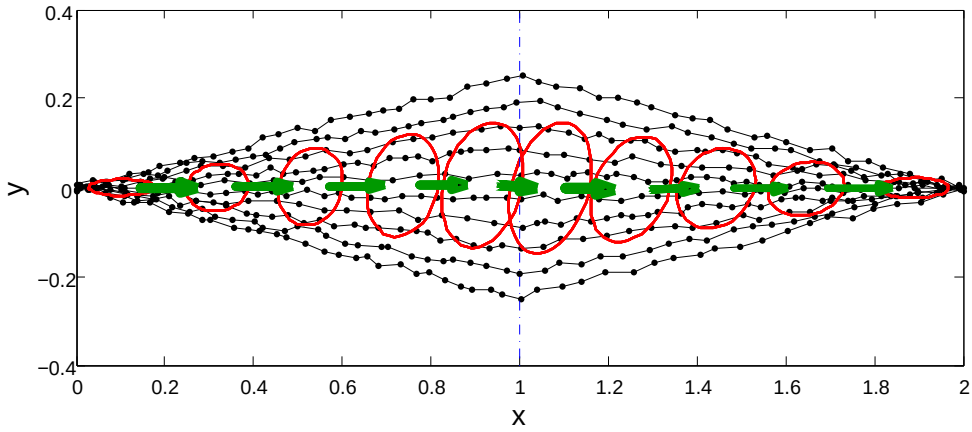


Figure 2.23: **Left-Right 10-State HMM with Time Restrictions: 5 Iterations.**

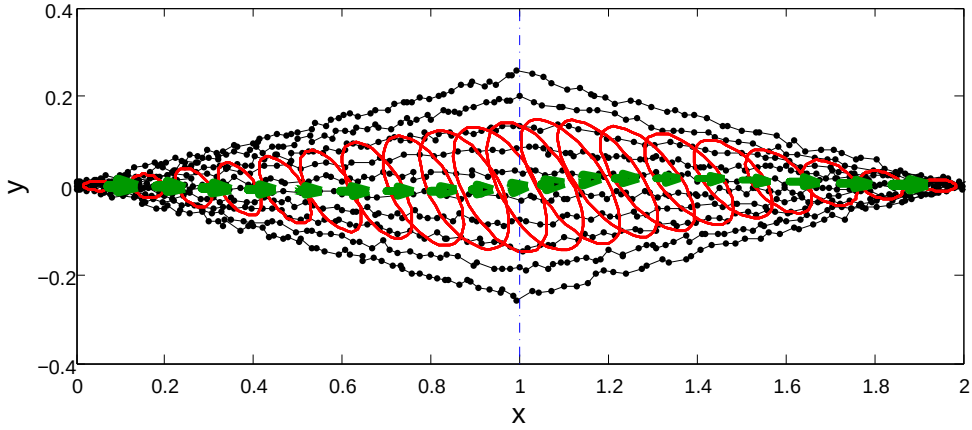


Figure 2.24: **Left-Right 20-State HMM with Time Restrictions: 5 Iterations.**

state finally ($q_T = N$, and $q_{T+1} = N + 1$). The transitions are initialized for a no-skip HMM with $a_{ii} = 0.5$ and $a_{i(i+1)} = 0.5$. The initialization of an HMM with time restrictions is performed in a

similar way (details are given in Section 2.2.3.2).

Figure 2.12 shows a trained left-right HMM with skip-transitions (as shown in Figure 2.3) and self-transitions. The training result of this 20 state HMM is very similar to the result of the ergodic 10 state HMM (Figure 2.8). Due to the choice of the skip-transitions, this 20 state HMM can only generate 2 branches (the result of the ergodic 20 state (Figure 2.11) has three branches). However, here the intention is that the state should capture the spacial variance within the movements. The synthesis of a single prototype is not possible by taking the sequence of means, as it is done in Method D. In the following, skip-transitions are not considered further.

Figure 2.13 shows a trained 5 state left-right HMM with self-transitions (where the self-transitions are not shown). The means of the covariances are distracted from the x-axis, and the sequence of state means is not very useful to synthesize a prototype of the movement class. The prototype should stay on the x-axis as for the parametric movement $\hat{\boldsymbol{x}}^p(t)$ for $p = 0$. This behavior becomes even worse for a greater number of states, as shown for $N = 7$ states in Figure 2.15. This is especially interesting since the initialization of the 7 state HMM (shown in Figure 2.14) seems to be already a good guess. In order to verify that this behavior is not caused by the initialization, a number of trials were performed: 1. In order to ensure that the transitions are optimal for the temporal behavior of the sequences, the training procedure had been used to estimate first only the transitions (until convergence) and then to estimate the transitions and covariance matrices. But the results after training (where all parameters are estimated) were with such an initialization essentially the same. 2. The covariance matrices and means were initialized randomly, similarly as shown in Figure 2.22. Again the results were essentially the same.

The problem that the means are deviated from the x-axis is only mitigated, if one decreases the variance along the y-axis (by setting $[a, b] = [-0.125, 0.125]$ instead of $[-0.25, 0.25]$). For a trained 7 state HMM (Figure 2.16), the means are on the x-axis. However, for more states, the means drift again from the x-axis. This is shown for a 20 state HMM in Figure 2.18. Another effect is that the means drift during the training procedure to locations where the variance is high. Figure 2.17 shows the 20 state HMM after 10 iterations of the Baum-Welch algorithm. After the convergence of the training (shown in Figure 2.18), most of the means are located in the middle of the movements (where $x \in [0.8, 1.2]$). In principle, this effect can be seen already for the 7 state HMM in Figure 2.16, where the first and the last state model a greater part of the movements than each of the other states. The disadvantage of such a behavior for the synthesis of a prototype is that only few means are given for those time sections, where the movement shows less variance. But especially in these sections, a prototype should be very accurate, since the variance is small.

The findings for the training without time restrictions are the following: In order to enable an accurate synthesis based on the means one should use a large number of states. But the choice of the right number of states of the left-right HMM is crucial. Already a low number of states can result in an undesired behavior of the means if the variance within the training data is large. The undesired behavior becomes severe when too many states are used. In addition, the state means are not distributed uniformly along the movements, the trend is here that only few means remain in those time sections of the trajectories, where the spacial variance of the movements is low.

In the following, the training of left-right HMMs with time restrictions is considered, which basically overcomes the problems mentioned above. In Figures 2.19, 2.20, and 2.21 training results of HMMs with time restrictions are given, where the HMMs have 5, 10, or even 20 states. As one can

see, the means are in these cases very suitable for generating good prototype movement. The means are not deviated from the x-axis and the means are distributed uniformly over the movements (in time). For the 5 state HMM, the time restrictions are set to $d_{\text{min}} = 6$ and $d_{\text{max}} = 12$, such that the model is appropriate for sequences with $T = 50$ samples. The 10 state HMM is trained with $d_{\text{min}} = 3$ and $d_{\text{max}} = 6$. For the 20 state HMM, the same settings $d_{\text{min}} = 3$ and $d_{\text{max}} = 6$ are used but for sequences with a length $N = 100$.

The convergence of the training procedure is relatively fast compared to the training without time restrictions. For the 10 state HMM, the means and covariances change after 15 iterations only negligibly. After 5 iterations, the parameters of the 10 state HMM and the 20 state HMM (Figures 2.23 and 2.24) are already very similar to the final training result (Figures 2.20 and 2.21). Even for a random initialization, as shown Figure 2.22, or a very simple initialization, as setting the means to $\boldsymbol{\mu}_i = (1, 0)$ and the covariance matrices to $\boldsymbol{\Sigma}_i = \mathbf{I}$, the HMM parameters are after 5 iterations nearly identical with the previous results after 5 iterations (Figure 2.23).

2.4 Final Remarks

The left-right no-skip model with time restrictions is very appropriate to learn a movement in which the variance is considered as noise. One can generate by Method D a good prototype of the movement class for which a model is trained. Without time restrictions, the number of states has to be chosen with care if the variance in some parts of the movements is large. With time restrictions, it is possible to use few but also a large number of states without complications. The state means are then well distributed over the sequences, and the training converges after few iterations.

Without extending the HMM, it requires a great training effort (huge training set and huge number of states) to model a parametric movement class in such a way that the model recognizes only instances of a parametric movement class. The model has no mechanism to model how a parametric movement varies depending on the movement parameters, and so it does not permit the generation of instances for given parameters nor the recognition of the parameters of an instance.

The parametric extension of the HMM (discussed in the following chapter) models the dependency of the movements on the parameters. The parametric HMM as left-right model is appropriate to generate good prototype movements for arbitrarily given parameters (see Chapter 4). Since performances of a parametric movement for fixed parameters still have spacial and temporal variations, it seems to be reasonable to stick to the training procedure with time restrictions (and a finish state) also in the case where PHMMs are used to learn parametric movements (Chapter 4).

Chapter 3

On Parametric Hidden Markov Models

A parametric HMM (PHMM) [134] is an extension of the hidden Markov model (Section 2.1) with additional latent parameters. The extension enables the PHMM to model the systematic variation within a class of sequences. In the context of parametric human movements, the variation is the variation within a movement class of a specific type. Consider, for example, the pointing actions in Figure 4.1, where an actor sits in front of a table and points in a similar way to objects that are placed at arbitrary table-top locations. The performed actions vary then according to the object location. A single usual hidden Markov model (HMM) could model the differences between the performances either as noise or through a large number of states. As discussed in Chapter 2 for HMMs, there is neither a proper means to recognize the continuous pointed to location nor a way to synthesize an accurate pointing action for an arbitrary pointed to location. This is a serious problem because the pointed to location is an important piece of information which allows one to identify the pointed to object. Moreover, the synthesis for controlling a humanoid robot (Chapter 5) requires in the same way that a movement is adapted to the environment if the objects are not located at predefined locations. A parametric HMM parameterized by the pointed to location is able to model how the movements vary. The PHMM allows one to recognize the parameters of a parametric movement (where the parameters can provide, for example, the information of the location pointed to) and can be used to synthesize movements for the desired parameters (e. g., in order to control the robot such that the robot points to an object at a location specified by the parameters).

A concrete introduction to PHMMs is given in Section 3.1. Related work is provided in Section 3.2. Different approaches to PHMMs are discussed in the Sections 3.3 and 3.4. Final remarks are given in Section 3.5.

3.1 The Parametric Hidden Markov Model

The parametric HMM (PHMM) is an extension of the hidden Markov model (HMM) with a set of additional (continuous) latent parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P)$, where the underlying HMM parameters $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ are a function $\boldsymbol{\theta} \mapsto \boldsymbol{\lambda}$ of the latent parameters. This is emphasized in the following by using the notation $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ for a PHMM. For a fixed parameter vector $\boldsymbol{\theta}_0$ the PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$ can be regarded as a usual HMM, i.e. $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$ is a usual HMM.

As a first remark, it seems to be obvious that a PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ can be a "better" model of a sequence class $\mathcal{X}$ that underlies some variation than a simple HMM $\boldsymbol{\lambda}$ of the same kind (same type of output

distributions, same number of states etc.), since the PHMM can adopt its parameters to each instance $\boldsymbol{X}$ of the sequence class by the choice of the latent parameters. As a consequence, the HMM parameters of the PHMM need to describe basically only the variation of sequences which is left if one would remove the systematic variation. A usual HMM would have to model the variation of the whole class. If one considers an HMM which models a parametric movement class, this results (generally) in a situation where the HMM also generalizes to sequences which are not part of the sequence class (discussed in Chapter 2). In this way, the PHMM is more useful for recognizing sequences that do really belong to the modeled class.

The concept of the PHMM becomes even more important if the parameters $\boldsymbol{\theta} = (\theta_i)$ have a meaningful interpretation. Consider therefore a parametric prototype movement $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$ as, for example, a pointing action, where an instance $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$ is a trajectory of the location of the finger tip which approaches a location $\boldsymbol{\theta} = (x, y)$ at a table-top. If one designs now a left-right PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ such that the HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}}$ for a parameter $\boldsymbol{\theta}$ is a good model of any performance $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_{T_k}$ of the prototype $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$ by a demonstrator, then one can assume that the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}})$ becomes for an observed action $\boldsymbol{X}$ with a pointed to location $\hat{\boldsymbol{\theta}}$ maximal for a value $\boldsymbol{\theta} \approx \hat{\boldsymbol{\theta}}$. This means that the maximum likelihood estimate $\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ should be a good guess for the true parameter $\hat{\boldsymbol{\theta}}$ of an observed pointing action $\boldsymbol{X}$. The parameter specifies then also the meaning (the pointed to location) of the action. Due to the design of the PHMM, the HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$ is also appropriate to generate a prototype movement for a parameter $\boldsymbol{\theta}$ which is similar to $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$.

However, it is inconvenient or even pointless to design a PHMM for each sequence class under consideration. Different models of the functional dependency $\boldsymbol{\theta} \mapsto \boldsymbol{\lambda}$ in combination with training procedures were developed in [134], which allows one to estimate all PHMM parameters on a finite set of training examples. An assumption is that the dependency $\boldsymbol{\theta} \mapsto \boldsymbol{\lambda}$ is smooth such that the dependency can be approximatively learned by the model based on a finite training set. This is reasonable if one considers human movements as, for example, the pointing action, where the movements for closely located pointed to locations are very similar. In addition, the training procedure also needs to be supervised, where the sequences $\boldsymbol{X}^k$ of the training set are labeled by their associated parameters $\boldsymbol{\theta}_k$ (e. g., the pointed to locations). In this way, the dependency $\boldsymbol{\theta} \mapsto \boldsymbol{\lambda}$ on the latent parameters can be learned as introduced through the labels, and the (estimated) latent parameter vector $\boldsymbol{\theta}$ for a given sequence (in the case of recognition) can be interpreted in the same way as the labels. Finally, this makes it also possible to synthesize a sequence/movement with the desired meaning/effect by using an appropriate parameter $\boldsymbol{\theta}$.

The intended advantages of the PHMM over the usual HMM in the case of parameterizable sequence classes are summarized in the following. These statements are based on the evaluation for the recognition of human gestures [134] and the experiments in the following chapters.

- improved recognition performance
  - in recognizing instances of a sequence class (since the model can adapt to each instance)
  - identifying the class membership in the case of several classes
- recognition of the parameter(s) of a particular instance of a parametric sequence
- synthesis of a parametric sequence for given parameter(s)
- learning the variation of a parametric sequence class based on a sparse training set

The models and training/recognition procedures of the work (Wilson and Bobick 1999 [134]) are

discussed in Section 3.3. An alternative approach to PHMMs (Herzog et al. [35]) is discussed in Section 3.4, this approach makes use of the usual HMM framework and interpolates HMMs.

3.2 Related Work

The parametric hidden Markov model was introduced by Wilson and Bobick (1999) in [134]. The approaches developed in [134] use either a linear model or neural networks in the non-linear case to model the dependency of the means of continuous multi-variate Gaussian output distributions. These models have been evaluated [134] for the recognition of parametric human gestures (pointing, or communicating a size by using two hands etc.). The models use the expectation-maximization (EM) procedures for training and recognition. The experiments emphasize the usability of the PHMMs to recognize the gestures and the gestures' parameters (which contains the information as, e. g., pointing directions) on the basis of recorded trajectories of the hand(s).

The concept of adaptive HMMs (basically a PHMM without a supervised training procedure) is used in speech recognition for speaker adaptation (see Gales 1997 [30]). In Leggetter and Woodland 1995 [71], a maximum likelihood approach for the training of a linear adaptive HMM is derived, which allows one to estimate the linear transformation of the mean and covariance matrix of the Gaussian output distribution of each state. An important aspect of the work [134] is that the authors' aim also at the recovery of the latent parameters, where the meaning of these parameters is important. The supervised training approaches in [134] introduce the meaning of the latent parameters. Contrary, the approaches in [30] are unsupervised. As a consequence, the meaning of the latent parameters is not given. This is inappropriate for the recognition and synthesis of parametric movements as discussed in the sections above.

In [132] a novel training algorithm for PHMMs is presented with the intention to improve the network training even on sparse training sets. An extension of [134] is that the state output distributions can be Gaussian mixtures. In the experiments, the learned models are used to synthesize movements. The considered movements are ballet, walking, and dance movements with different styles, where the PHMM parameterization is used to model the style. The movements are represented in joint-angles which are (PCA) filtered before the training. The synthesis is performed by sampling from a PHMM. Therefore, a state sequence is step-wise generated based on the transition matrix, where state repetitions are removed. The outputs are then generated for a given parameter (specifying the style) by sampling from the Gaussian mixtures components (which are randomly chosen). The outputs are then used as control points for spline interpolation to generate a movement sequence. The number of samples close to a control point correlates with the number of the state repetitions in the original state sequence. (This is Method C discussed in Section 2.2.4 of Chapter 2.)

The sampling from the output distributions is a rather poor strategy if one is interested in the synthesis of prototype movements, as discussed in Chapter 2. In [48] (Herzog et al.) the PHMM based synthesis is based simply on the Gaussian means of left-right PHMMs, this technique is used to synthesize accurate movement prototypes for pointing and reaching actions. In [48] an interpolative PHMM (as discussed in Section 3.4) is used for the recognition and synthesis experiments.

3.3 The Model-Based PHMMs

Two parametric HMMs, a linear and a non-linear, were introduced in [134] (Wilson and Bobick 1999). The models are continuous parametric HMMs $\boldsymbol{\lambda}^{\boldsymbol{\theta}} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$, where each state i has a multivariate Gaussian output distribution $b_i(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. In both approaches, only the adaptation of distributions' means $\boldsymbol{\mu}_i = \boldsymbol{f}_i(\boldsymbol{\theta})$ depending on the parameter $\boldsymbol{\theta}$ is modeled. All the other HMM parameters ($\boldsymbol{A}$, $\boldsymbol{\Sigma}_i$, and $\boldsymbol{\pi}$) are not adapted by $\boldsymbol{\theta}$ and are constant after the training.

In the linear PHMM, the functions $\boldsymbol{\mu}_i = \boldsymbol{f}_i(\boldsymbol{\theta})$ are modeled in a linear way. The non-linear PHMM models the functions through neural networks. It is worthwhile to note, that each state i of the PHMM has its own model $\boldsymbol{\mu}_i^{\boldsymbol{\theta}} = \boldsymbol{\mu}_i(\boldsymbol{\theta})$. For example, the non-linear PHMM has as many networks as states, and each network is regarded independently during the training of the model. As a consequence, each Gaussian can adapt in a different way depending on the parameter $\boldsymbol{\theta}$.

The training of both models is supervised. Each training sequence $\boldsymbol{X}^k$ has to be labeled with the associated parameter $\boldsymbol{\theta}_k$. The training procedures do not require a fixed sequence length $T = T_k$, but the number of training examples can not be arbitrarily small, e. g., two sequences are not sufficient to deduce how the means vary depending on $\boldsymbol{\theta}$ when $\boldsymbol{\theta}$ is a multidimensional vector.

The procedures of the model training and the recognition of the parameters are discussed below. In the case of the linear model, both the procedures are EM methods. The non-linear model requires a generalized EM (GEM) method. In the following, the dimension of the outputs $\boldsymbol{x}$ and the parameterization $\boldsymbol{\theta}$ are denoted by D and P, respectively. The number of states is denoted by N.

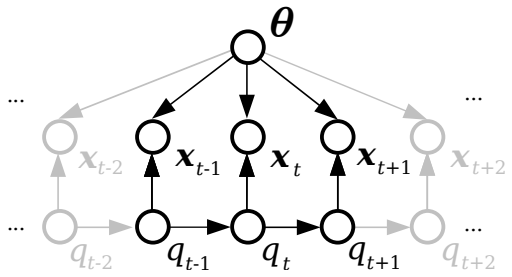


Figure 3.1: Graphical Model of Model-Based PHMMs. The figure shows how a generated output sequence $\boldsymbol{x}_1\boldsymbol{x}_2 \ldots \boldsymbol{x}_t \ldots \boldsymbol{x}_T$ depends on the parameter $\boldsymbol{\theta}$. The variable q_t denotes the hidden state at time step t.

3.3.1 The Linear Model

In Wilson and Bobick's linear PHMM (LPHMM) $\boldsymbol{\lambda}^{\boldsymbol{\theta}} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ with Gaussian output distributions $b_i(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, the dependencies $\boldsymbol{\mu}_i = \boldsymbol{f}_i(\boldsymbol{\theta})$ of the means on the parameter $\boldsymbol{\theta}$ are modeled through a linear model $\boldsymbol{\mu}_i^{\boldsymbol{\theta}} = \boldsymbol{\mu}_i(\boldsymbol{\theta})$, as given by

$$\boldsymbol{\mu}_i^{\boldsymbol{\theta}} = \bar{\boldsymbol{\mu}}_i + \boldsymbol{W}_i\,\boldsymbol{\theta}. \tag{3.1}$$

The matrix $\boldsymbol{W}_i \in \mathbb{R}^{D\times P}$ defines herein how the mean $\boldsymbol{\mu}_i$ of state i varies. The offset $\bar{\boldsymbol{\mu}}_i \in \mathbb{R}^D$ defines the location of the mean for a parameter $\boldsymbol{\theta} = \boldsymbol{0}$. For both, the training of the model and the recognition of parameters, expectation-maximization methods are given in [134], which are outlined in the following.

3.3.1.1 Training

As in the Baum-Welch algorithm for HMMs (Section 2.1.5), the estimation of the model parameters is performed iteratively starting from an initial model. In each EM iteration, the model parameters are modified to increase the likelihood of the model for a given training set. As in the Baum-Welch EM algorithm, an analytical solution for the maximization in the M-step is derived in [134]. Both, the expectation and the maximization step (E-step, M-step) are alterations of the Baum-Welch algorithm.

Since the PHMM training procedure is supervised, the sequences $\boldsymbol{X}^k$ of the given training set $\mathcal{X} = \{\boldsymbol{X}^k = \boldsymbol{x}_1^k \dots \boldsymbol{x}_{T_k}^k\}_{k=1}^K$ need to be labeled with the associated parameters $\boldsymbol{\theta}_k$. The PHMM parameters that are adjusted in each training iteration are the basic HMM parameters ($\boldsymbol{A}$, $\boldsymbol{\pi}$, the covariance matrices $\boldsymbol{\Sigma}_i$, and the offsets $\bar{\boldsymbol{\mu}}_i$) and, in addition, the matrices $\boldsymbol{W}_i$ (see Equation (3.1)). In the following, the E-step and M-step of the modified Baum-Welch algorithm are discussed.

The E-step. The E-step in the original Baum-Welch algorithm concerns the calculation of the responsibilities $\gamma_i^k(t) = p(q_t = i|\boldsymbol{X}^k, \boldsymbol{\lambda})$ and posterior transition probabilities $\xi_{ij}^k(t) = p(q_t = i, q_{t+1} = j|\boldsymbol{X}^k, \boldsymbol{\lambda})$. Here, in the case of the PHMM, the values $\gamma_i^k(t)$ and $\xi_{ij}^k(t)$ for a sequence $\boldsymbol{X}^k$ are calculated based on the adopted HMM parameters $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_k}$. Since only the means of the Gaussians are adopted by the linear PHMM, this means for a sequence $\boldsymbol{X}^k$: First, the means $\bar{\boldsymbol{\mu}}_i$ of the PHMM are "shifted" to the locations $\boldsymbol{\mu}_i^k = \boldsymbol{\mu}_i^{\boldsymbol{\theta}_k}$ that are designated by the PHMM for a sequence with a parameter $\boldsymbol{\theta}_k$. The means $\boldsymbol{\mu}_i^k = \boldsymbol{\mu}_i^{\boldsymbol{\theta}_k}$ are calculated based on Equation (3.1) for the current model parameters $\boldsymbol{W}_i$ and $\bar{\boldsymbol{\mu}}_i$ by using the parameter $\boldsymbol{\theta} = \boldsymbol{\theta}_k$. The value $\boldsymbol{\theta}_k$ is given by the label of the sequence $\boldsymbol{X}^k$. Then, the responsibilities $\gamma_i^k(t)$ and transition probabilities $\xi_{ij}^k(t)$ are calculated for the sequence $\boldsymbol{X}^k$. It might be worthwhile to note, that the calculation of the responsibilities and transition probabilities relies on the evaluation of the forward and backward variables $\alpha_i^k(t)$ and $\beta_i^k(t)$. The evaluation of these variables for the sequence $\boldsymbol{X}^k$ requires as well the usual HMM parameters $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_k}$, where the means $\bar{\boldsymbol{\mu}}_i$ are shifted.

The M-step. The M-step consists of the application of the update formulas on the model parameters. The update formulas for the parameters $\boldsymbol{\Sigma}_i$, $\bar{\boldsymbol{\mu}}_i$, and $\boldsymbol{W}_i$ are derived in [134] by maximizing the $\mathcal{Q}$-function (Section 2.1.5), which is done by setting the derivative w.r.t. the parameters to zero. To allow the maximization with respect to $\bar{\boldsymbol{\mu}}_i$ and $\boldsymbol{W}_i$, the Equation (3.1) of the adaptive $\boldsymbol{\mu}_i^{\boldsymbol{\theta}}$ is rearranged in [134] as follows:

$$\boldsymbol{\mu}_i^{\boldsymbol{\theta}} = \bar{\boldsymbol{\mu}}_i + \boldsymbol{W}_i\,\boldsymbol{\theta}_k = \left[\boldsymbol{W}_i \,\middle|\, \bar{\boldsymbol{\mu}}_i\right] \begin{bmatrix}\boldsymbol{\theta}_k \\ 1\end{bmatrix} \equiv \boldsymbol{Z}_i\,\boldsymbol{\Theta}_k. \tag{3.2}$$

Based on this, the following update formulas of $\boldsymbol{W}_i$ and $\bar{\boldsymbol{\mu}}_i$ are derived in [134]:

$$[\boldsymbol{W}_i \mid \bar{\boldsymbol{\mu}}_i] = \boldsymbol{Z}_i = \left[\sum_{k=1}^{K}\sum_{t=1}^{T_k} \gamma_i^k(t)\boldsymbol{x}_t^k\boldsymbol{\Theta}_k^T\right]\left[\sum_{k=1}^{K}\sum_{t=1}^{T_k} \gamma_i^k(t)\boldsymbol{x}_t^k\boldsymbol{\Theta}_k\boldsymbol{\Theta}_k^T\right]^{-1}, \tag{3.3}$$

where $\boldsymbol{\Theta}_k = [(\boldsymbol{\theta}_k)^T \mid 1]^T$.The update formulas of the covariance matrices $\boldsymbol{\Sigma}_i$ are [134]:

$$\boldsymbol{\Sigma}_i = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T_k} \gamma_i^k(t)(\boldsymbol{x}_t^k - \boldsymbol{\mu}_t^k)(\boldsymbol{x}_t^k - \boldsymbol{\mu}_t^k)^\top}{\sum_{k=1}^{K}\sum_{t=1}^{T_k} \gamma_i^k(t)},$$

which are similar to the formulas of a usual HMM. But these formulas make use of the shifted means $\boldsymbol{\mu}_i^k$ which are calculated in the E-step. The update formulas for the transition matrix $\boldsymbol{A}$ are the same as for usual HMMs. An update of $\boldsymbol{\pi}$ is not required for a PHMMs with $\boldsymbol{\pi} = (1, 0, \ldots, 0)$, which is always the case in this thesis.

3.3.1.2 Recognition

A PHMM trained for a certain class can be used to recognize whether a given sequence belongs to the modeled class. As discussed for HMMs, a common method is to use the likelihood of the model given a sequence.

In the case of a trained PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$, the calculation of the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ for a given sequence $\boldsymbol{X} = \boldsymbol{x}_1 \ldots \boldsymbol{x}_T$ involves the estimation of the latent parameter $\boldsymbol{\theta}$ of the sequence $\boldsymbol{X}$ in advance, since the parameter $\boldsymbol{\theta}$ of the movement is usually unknown. (If one considers, for example, a system that should recognize pointing actions performed by a person sitting in front of a table, where the person wants to mark some arbitrary table-top location, then the pointed to location is not known.)

The method [134] of estimating the parameter $\boldsymbol{\theta}$ of the sequence as a maximum likelihood estimate $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ is EM-based and very similar to the EM method for the training. Starting from an initial guess for $\boldsymbol{\theta}$, the parameter $\boldsymbol{\theta}$ is adjusted in each EM iteration to a value which increases the likelihood: In the E-step, the responsibilities $\gamma_t(i)$ for the sequence $\boldsymbol{X} = \boldsymbol{x}_1 \ldots \boldsymbol{x}_T$ and the current value $\boldsymbol{\theta}$ are calculated in the same way as discussed in the E-step of the training procedure. In the maximization step, the $\mathcal{Q}$-function is then maximized with respect to $\boldsymbol{\theta}$ which results in the following update formula [134]:

$$\boldsymbol{\theta} = \left[\sum_{t=1}^{T}\sum_{i=1}^{N} \gamma_i(t) \boldsymbol{W}_i^\top \boldsymbol{\Sigma}_i^{-1} \boldsymbol{W}_i\right]^{-1} \left[\sum_{t=1}^{T}\sum_{i=1}^{N} \gamma_i(t) \boldsymbol{W}_i^\top \boldsymbol{\Sigma}_i^{-1} (\boldsymbol{x}_t - \bar{\boldsymbol{\mu}}_i)\right]. \tag{3.4}$$

In [134], about 10 iterations turned out to be sufficient.

After the parameter $\boldsymbol{\theta}$ is estimated for the sequence $\boldsymbol{X}$, the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ can be used as an indicator of whether the sequence belongs to the class modeled by the PHMM. But contrary to an approach based on a usual HMM, also the estimate $\boldsymbol{\theta}$ of the sequence parameters is available, which can provide very useful information (e. g., the pointed to location in the case of a pointing action and a PHMM that is trained on pointing actions).

3.3.2 The Non-Linear Model

The non-linear PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ [134] differs from the linear model only in the way the dependencies $\boldsymbol{\mu}_i = \boldsymbol{f}_i(\boldsymbol{\theta})$ of the Gaussian means are modeled. In the non-linear case, each state i has its own logistic neural network with a single hidden layer:

$$\boldsymbol{\mu}_i = \boldsymbol{f}_i(\boldsymbol{\theta}) = \boldsymbol{W}_i^2 \boldsymbol{g}(\boldsymbol{W}_i^1 \boldsymbol{\theta} + \boldsymbol{b}_i^1) + \boldsymbol{b}_i^2, \tag{3.5}$$

where the matrix $\boldsymbol{W}_i^j$ and vector $\boldsymbol{b}_i^j$ store the network weights and biases used to calculate the activations of the hidden units ($j = 1$) and the outputs of the output units ($j = 2$). The function $\boldsymbol{g}(\boldsymbol{z}) = (\sigma(z_1), \ldots, \sigma(z_H))$ calculates the logistic function $\sigma(z)$ [134] for the units $1, \ldots, H$. Note, the dimensions are: $\boldsymbol{W}_i^1 \in \mathbb{R}^{H \times P}$, $\boldsymbol{W}_i^2 \in \mathbb{R}^{D \times H}$, $\boldsymbol{b}_i^1 \in \mathbb{R}^H$, and $\boldsymbol{b}_i^2 \in \mathbb{R}^D$, where D and P are the dimensions of the means $\boldsymbol{\mu}_i$ and the parameterization $\boldsymbol{\theta}$.

3.3.2.1 Training

As in the case of the linear model, the training is supervised, and follows the scheme of the EM-algorithm. The training starts from an initial model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$, which includes here also the network parameters $\boldsymbol{W}_i^j$ and $\boldsymbol{b}_i^j$ for each state i. As a consequence, the maximization step of the EM-algorithm also includes the maximization of the $\mathcal{Q}$-function with respect to the network parameters. Therefore, a back-propagation-based method is proposed in [134]. Since the back-propagation algorithm for network training is an iterative technique, the EM-algorithm is a generalized EM-algorithm (GEM) [134]. In [134] it is pointed out that only few back-propagation steps are required in one GEM iteration.

To train a non-linear PHMM, the number H of hidden units needs to be chosen. The network needs to be complex enough to model the systematic variation of the sequence class to be learned. On the other hand, the choice of too many hidden units can result in over fitting to the data, which reduces the capability of the model to generalize. To determine automatically the number of hidden units, cross validation is mentioned in [134] as being probably the only practical method.

3.3.2.2 Recognition

The approach to using the non-linear PHMM for recognition is basically the same as for the linear PHMM. Given a sequence $\boldsymbol{X}$ and the model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$, first the parameter $\boldsymbol{\theta}$ of the sequence has to be estimated, before the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ can be used. The estimation of the parameter $\boldsymbol{\theta}$ is similar to the case of the linear PHMM. But as for the training, a GEM technique is used in [134] to find the maximum likelihood estimate $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$. In order to maximize the $\mathcal{Q}$-function with respect to $\boldsymbol{\theta}$, a gradient ascent method is used in [134].

3.4 The Interpolative PHMM

The interpolative PHMM has been introduced in the work [35] (Herzog et al.). The basic idea of the interpolative approach to PHMMs is to setup some local HMMs $\boldsymbol{\lambda}^{\boldsymbol{\theta}_i}$, where each HMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}_i}$ represents sequences with a parameter $\boldsymbol{\theta}_i$. The HMM parameters $(\boldsymbol{A}^{\boldsymbol{\theta}}, \boldsymbol{B}^{\boldsymbol{\theta}}, \boldsymbol{\pi}^{\boldsymbol{\theta}})$ for an arbitrary parameter $\boldsymbol{\theta}$ are computed by component-wise linear interpolation of the local HMMs. Therfore, the model including the local HMMs is called interpolative PHMM (IPHMM). As for the PHMMs discussed above, only the continuous case of the interpolative PHMM is considered, where each state has a single multivariate Gaussian output distribution $b_i^{\boldsymbol{\theta}}(\boldsymbol{x}) \equiv \mathcal{N}_i^{\boldsymbol{\theta}}(\boldsymbol{x}) \equiv \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i^{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_i^{\boldsymbol{\theta}})$. The interpolative PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ adopts all HMM parameters $\boldsymbol{A}^{\boldsymbol{\theta}} = (a_{ij}^{\boldsymbol{\theta}})$, $\boldsymbol{\mu}_i^{\boldsymbol{\theta}}$, $\boldsymbol{\Sigma}_i^{\boldsymbol{\theta}}$ and $\boldsymbol{\pi}^{\boldsymbol{\theta}} = (\pi_i^{\boldsymbol{\theta}})$.

In the following, the case of a single scalar parameterization $\boldsymbol{\theta} = u$ and two local models $\boldsymbol{\lambda}^{u=0}$ and $\boldsymbol{\lambda}^{u=1}$ is considered. As an example, one can consider simple 2D sequences starting from the same point but leading to different points which are on a one straight line that is parameterized by a parameter u that is also used to parameterize the sequences. The HMMs $\boldsymbol{\lambda}^{u=0}$ and $\boldsymbol{\lambda}^{u=1}$ would model then sequences leading to the position $u = 0$ and $u = 1$ on the line, as illustrated in Figure 3.2, where each Gaussian distribution $\mathcal{N}_i^u(\boldsymbol{x}) = b_i^u(\boldsymbol{x})$ is displayed as an ellipse $\mathcal{N}_i^u$. The interpolation formulas

to compute the HMM parameters $\boldsymbol{\lambda}^u$ for an arbitrary parameter $u \in [0,1]$ are

$$\begin{aligned} \boldsymbol{\mu}_i^u &= (1-u)\boldsymbol{\mu}_i^0 + u\boldsymbol{\mu}_i^1, \\ \boldsymbol{\Sigma}_i^u &= (1-u)\boldsymbol{\Sigma}_i^0 + u\boldsymbol{\Sigma}_i^1, \\ a_{ij}^u &= (1-u)a_{ij}^0 + ua_{ij}^1, \\ \pi_i^u &= (1-u)\pi_{ij}^0 + u\pi_i^1. \end{aligned} \tag{3.6}$$

It is worthwhile to note that the interpolation formulas preserve the normalization of the probabilities ($\sum_j a_{ij}^u = 1$ and $\sum_i \pi_i^u = 1$). The matrix $\boldsymbol{\Sigma}_i^u$ has still the properties of a covariance matrix, since $u \in [0,1]$. For the case of local models $\boldsymbol{\lambda}^{u=u_1}$ and $\boldsymbol{\lambda}^{u=u_2}$, where the parameters u_1 and u_2 are different from 0 and 1, the parameter u in the interpolation formulas has to be substituted by $u' = (u-u_1)/(u_2-u_1)$.

The HMM $\boldsymbol{\lambda}^u$ which results by the interpolation of the two HMMs (Figure 3.2) is depicted in Figure 3.3 for an interpolation parameter $u = 0.5$. The interpolated HMM in Figure 3.3 is obviously a good model of 2D sequences with a parameter $u = 0.5$. It is not always the case that the state-wise interpolation of HMMs results in a meaningful HMM which accurately represents the sequences for the used interpolation parameter. To achieve a meaningful interpolated HMM, one has to ensure that corresponding states of the local HMMs model the same semantic parts of the sequences as in Figure 3.3. The following consideration justifies this requirement: assume two HMMs $\boldsymbol{\lambda}^{u=0}$ and $\boldsymbol{\lambda}^{u=1}$ which model sequences that consists of two parts. The first parts of the sequences with $u \in \{0,1\}$ are the sequences considered above, where the first part leads to the position u on the straight line in Figure 3.2. The second part is the reverse of the first part. Such a sequence could be a forward-backward movement of a hand. It is then possible that, e. g., the third state of the HMMs $\boldsymbol{\lambda}^{u=0}$ models some part of the forward motion of the hand whereas the third state of the HMM $\boldsymbol{\lambda}^{u=1}$ models already some part of the backward motion. Clearly, the interpolation of the considered states does not make sense. And it could happen that the interpolated HMM $\boldsymbol{\lambda}^{u=0.5}$ would represent movements which never even reach the straight line. As a conclusion, a state-wise alignment, where corresponding states of the local HMMs model corresponding parts of the movements, is vital.

In the following Section 3.4.1, the training and setup of local HMMs for a meaningful interpolation is described. The recognition using the interpolative PHMM is described in Section 3.4.2. The extension of the interpolative PHMM to a multi-variate parameterization $\boldsymbol{\theta} \in \mathbb{R}^P$ is discussed in Section 3.4.3. The grid-based extension (discussed in Section 3.4.4) makes it possible to use a larger set of local HMMs in order to increase the accuracy of the IPHMM .

3.4.1 Training: Synchronization of HMM States

Here it is discussed how it is assured that the corresponding states of local HMMs model the same semantic parts of movements or sequences. This is necessary to achieve a meaningful state-wise interpolation, as mentioned above. The required alignment of the states is similar to the alignment of two sequences by dynamic time warping (DTW) (Appendix B). However, DTW cannot be applied here, since the states of the local HMMs need to be aligned in the presence of many training sequences.

The underlying idea of the synchronization technique is to set up local HMMs $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ by using the invariance of HMMs to temporal variations. To be more concrete, it is a two step procedure: 1. A global HMM is trained on the basis of the whole training data that is available for any parameter $\boldsymbol{\theta}$.

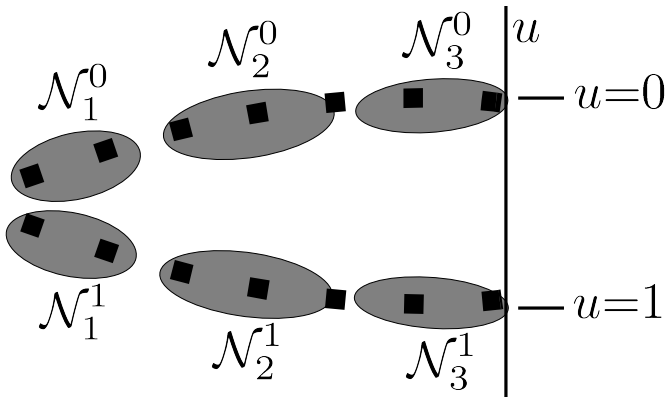


Figure 3.2: Two Local HMMs. The upper ellipses are depicting Gaussians $\mathcal{N}_1^0, \ldots, \mathcal{N}_3^0$ for the states $i = 1, 2, 3$ of a local HMM $\boldsymbol{\lambda}^{u=0}$, whereas the lower three ellipses belong to a local model $\boldsymbol{\lambda}^{u=1}$. The dots within the dark ellipses are sketching training sequences leading to the target positions $u = 0$ and $u = 1$ on the vertical line. The target positions correspond to the parameters of the sequences.

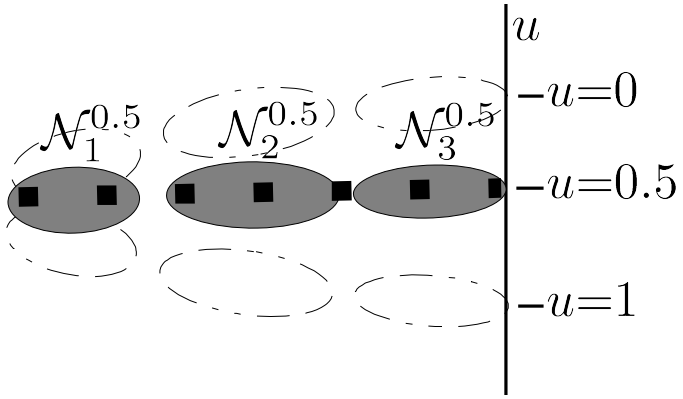


Figure 3.3: Interpolated HMM. The dark ellipses illustrate the Gaussians $\mathcal{N}_1^{0.5}, \ldots, \mathcal{N}_3^{0.5}$ for the states $i = 1, 2, 3$ of an HMM $\boldsymbol{\lambda}^{u=0.5}$ which is the result of interpolating the local HMMs shown in Figure 3.2 with an interpolation parameter $u = 0.5$. The interpolated HMM is a good model of the training sequences leading to the target position $u = 0.5$ on the vertical line.

2. Some local HMMs are trained for certain parameters by using the global HMM as initialization so that the states of the local HMMs are responsible for the same semantical parts of the sequences as the corresponding states of the global HMM. In this way, corresponding states of the local HMMs model the same semantical parts of the movements. This results in a state-wise alignment that is suitable for interpolation as illustrated in Figure 3.4.

In the following, the two steps are discussed for the case of a scalar parameterization $\boldsymbol{\theta} = u$ of the interpolative PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}=u}$ that is represented by two local HMMs $\boldsymbol{\lambda}^{u=0}$ and $\boldsymbol{\lambda}^{u=1}$. The assumption is that a training set $\mathcal{X}$ is given which contains sequences of a single movement type. The set $\mathcal{X}$ has to be of the form $\mathcal{X} = \mathcal{X}^0 \cup \mathcal{X}^1$, where the two subsets $\mathcal{X}^0$ and $\mathcal{X}^1$ contain training sequences with the parameters $u = 0$ and $u = 1$, respectively. The case of a multi-variate parametrization $\boldsymbol{\theta}$ is discussed

in Section 3.4.3

In *step one*, a global HMM $\boldsymbol{\lambda}$ is trained on the whole training set $\mathcal{X} = \mathcal{X}^0 \cup \mathcal{X}^1$. For the training of the global HMM, the EM algorithm (see Section 2.1.5) is used. A trained global HMM is sketched in Figure 3.4 through the light gray Gaussians. The situation that movements of different parameterizations are covered in such a symmetrical way by the Gaussians of the global HMM as in Figure 3.4 can be ensured by:

- *Using left-right HMMs.* In the left-right model as shown in Figure 2.2, each hidden state sequence has to pass through the states in a given order. Thus, each state is responsible for at least some part of any sequence, i. e., a state can not generate outputs for one movement and none for the remaining movements as it is possible in an ergodic HMM.

- *Constraining the "time-warp capabilities" of the HMM.* This is done in order to prevent an asymmetric alignment of the states of the local HMMs. This is, for example, the case, when some states generate very few outputs for movements with a certain parameter and many outputs for movements with other parameters. To assure that all states have approximately the same output duration, time restrictions are added to the states of the HMMs as discussed in the Sections 2.2.3.2 and 2.3.2. A reasonable choice is to allow each state to generate, e. g., at most $d_{\text{max}} = 3$ outputs and at least $d_{\text{min}} = 2$. However, the number of states has to be chosen depending on the sequence length. This can also be solved by resampling the sequences to the another length (e. g., by linear interpolation). The idea of constraining the time-warping capabilities is also used for dynamic time warping (DTW), see Appendix B. Interesting examples for a proper and improper alignment are shown for DTW in [62] (Figure 2 B and C). The example in Figure 2C (see [62]) is very similar to an improper alignment of the local HMMs when time restrictions are not used.

Based on these settings, one yields a symmetric global HMM as sketched in Figure 3.4. In Figure 3.5 (left) this is shown for recoded 3D pointing trajectories and three local HMMs.

In *step two*, the partial training sets $\mathcal{X}^{\boldsymbol{\theta}=u}$ for sequences with a parameter $\boldsymbol{\theta} = u$ are used to set up the local HMMs $\boldsymbol{\lambda}^{\boldsymbol{\theta}=u}$. Each local HMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}=u}$ is set up in the same way. In the following, the setup of the local HMM $\boldsymbol{\lambda}^{u=0}$ based on the training set $\mathcal{X}^{u=0}$ is considered. Again, the EM algorithm is used. But now, the parameters of the global HMM $\boldsymbol{\lambda}$ are used as initial values for the EM. In addition, to preserve the state alignment of the local HMM with the global HMM, the means are kept fixed after the first EM step. In the context of Figure 3.4, this procedure results in a shrinking of the light gray ellipsoids $\mathcal{N}_i$ of the global HMM to the dark gray ellipsoids $\mathcal{N}_i^0$ of the local HMM $\boldsymbol{\lambda}^{u=0}$.

In the following, it is exemplified why this adapted EM procedure gives a proper state alignment for the local HMMs $\boldsymbol{\lambda}^0$ and $\boldsymbol{\lambda}^1$: In the first E-step of the EM algorithm for a local model $\boldsymbol{\lambda}^u$ ($u \in \{0, 1\}$), the posterior probabilities $\gamma_i^k(t) = P(q_t = i | \boldsymbol{X}^k, \boldsymbol{\lambda})$ of being in state i at time step t are computed for each sequence $\boldsymbol{X}^k = \boldsymbol{x}_1^k \ldots \boldsymbol{x}_T^k$ in $\mathcal{X}^u$. But this is done based on the current parameter values, which are in this iteration the values of the global HMM. Thus, $\gamma_i^k(t)$ is the responsibility of state i of the global HMM $\boldsymbol{\lambda}$ for generating $\boldsymbol{x}_t^k$. In the following M-step of the EM algorithm, each $\boldsymbol{\mu}_i$ of a Gaussian of state i is re-estimated as a $\gamma_i^k(t)$-weighted mean:

$$\boldsymbol{\mu}_i = \frac{\sum_{t,k} \gamma_i^k(t) \boldsymbol{x}_t^k}{\sum_{t,k} \gamma_i^k(t)}. \tag{3.7}$$

The meaning of this procedure is now analyzed on the basis of Figure 3.4. In the figure, two sequences $\boldsymbol{X}^0$ and $\boldsymbol{X}^1$ are shown, where the upper sequence $\boldsymbol{X}^0$ leads in the context of the figure to a position $u = 0$ on the vertical line. The lower sequence leads to a position $u = 1$. It is assumed in the following that these positions are used as a parameterization of the PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}=u}$ and that the training sets are $\mathcal{X}^0 = \{\boldsymbol{X}^0\}$, $\mathcal{X}^1 = \{\boldsymbol{X}^1\}$, and $\mathcal{X} = \mathcal{X}^0 \cup \mathcal{X}^1$. The global HMM (indicated by the dark gray ellipsoids $\mathcal{N}_i$) is a left-right model, which is a result of the training procedure (*step one*), where the training is performed on the basis of the set $\mathcal{X}$. Now the training of the local HMM $\boldsymbol{\lambda}^0$ by the use of the set $\mathcal{X}^0 = \{\boldsymbol{X}^0\}$ is considered. In the first E-step, the responsibilities $\gamma_i^0(t), t = 1, 2$ for the outputs $\boldsymbol{x}_{t=1}^0$ and $\boldsymbol{x}_{t=2}^0$ (of the sequence $\boldsymbol{x}_1^0 \cdots \boldsymbol{x}_7^0$, which is the upper sequence in Figure 3.4) are large for the state $i = 1$ but small for all the other states ($i > 1$). This is caused by the position of the Gaussian $\mathcal{N}_1$ of the global HMM $\boldsymbol{\lambda}$. As a consequence, the mean $\boldsymbol{\mu}_1^0$ of the Gaussian $\mathcal{N}_1^0$ of the state $i = 1$ of the local HMM $\boldsymbol{\lambda}^0$, as calculated by Equation (3.7), is between $\boldsymbol{x}_1^0$ and $\boldsymbol{x}_2^0$ as shown in Figure 3.4. In the same way, the first E-step performed for setting up the local HMM $\boldsymbol{\lambda}^1$ on the basis of the lower sequence $\boldsymbol{x}_1^1 \boldsymbol{x}_2^1 \ldots \boldsymbol{x}_T^1$ results in a mean $\boldsymbol{\mu}_1^1$ of the Gaussian $\mathcal{N}_1^1$ of the state $i = 1$ of the HMM $\boldsymbol{\lambda}^1$, where $\boldsymbol{\mu}_1^1$ is between $\boldsymbol{x}_1^1$ and $\boldsymbol{x}_2^1$. Hence, the means $\boldsymbol{\mu}_1^0$ and $\boldsymbol{\mu}_1^1$ of the first states of the local HMMs $\boldsymbol{\lambda}^0$ and $\boldsymbol{\lambda}^1$ have a symmetrical alignment that is inherited from the global model $\boldsymbol{\lambda}$. The reasoning for the alignment of the other states of the two local HMMs is similar. Since the procedure in *step two* keeps the means fixed after the first EM iteration, the alignment is preserved. The smaller covariances of the Gaussians of the local HMMs, as shown in Figure 3.4, is a result of the partial training sets $\mathcal{X}^0 = \{\boldsymbol{X}^0\}$ and $\mathcal{X}^1 = \{\boldsymbol{X}^0\}$. This is a desired effect, since the local HMMs should model only the variance within the sequences for certain parameters (here: $u = 0$ and $u = 1$).

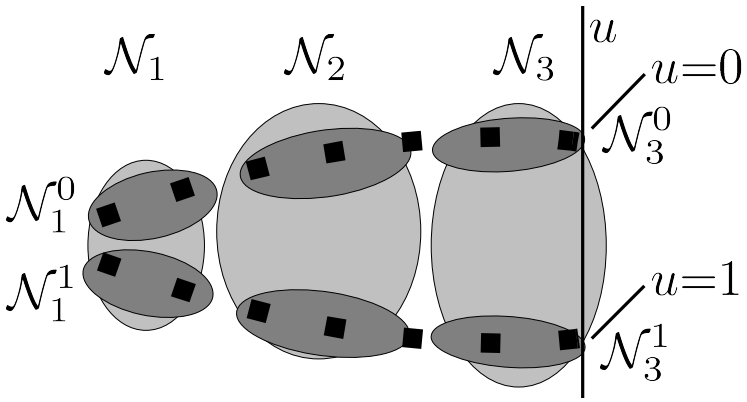


Figure 3.4: State-Alignment of Local HMMs. As in Figure 3.2, the upper three dark ellipses are depicting Gaussians $\mathcal{N}_1^0, \ldots, \mathcal{N}_3^0$ for the states $i = 1, 2, 3$ of a local HMM $\boldsymbol{\lambda}^0$. The lower three dark ellipses belong to a local model $\boldsymbol{\lambda}^1$. The dots within the dark ellipses are sketching training sequences $\boldsymbol{X}^0$ and $\boldsymbol{X}^1$ with parameters $u = 0$, and $u = 1$ depending on their target position u on the vertical line. In addition, the Gaussians $\mathcal{N}_i$ of a global model $\boldsymbol{\lambda}$ are indicated in light gray. This global HMM $\boldsymbol{\lambda}$ is a model of all training sequences with $u \in [0, 1]$. It is used to set up the local HMMs.

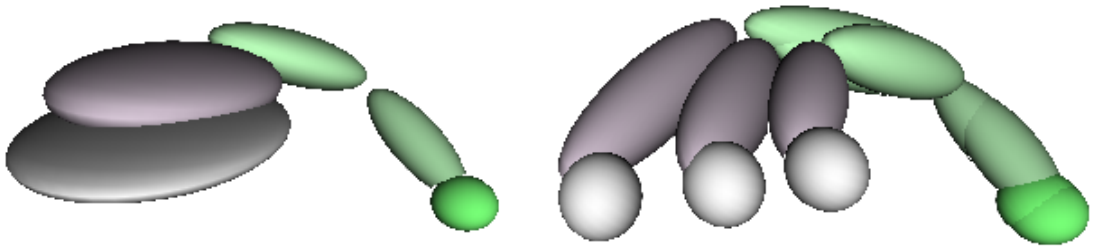


Figure 3.5: State-Alignment of Local HMMs for Recored Data. Similar to Figure 3.4, the state outputs of a global HMM (left) and three local HMMs (right) of an interpolative PHMM are shown. The local HMMs (right) are trained on recorded 3D trajectories of the finger tip of a person performing pointing action to three different pointed to locations at a table-top (the action is also shown in Figure 4.1). The index finger always starts at the same point which is modeled by the Gaussian displayed by the green ball. The specific pointed to locations are modeled by the light gray Gaussians (balls) of the three local HMMs. The global HMM (left) has been used to set up the local HMMs. It models the finger tip trajectories to all table-top locations and has thus a "disc like" Gaussian covering the plane of the table-top. Note, the global HMM models the systematic variation of the movements as noise.

3.4.2 Recognition

The approach of using the interpolative PHMM for recognition is basically the same as discussed for the model-based PHMMs in Section 3.3. Before the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ of the model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ given a sequence $\boldsymbol{X}$ can be used for recognition, the parameter $\boldsymbol{\theta}$ has to be determined for the sequence $\boldsymbol{X}$. The parameter is estimated again in a maximum likelihood approach $\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$. Therefore, an optimization technique such as gradient-ascent is used, where in each iteration step the parameter $\boldsymbol{\theta}$ is modified in the direction of the gradient $\nabla_{\boldsymbol{\theta}} \ln p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ in order to maximize the value of the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$. The gradient is calculated in this thesis numerically by evaluating the log-likelihood $\ln p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ for different values of $\boldsymbol{\theta}$.

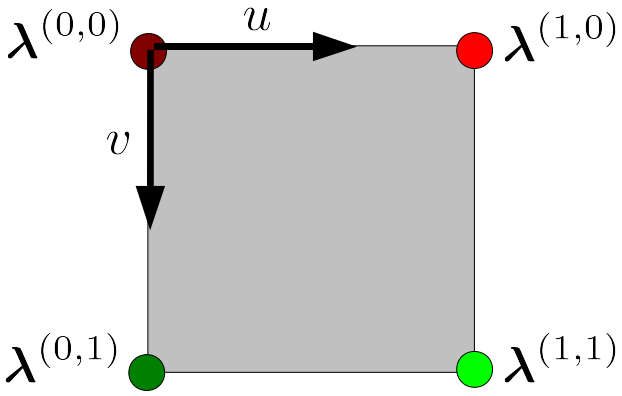


Figure 3.6: Setup of Local HMM for a 2D-Parametrization. The figure shows a setup of an interpolative PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ for a 2-dimensional parameterization $\boldsymbol{\theta} = (u, v)$. The setup requires four local HMMs $\boldsymbol{\lambda}^{(u,v)}$, $u, v \in \{0, 1\}$ to allow to interpolate the HMM parameters $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(u,v)}$ for $u, v \in [0, 1]$.

3.4.3 Multivariate Parametrizations

In sections above, only the uni-variate case of a parametrization $\boldsymbol{\theta} = u$ of an IPHMM is considered. The extension of the IPHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ to the multi-variate case of a parameterization $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P)$ can be accomplished by using 2^P local HMMs $\boldsymbol{\lambda}^{(\theta_1,\ldots,\theta_P)}$ for $\theta_1, \ldots, \theta_P \in \{0, 1\}$.

Deduction of HMM Parameters for an Arbitrary Parameterization. The deduction of the HMM parameters $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(\theta_1,\ldots,\theta_P)} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ for an arbitrary parameter $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_P) \in [0, 1]^P$ can be approached as an iterative procedure. First all local models $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(\theta'_1,\ldots,\theta'_P)}$ for $\theta'_p \in \{0, 1\}$ are used to interpolate the local models $\boldsymbol{\lambda}^{(\hat{\theta}_1,\theta'_2,\ldots,\theta'_P)}$ for $\theta'_2, \ldots, \theta'_P \in \{0, 1\}$. The resulting HMMs $\boldsymbol{\lambda}^{(\theta_1,\theta'_2,\ldots,\theta'_P)}$ are then used to deduce HMMs $\boldsymbol{\lambda}^{(\hat{\theta}_1,\hat{\theta}_2,\theta'_3\ldots,\theta'_P)}$ for $\theta'_3, \ldots, \theta'_P \in \{0, 1\}$, which have in the first and second component the required parameter values $\hat{\theta}_1$ and $\hat{\theta}_2$. Finally, when the iteration over each component θ_p is complete, the HMM $\boldsymbol{\lambda}^{(\hat{\theta}_1,\ldots,\hat{\theta}_P)}$ for the given parameter $\hat{\boldsymbol{\theta}}$ has been deduced.

In the following, the particular case of a 2-dimensional parameterization $\boldsymbol{\theta} = (u, v)$ is exemplified. To deduce the HMM parameters $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(\hat{u},\hat{v})}$ for arbitrary parameters $\hat{u}, \hat{v} \in [0, 1]$, four local HMMs $\boldsymbol{\lambda}^{(u,v)=(0,0)}$, $\boldsymbol{\lambda}^{(u,v)=(1,0)}$, $\boldsymbol{\lambda}^{(u,v)=(0,1)}$, and $\boldsymbol{\lambda}^{(u,v)=(1,1)}$ are used, which are the models for sequences with corresponding parameter vectors (u, v). The rectangular 2D domain of the parameter vector $(u, v) \in [0, 1]^2$ is illustrated in Figure 3.6, where the points $(u, v) = (0, 0), (1, 0), (0, 1), (1, 1)$ of the domain are labeled by the corresponding local HMMs. To deduce the HMM parameters $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(\hat{u},\hat{v})}$, first the HMM parameters of the models $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(\hat{u},v')}$, $v' \in \{0, 1\}$ are interpolated by component-wise linear-interpolation:

$$\boldsymbol{\lambda}^{(\hat{u},0)} = (1 - \hat{u})\boldsymbol{\lambda}^{(0,0)} + \hat{u}\boldsymbol{\lambda}^{(1,0)}, \tag{3.8}$$

$$\boldsymbol{\lambda}^{(\hat{u},1)} = (1 - \hat{u})\boldsymbol{\lambda}^{(0,1)} + \hat{u}\boldsymbol{\lambda}^{(1,1)}, \tag{3.9}$$

where each equation is just a short-hand of the formulas in Equation (3.6). The interpolations in (3.8) and (3.9) correspond in Figure 3.6 to the interpolations between the two HMMs at the red dots and between the two HMMs at the blue dots. Based on the models $\boldsymbol{\lambda}^{(\hat{u},0)}$ and $\boldsymbol{\lambda}^{(\hat{u},1)}$, the HMM parameters $\boldsymbol{\lambda}^{\boldsymbol{\theta}=(\hat{u},\hat{v})}$ can now be calculated, again, by using linear-interpolation:

$$\boldsymbol{\lambda}^{(\hat{u},\hat{v})} = (1 - \hat{v})\boldsymbol{\lambda}^{(\hat{u},0)} + \hat{v}\boldsymbol{\lambda}^{(\hat{u},1)}. \tag{3.10}$$

The iterative interpolation procedure in the case of a 3-dimensional parameterization $\boldsymbol{\theta} = (u, v, w)$ of the interpolative HMM is used in Chapter 5.

Training. The training procedure in the multi-variate case is the same as discussed in Section 3.4.1. But here, in the case of a P-dimensional parameterization of the interpolative PHMM $\boldsymbol{\lambda}^{(\theta_1,\ldots,\theta_P)}$, one requires 2^P training sets $\mathcal{X}^{(\theta_1,\ldots,\theta_P)}$ with training sequences that corresponds to the parameter vectors $(\theta_1, \ldots, \theta_P) \in \{0, 1\}^P$. The training consists of the same two step procedure: first, a global HMM is trained on the whole training data available, and then the local HMMs $\boldsymbol{\lambda}^{(\theta_1,\ldots,\theta_P)}$ ($(\theta_1, \ldots, \theta_P) \in \{0, 1\}^P$) are trained on the corresponding training sets. Therefore, the procedure described Section 3.4.1 is used.

As mentioned for the uni-variate case, it is not necessary to use a parameterization where each parameter θ_p is restricted to the interval $[0, 1]$. As an example, in the bi-parametric case $\theta = (u, v)$ the four local HMMs can correspond to parameter vectors (u, v) with $u \in \{u_1, u_2\}$ and $v \in \{v_1, v_2\}$. To interpolate the HMM $\boldsymbol{\lambda}^{(\hat{u},\hat{v})}$ the interpolation formulas (3.8), (3.9), and (3.10) can be used, where the

variables $\hat{u}$ and $\hat{v}$ are substituted by $\hat{u}' = (\hat{u} - u_1)/(u_2 - u_1)$ and $\hat{v}' = (\hat{v} - v_1)/(v_2 - v_1)$. Moreover, it is even not necessary to use local HMMs for parameter vectors (u, v) which are the corners of a rectangle (in the case $P = 2$), but then it is not so easy to interpret the interpolation parameters u, v.

Recognition. The recognition procedure in the multi-variate case is the same as discussed in Section 3.4.2. However, to evaluate the likelihood $p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ for a given parameter $\boldsymbol{\theta}$, the HMM parameters $\boldsymbol{\lambda}^{\boldsymbol{\theta}} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ for $\boldsymbol{\theta}$ have to be computed by applying the iterative interpolation procedure discussed above.

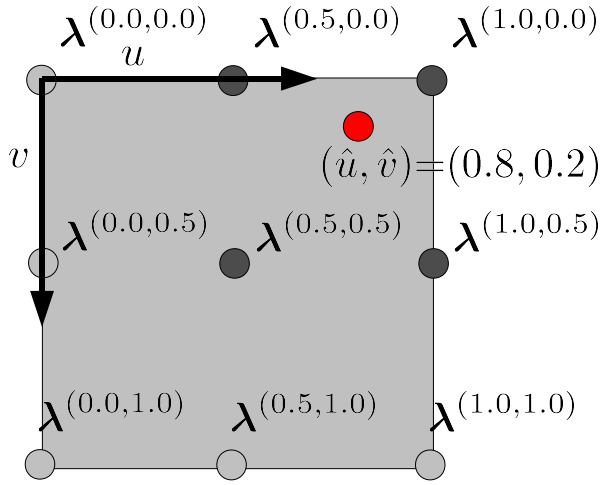


Figure 3.7: Grid of Local HMMs. The figure shows a grid of local HMMs that can be used to deduce the HMM parameters of an interpolative PHMM $\boldsymbol{\lambda}^{(u,v)}$ for arbitrary parameters (u, v). For example, the local HMMs indicated by the dark gray circles are used for the interpolation of the HMM parameters $\boldsymbol{\lambda}^{(\hat{u},\hat{v})}$ for parameters $(\hat{u}, \hat{v}) = (0.8, 0.2)$.

3.4.4 Grid-Based Extension

In certain cases, the adaptivity of the interpolative PHMM $\boldsymbol{\lambda}^{(\theta_1,\ldots,\theta_P)}$ as discussed above might be too coarse. It is possible to increase the accuracy of the PHMM by using more than the required 2^P local HMMs.

In the following the case of a 3×3 grid for a 2-dimensional parameterization $\boldsymbol{\theta} = (u, v)$ is discussed, but the use of arbitrary grid sizes (e. g., 2×3 or 5×5) is basically the same and the extension to the multi-variate case ($P > 2$) is straight forward. In the following it is assumed that 9 training sets $\mathcal{X}^{(u,v)}$ of sequences with associated parameters $(u, v) \in \{0.0, 0.5, 1.0\}^2$ are available. Based on these sets, 9 local HMMs $\boldsymbol{\lambda}^{(u,v)}$ are set up in an synchronized way by the use of a global HMM trained on the whole training data as discussed in Section 3.4.1. The resulting grid of 3×3 HMMs is illustrated in Figure 3.7. The HMM parameters $\boldsymbol{\lambda}^{(\hat{u},\hat{v})} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ for arbitrary parameters $(\hat{u}, \hat{v}) \in [0, 1]$ are then deduced on the basis of the four local HMMs $\boldsymbol{\lambda}^{(u,v)}$ that are close to the point $(\hat{u}, \hat{v})$. As an example, consider the parameter values $(\hat{u}, \hat{v}) = (0.8, 0.2)$. Then, the grid points (u, v) with $u \in \{0.5, 1.0\}$ and $v \in \{0.0, 0.5\}$ are used. To deduce the HMM parameters $\boldsymbol{\lambda}^{(\hat{u},\hat{v})} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$, the interpolation formulas (3.8), (3.9), and (3.10) are applied, but with the HMMs $(\boldsymbol{\lambda}^{(0.5,0.0)}, \boldsymbol{\lambda}^{(1.0,0.0)}, \boldsymbol{\lambda}^{(0.5,0.5)}, \boldsymbol{\lambda}^{(1.0,0.5)})$ in place of the HMMs $(\boldsymbol{\lambda}^{(0,0)}, \boldsymbol{\lambda}^{(1,0)}, \boldsymbol{\lambda}^{(0,1)}, \boldsymbol{\lambda}^{(1,1)})$. In addition, the interpolation parameters have to

be substituted by $\hat{u}' = (\hat{u} - 0.5)/0.5$ and $\hat{v}' = (\hat{v} - 0.0)/0.5$. The four local HMMs that are used are indicated in Figure 3.7 by dark gray circles. The point $(\hat{u}, \hat{v}) = (0.8, 0.2)$ in the parameter space is shown as well (red circle).

Recognition. The estimation of the parameterization $\boldsymbol{\theta}$ for a given sequence $\boldsymbol{X}$ and an interpolative PHMM can be performed as discussed before. This involves the estimation of $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}})$ through gradient-ascent.

Alternatively, one could use a multi step approach which is discussed in the following for a 3×3 grid of local PHMMs in the 2-dimensional case $P = 2$. First, the gradient-ascend method is applied on the interpolative PHMM that based only on the four HMMs at the corners of the grid, since this PHMM is a corse/smooth version of the grid-based PHMM. This first estimate can then be refined for the PHMM based on the whole grid.

3.5 Final Remarks

One concern is the implementation of the PHMM training algorithms. The linear and non-linear PHMMs require some extensions of the usual HMM training framework, whereas the interpolative PHMM makes only use of the usual HMM training procedures. The interpolative approach is also a simple way to adapt all HMM parameters, whereas in the linear and non-linear approach to PHMMs, only the adaptation of the means of the output distributions has been considered so far [134]. From a theoretical point of view, the linear model is more restrictive than the interpolative PHMM. For example, the linear dependency of the means on a 2-dimensional parameter can be modeled through bi-linear interpolation formulas.

An import aspect is that the linear (and also the non-linear) model can be trained on sequences labeled with arbitrary values of the parametrization, whereas the setup of the local HMMs for the interpolative PHMM requires that the training sequences are provided for a grid in the parameter space. This becomes especially cumbersome if the parameterization is highly dimensional, since the minimum requirement of grid points for the interpolative model is 2^P, where P is the dimension of the parametrization. The problem of the interpolative PHMM that the training sequences have to be provided for grid points might be less problematic in the following setup: Since all three approaches to PHMM require that the sequences are labeled with their associated parameters, a suitable setup to learn pointing actions (demonstrated by a person sitting in front of a table) is to use a raster on the table-top to derive the pointed to locations that are used as labels for the recorded sequences. For such a setup the interpolative HMM would be a simple approach to PHMMs.

However, the non-linear PHMM is the most general model (concerning the adaptivity of the means) and has the possibility to adapt also to sequences where the dependency on the parametrization is highly non-linear. But it makes use of neural networks which introduce the problem of choosing a proper network topology that is general enough to model the sequences but does not suffer from over fitting.

In [134] the linear and non-linear PHMM have been applied and evaluated for the recognition of basic human gestures. In addition to the application of recognition, the following chapter investigates also the applicability of PHMMs to learn human movements in order to synthesize these movements for arbitrary parameters.

Chapter 4

Representing Parametric Movements with Parametric Hidden Markov Models

In this chapter the parametric hidden Markov models (PHMMs) are evaluated for representing parametric human movements. In Section 4.1 the basics of representing movements through PHMMs are discussed. This discussion makes use of the parametric movement that is introduced in Chapter 2. In Section 4.2, the PHMMs (LPHMM, IPHMM, and QPHMM) are evaluated for the synthesis and recognition of basic parametric arm actions. These actions are pointing (Figure 4.1) and reaching. The actions are parameterized by a single table-top location. The quadratic PHMM (QPHMM) is an extension of the linear PHMM (LPHMM), which is introduced (Section 4.2.3) and evaluated (Section 4.2.4.1). Besides the spatial synthesis and recognition errors, the time durations of the state outputs (Section 4.2.4.2) are discussed. In Section 4.3, a parametric PHMM is used to learn a parametric movement which is more complicated. The movement is parameterized by two table-top locations, where the hand of the human approaches first one location and then the second location. A summary is given in Section 4.4.

4.1 Representing Parametric Human Movements

In this section, the application of the parametric hidden Markov models (PHMMs) for representing parametric human movements is discussed. The basic concept is that a single parametric HMM can be trained on demonstrations of a parametric movement of certain type. The trained PHMM allows one to synthesize the movement for arbitrary values of the parameterization and to recognize an instance of the movement and its parameters.

The concept of parametric movements which is used in this thesis is essential (see also Chapter 2 and Section 1.2.1). In Figure 4.1, a performance of a pointing action is shown, where the person points to an object at a location $\boldsymbol{p} = \boldsymbol{p}_0$ at the table-top. The arm pose (at a certain point in time) can be represented as a state vector $\boldsymbol{x}$, which specifies the arm pose in some format such as joint angles or some descriptive points of the body (body-fixed points) as the finger tip and centers of the arm joints etc. The performance of the arm movement can then be regarded as a trajectory $\boldsymbol{x}(t)$ in the state/pose space with $t \in [0, T]$. It seems to be reasonable to assume that each repetition of the pointing action by the person for a fixed pointed to object location $\boldsymbol{p}_0$ is an approximation of an underlying prototype movement $\hat{\boldsymbol{x}}(t) = \hat{\boldsymbol{x}}^{\boldsymbol{p}_0}(t)$. But this prototype has to be different when the pointed to object is placed

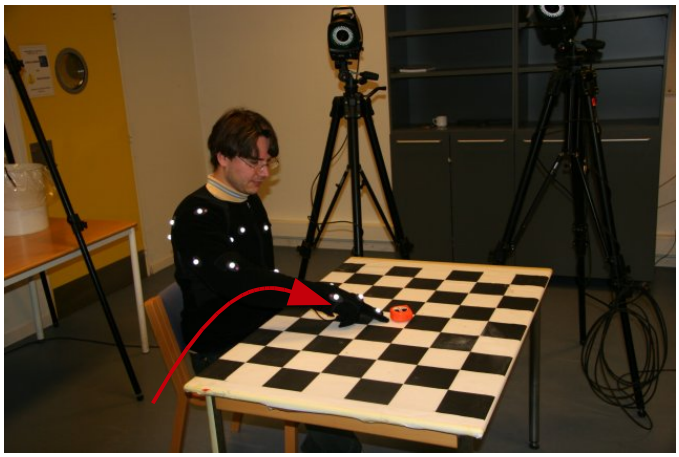


Figure 4.1: Parametric Pointing Action. The picture shows a person demonstrating pointing actions. The body motion is recorded by a marker based motion capture system. Each demonstration is an instance of the parametric pointing action, which depends on the pointed to object location. The red arc indicates the trajectory of the index finger tip leading to the object (orange). Each performance starts from a certain pose (considered as base pose), where the arm is hanging down, and returns to the base pose.

at another (arbitrary) location $\boldsymbol{p}$, since the index finger tip has to reach the location $\boldsymbol{p}$. If the object location is the only characteristic aspect of the movement, then it is reasonable to denote the movement prototype as $\hat{\boldsymbol{x}}^{\boldsymbol{p}}(t)$ depending on $\boldsymbol{p}$, where $\boldsymbol{p}$ is the parameterization of the parametric movement $\hat{\boldsymbol{x}}^{\boldsymbol{p}}(t)$.

The case that a parametric movement as the pointing actions (Figure 4.1) depends only on a single location (as an object location $\boldsymbol{p}$) is established for each performance of the pointing action by defining a base pose of the arm (where the arm is hanging down beside the person). Each performance of the pointing action begins and ends (regardless of the object location) in the base pose. Another example for a more complicated parametric arm movement in the table-top scenario of Figure 4.1 is an action where an object at the table-top is moved from a location $\boldsymbol{q}$ to a new target location $\boldsymbol{r}$ (e.g., by touching on the object). If neither the location $\boldsymbol{q}$ nor $\boldsymbol{r}$ are predefined for this action, then the arm movement during the object interaction can be defined again as a parametric arm movement $\boldsymbol{x}^{\boldsymbol{p}}(t)$ which is parameterized by the two object locations, i.e. $\boldsymbol{p} = (\boldsymbol{q}, \boldsymbol{r})$. An interesting aspect of such a bi-parametric action is that it can be used as an action primitive from which one can build more complex actions. Consider therefore another bi-parametric action $\bar{\boldsymbol{x}}^{\bar{\boldsymbol{p}}=(\bar{\boldsymbol{q}},\bar{\boldsymbol{r}})}$ as moving the hand from a location $\bar{\boldsymbol{q}}$ to a location $\bar{\boldsymbol{r}}$, then these both action primitives can be concatenated properly, i.e. if the primitive $\boldsymbol{x}^{\boldsymbol{p}}(t)$ is executed first then the action $\bar{\boldsymbol{x}}^{\bar{\boldsymbol{p}}}(t)$ can be concatenated by using $\bar{\boldsymbol{q}} = \boldsymbol{r}$ without generating a gap between the actions. Hence, both primitives can be concatenated several times to generate a complex action.

However, following this concept of parameterized movements, it is easy to guess even more complicated parameterizations or a parameterization that is as simple as that of the parametric movement introduced in Section 2.3, which is parameterized by a single scalar parameter p. However, it is not necessary that the parameters describe some points (as via-points [78]) in space which are approached during the execution of the movement. A parametric movement can be also parameterized by other parameters such as style parameters etc. In [132] the movements are parameterized by parameters

which describe the style of dancing movements etc. A style parameter could be used also to define the execution speed of a movement. However, style parameters are not considered in this thesis.

An important aspect of the parametric movements is that the parameters of the movements can provide important information about the movement. If one considers an observed performance of the parametric pointing action, then it is necessary to know that the performed movement is a pointing action but an important aspect is to know to which location the person points. This information may be used then to identify the object that is meant by the person. The parameterization defines in this sense an essential part of the meaning of the movement. In the case of the relocation action, the parameterization $\boldsymbol{p} = (\boldsymbol{q}, \boldsymbol{r})$ defines the effect on the environment, i.e., an object is moved from a location $\boldsymbol{q}$ to the location $\boldsymbol{r}$. In both cases the meaning/effect defines the semantic of a certain movement instance.

4.1.1 Representing Parametric Movements with PHMMs

In the following it is argued why it makes perfect sense to represent a parametric movement $\hat{\boldsymbol{x}}^{\boldsymbol{\theta}}(t)$ with a PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$, where the parameterization of the parametric movement is used as parameterization of the model. The intention is the following: in Chapter 3, the PHMMs are introduced as an extension of HMMs which adapt the usual HMM parameters depending on the parameter $\boldsymbol{\theta}$. For an arbitrary but fixed parameter $\boldsymbol{\theta} = \boldsymbol{\theta}_0$ of the parametric movement, a PHMM is a usual HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$. In Chapter 2 it is shown that a left-right no-skip HMM with Gaussian output distributions is appropriate to represent a non-parametric movement (as $\hat{\boldsymbol{x}}^{\boldsymbol{\theta}_0}(t)$), where the variance (as in the case of human repetitions of the specific movement $\hat{\boldsymbol{x}}^{\boldsymbol{\theta}_0}(t)$) is modeled by the Gaussians. The sequence of means provide then a good prototype close to the assumed prototype $\hat{\boldsymbol{x}}^{\boldsymbol{\theta}_0}(t)$ of the parametric movement $\hat{\boldsymbol{x}}^{\boldsymbol{\theta}}(t)$.

However, the PHMMs have to be trained on movement data. In order to see that the intention is reasonable, the synthetic parametric prototype movement $\hat{\boldsymbol{x}}^{\theta}(t)$ ($\theta \in \mathbb{R}$) from Section 2.3 is used. In Figure 4.2, the training result of a linear PHMM with a left-right no-skip transition structure and 21 states is shown. The PHMM is trained with time restrictions ($d_{\text{min}} = 3$ and $d_{\text{max}} = 6$). The settings of the training data are basically the same as in Section 2.3.2, where the 10 movements have different progression rates (i.e., $\alpha = 0.3$, see Section 2.3) and are slightly disturbed by noise (see Section 2.3). The training with 10 Baum-Welch EM iterations takes 50s on a 2.4GHz Core2 single-core processor. In Figure 4.2, the transition structure and the output Gaussians are depicted for the HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\theta_0}$ with $\theta_0 = 0$. The shifted Gaussians of the HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\theta_0}$ are shown for the parameters $\theta_0 = 0.25$ (blue) and $\theta_0 = -0.25$ (magenta) as ellipses ($\sigma = 1.0$). As one can see, the covariances are very small and model basically only the variance within the training sequences for each specific parameter θ_0. The synthesis by interpolating linearly the means $\boldsymbol{\mu}_i^{\theta}$ for a parameter $\theta = \theta_0$ (Method D, Section 2.2.4) results in a trajectory (see Figure 4.3) which is very similar to the corresponding instance of the true parametric prototype movement $\hat{\boldsymbol{x}}^{\theta}(t)$ (Figure 2.6). In the Figures 4.4 and 4.5 the synthesis results are shown for a trained 6 state LPHMM and a trained 30 state LPHMM. Obviously, the results of the 6 state PHMM approximate the true prototypes only coarsely. The results of the 30 state PHMM are slightly distorted through the different progression rates of the training sequences. The synthesis of a 30 state PHMM, where the training sequences have the same progression rates ($\alpha = 0.0$), is shown in Figure 4.6.

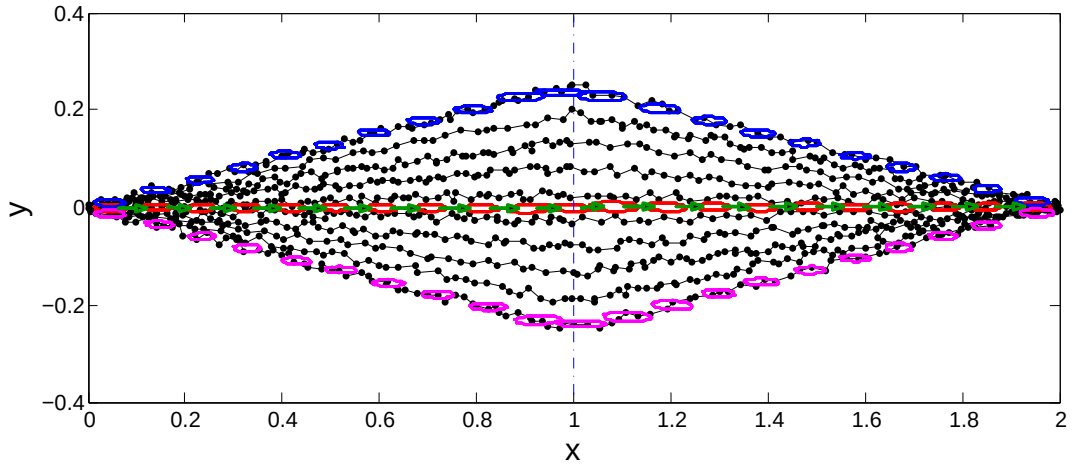


Figure 4.2: Trained Left-Right PHMM. The figure illustrates a left-right no-skip LPHMM with 21 states that is trained on the examples of the parametric movement $\hat{\boldsymbol{x}}^{\theta}(t)$ ($\theta \in \mathbb{R}$) that is discussed Section 2.3. The Gaussian output distributions and transition arcs are shown along the x-axis for a parameter $\theta = 0$. In addition, the output distributions are shown for the parameters $\theta = 0.25$ and $\theta = -0.25$.

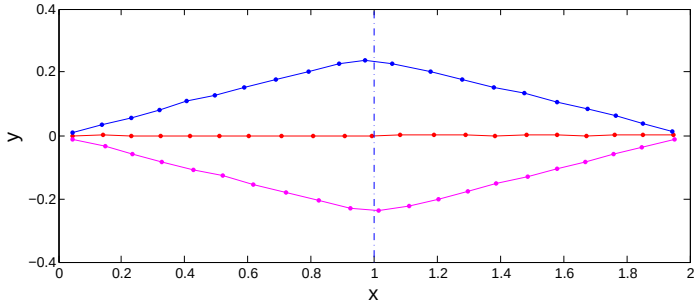


Figure 4.3: PHMM Synthesis: 21 States. The plot shows the synthesis by interpolating linearly the means of the PHMM (Figure 4.2) for the parameters $\theta = 0.25$, 0, and -0.25.

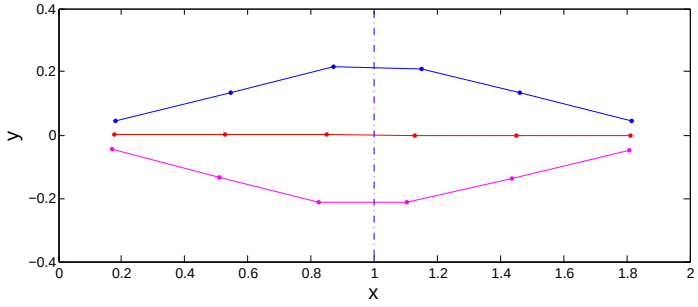


Figure 4.4: PHMM Synthesis: 6 States. The plot shows the synthesis results of a 6 state LPHMM (similar to Figure 4.3).

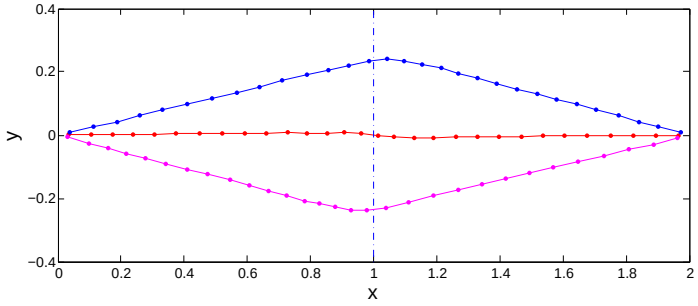


Figure 4.5: PHMM Synthesis: 30 States. The plot shows the synthesis results of a 30 state LPHMM (similar to Figure 4.3).

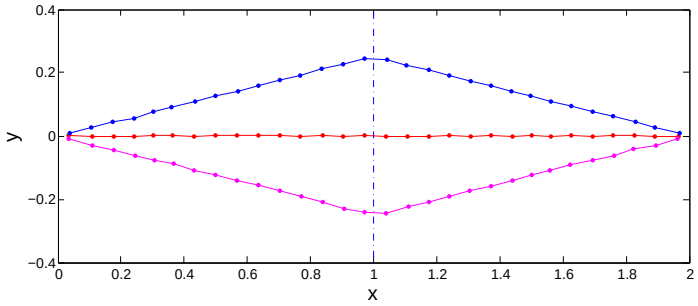


Figure 4.6: PHMM Synthesis: 30 States. Contrary to Figure 4.5, the utilized LPHMM is trained on movements which do not have different progression rates.

4.1.2 Utilizing the PHMM for Representing Parametric Movements

This section gives a short overview about how the PHMMs are used to deal with parametric movements. A single training set $\mathcal{X} = \{\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k\}$ of demonstrations of a parametric movement is assumed to be given, where each training example $\boldsymbol{X}^k$ is labeled by its associated parameter $\boldsymbol{\theta}_k$.

4.1.2.1 Model Training

The following settings are used for the PHMM training on parametric movement data. The PHMMs are initialized with a left-right no-skip transition structure (Section 2.2.1) with a preferably large number N of states. The PHMM are trained with time restrictions d_{min} and d_{max} (Section 2.2.3.2). The transition structure of the sub-states and the initialization of the transition matrix are discussed in Section 2.2.3.2. A finish state $N+1$ is introduced as discussed in Section 2.2.3.1.

Due to the time restrictions, the sequence lengths T_k have to be within the range of allowed sequence lengths $N \cdot d_{\text{min}}, \ldots, N \cdot d_{\text{max}}$. Preferably, the sequence lengths are about $N(d_{min} + d_{max})/2$, since otherwise the warping capability of the PHMM is compromised. For example, if $T_k = N \cdot d_{min}$ for all sequences, then there exists only one hidden state sequence (with nonzero probability) for all sequences. The following approach is used to handle this concern. The sequences are re-sampled to sequences of uniform length T close to $N(d_{\text{min}} + d_{\text{max}})/2$, e.g., by using linear interpolation. The average sequence length $\bar{T}_s$ or duration $\bar{T}_d$ of the original sequences can still be used to synthesis sequences of the appropriate length (if required).

The training sequences in this thesis are parameterized by one or two locations which are usually on a plane (as, for example, the table-top, see Figure 4.1). If the locations are on a plane, then a 2-dimensional vector is used to parameterize a location. (For a 3D parametrization of the locations, one requires also appropriate training data.) Another concern is that the training data should have sufficient examples. A further concern of the interpolative PHMM is that it requires training data for certain points in the parameter space, for which the local HMMs are set up.

4.1.2.2 Synthesis

The synthesis of a movement is very similar to the HMM-based synthesis. But in the case of a PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ a movement can be synthesized for any specific parameter $\boldsymbol{\theta}_0$. For a fixed parameter $\boldsymbol{\theta}_0$ the PHMM becomes an HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$, from which a prototype can be synthesized by using the Method D which is discussed in Section 2.2.4. A basic method is to interpolate the sequence of means linearly and to neglect the time durations. From the resulting prototype trajectory $\boldsymbol{x}(t)$, a sequence of desired length (e.g., the average sequence length $\bar{T}_s$) can be generated. If the temporal progress is important, then one would have to re-parameterize the trajectory $\boldsymbol{x}(t)$ as given by the average output durations of each state (Equation (2.28)). However, in Section 4.2.4.2 one can see that the output durations do not vary dramatically. It is worthwhile to note that a linear PHMM does not model the dependency of the progression rates of the sequences on $\boldsymbol{\theta}$, since the transition matrix is constant.

4.1.2.3 Recognition

For a given movement $\boldsymbol{X}$ and a PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ which models a parametric movement, the value of parameterization can be estimated in a maximum likelihood approach

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}}),$$

the maximization of the likelihood is discussed for the different PHMMs in in Chapter 3. In order to estimate the parameter, the likelihood function requires to be smooth and concave. In Figure 4.15, this is shown for a given pointing action.

The classification of a movement $\boldsymbol{X}$ in the case of several parametric movements classes $\mathcal{X}^k$, where each class is modeled by a PHMM $\boldsymbol{\lambda}_k^{\boldsymbol{\theta}}$, can be performed similarly as in the case of HMMs which is discussed in Section 2.2.5. The only difference is here that the parameter of the movement needs to be estimated for each PHMM in advance:

$$\boldsymbol{\theta}_{\mathrm{ML}}^k = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\lambda}^{\boldsymbol{\theta}_k}).$$

The movement can then be classified as belonging to the class k_{ML} with

$$k_{\mathrm{ML}} = \arg\max_{k} p(\boldsymbol{X}|\boldsymbol{\lambda}_k^{\boldsymbol{\theta}_{\mathrm{ML}}^k}).$$

However, for the last step also a more sophisticated method as discussed in Section 2.2.5 is reasonable. If one does not know whether a sequence belongs to any of the modeled movements, then the sophisticated method seems to be particularly important, otherwise one would have to rely on a simple thresholding mechanism. It is worthwhile to note that the classification method above provides also the estimate of the movement parameter(s), which cannot be estimated in a usual HMM approach to classification.

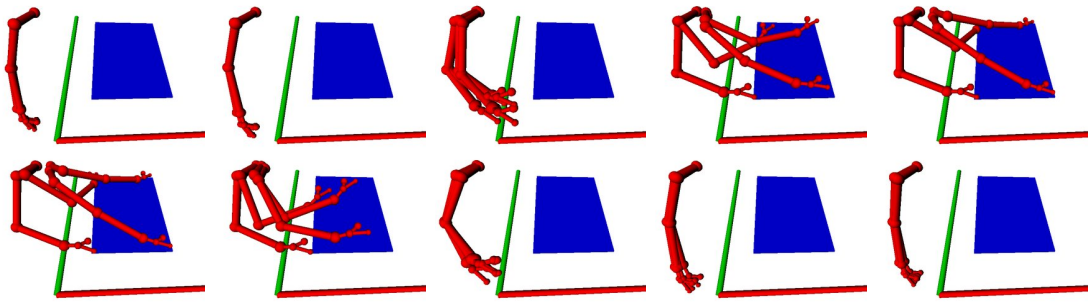


Figure 4.7: Pointing Action. The image sequence shows row-wise about every 11th sample of the recorded pointing actions (which are rescaled to 100 samples). The actions have been synchronized in advance. The values of the parametrization of the four instances of the parametric action correspond to the 4 corners of the used table-top region, which is shown here in blue. The seven recorded data points (of the recored trajectories) are indicated through the balls between the sticks of an arm. The index finger approaches quite accurately the corners, thus the finger can be identified in the first picture in the second row. The ball of the thumb is depicted slightly larger than the ball of the index finger. The red and the green bars indicate the x- and y-axis of the reference frame during the recording session. The unit length of both bars is 1m, but the red bar is cut at the image boundaries of the pictures.

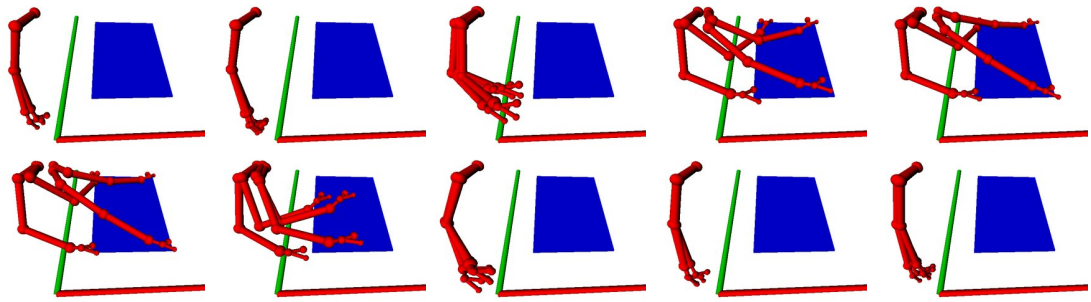


Figure 4.8: Reaching Action. Similar to Figure 4.7, every 11th sample of the recorded reaching actions is shown here. The reaching actions are performed very similarly to the pointing actions, but at the target point an object is grasped between thumb and index finger. In the first picture of the second row, the hand has grasped the object.

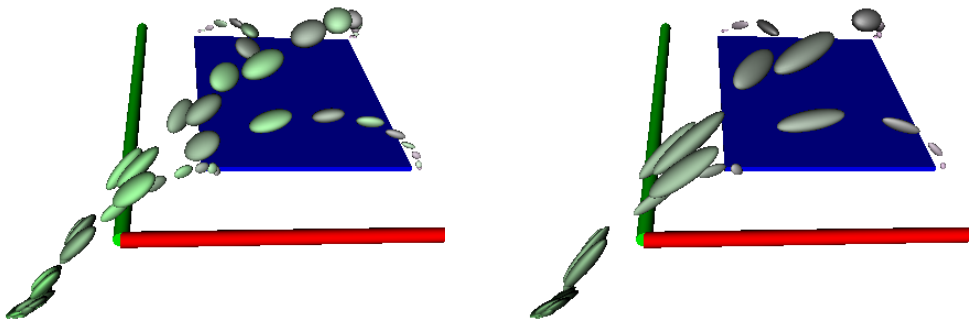


Figure 4.9: Gaussians of Local HMMs. The picture on the left shows the first 20 Gaussians of each of the 4 local HMMs of a 40 state 2×2 IPHMM trained on reaching actions. The picture shows the same for the first 10 states of a 20 state IPHMM.

4.2 Evaluation on Basic Parametric Actions

In this section, the PHMM-based recognition and synthesis of movements is evaluated. The focus is on pointing (Figure 4.7) and reaching actions (Figure 4.8), which are probably two of the most important movements in human-robot interaction scenarios. First the data sets and the trained PHMMs are described in Sections 4.2.1 and 4.2.2. In Section 4.2.3, it is explained how the linear PHMM is utilized as a quadratic PHMM. In Section 4.2.4, the synthesis performance of the interpolative, linear, and quadratic PHMMs evaluated. The time durations of PHMM states are shown in Section 4.2.4.2. The recognition is evaluated in Section 4.2.5.

4.2.1 The Data Set

The movement data for the evaluation is acquired with an eight camera visual marker motion capture system of Vicon. The setup of the capture session for data acquisition is as follows: The person/actor sits in front of a table, as shown in Figure 4.1. The pointing and reaching actions are performed in such a way that they are starting and ending in the same base pose, where the arm is hanging down (Figures 4.7 and 4.8). The recognition and synthesis experiments are based on the trajectories of seven 3D points located at different segments of the body. The seven data points are: the sternum; the shoulder, and the elbow of the right arm; the index finger, its knuckle, and the thumb of the right hand. Hence, the dimensionality is $D = 21$. The motion capturing is performed by aligning a model of the right arm (Figure 4.10) to the captured marker locations (the markers are shown in Figure 4.1). Some recorded actions of each type are shown in Figure 4.7 and Figure 4.8. Obviously, it is not easy to distinguish the pointing and reaching actions. The intention of using such actions, which are performed in a similar way, is to test the classification performance of the PHMMs in a non-trivial case. The pointing actions are actions such as pointing to a specific object in order to communicate a specific object ("This object

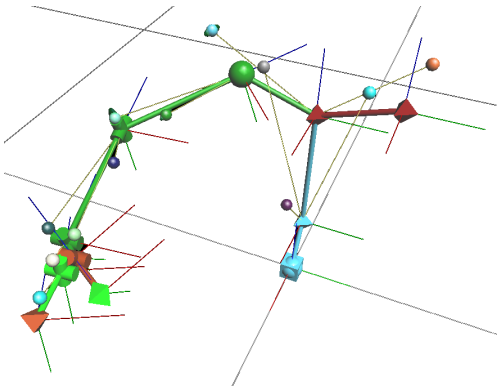


Figure 4.10: Human Model for Motion Capturing. The figure shows a upper body model of the person which is used for capturing arm actions. This model is designed to capture the important aspects of the body pose. The tiny colored balls of the model specify marker locations. The model and marker locations are calibrated for the person (Figure 4.1) that is captured. The Vicon system can reconstruct the pose of such a model with a high frame rate based on the (determined) correspondences between model markers and observed markers, which can be seen in Figure 4.1 as white points.

here is..."). The reaching actions are movements, where the person reaches for a particular object in order to grasp it. Both arm actions are characterized by the 2D target location at the table-top which is approached. Therefore, the 2D location is used as parameterization of the actions.

The active table-top region for which actions are recorded has a size of 80cm $\times$ 30cm (width $\times$ depth). A location within this region is specified through normalized coordinates $\theta = (u, v)$, where $u, v \in [0.0, 1.0]$. From the person's perspective (Figure 4.1), locations with $u = 0.0$ or $u = 1.0$ define the left and the right boundary of the region, respectively. Locations with $v = 0.0$ define the boundary close to the person.

For the recording of the training sequences, a regular 3×3 raster is used, where the raster points $(1, 1)$ and $(3, 3)$ correspond to the locations $(u, v) = (0.0, 0.0)$ and $(1.0, 1.0)$. For each raster point, 10 demonstrations of each action type are recorded. For evaluation, 4 demonstrations of each action type are recorded for each raster point of a denser 7×5 raster. The outermost raster points of the 7×5 raster match those of the 3×3 raster. Hence, the number of sequences for evaluation is 280. Each recorded movement sequence is labeled with the corresponding location $\boldsymbol{\theta} = (u, v)$. Finally, all the training and evaluation sequences are rescaled to a uniform length of $T = 100$ samples.

4.2.2 PHMM Training

Mainly, the interpolative PHMM (IPHMM) and linear PHMM (LPHMM) are considered. Both are trained on the movements as left-right (no-skip) PHMMs with $N = 20$ states and time restrictions $d_{\text{min}} = 4$ and $d_{\text{max}} = 6$, i.e., a hidden state sequence (with nonzero probability) can generate between 4 and 6 outputs from one state. A hidden state sequence (with nonzero probability) of the PHMMs has a length between 80 and 120. This is appropriate for the training and test sequences, which have a sequence length $T = 100$.

The PHMMs are trained either on all training sequences of the 3×3 raster or only on the sequences of the outermost points of the 3×3 raster. Depending on the training data, a PHMM is addressed as 3×3 IPHMM/LPHMM or 2×2 IPHMM/LPHMM. A 3×3 IPHMM uses 3×3 local HMMs for interpolation, whereas a 2×2 IPHMM uses only 2×2 local HMMs. Note, the internal structure of the LPHMMs is always the same, only the training data used is different. For clarity, 4 PHMMs are trained on pointing actions and 4 PHMMs are trained on reaching actions.

The synthesis performance of the linear PHMM is also compared with the quadratic PHMM (QPHMM) in Section 4.2.4.1. Details of the training of the QPHMM are provided in that section. The quadratic PHMM utilizes the linear PHMM as discussed in the following section.

4.2.3 The Quadratic PHMM

A straight forward approach to extend the linear PHMM (LPHMM) to a model, which uses a second-order polynomial to model the adaptation of the state means, is to use a higher dimensional parameterization than the actually needed in the case of a usual LPHMM. The additional parameters of the extended parametrization provide the squares and products of the original parameters. A LPHMM used in such a way is referred to as quadratic PHMM (QPHMM). The use of another polynomial, as, for example, a third-order polynomial, is straightforward. In such a case the PHMM is called cubic PHMM (CPHMM).

The extension is exemplified in the following for a linear PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ with a 2-dimensional parameterization $\boldsymbol{\theta} = (u, v) \in \mathbb{R}^{P'=2}$. The quadratic PHMM $\boldsymbol{\lambda}^{\boldsymbol{\Theta}}$ is basically a linear PHMM, where the parameterization is now a 5-dimensional vector $\boldsymbol{\Theta} = \boldsymbol{\Theta}(u, v) = (\Theta_1, \ldots, \Theta_{P''}) = (u, v, u^2, v^2, uv)$ (with $P'' = 5$), which provides for a given parameter vector $\boldsymbol{\theta} = (u, v)$ also the squares and the product of the components of $\boldsymbol{\theta} = (u, v)$.

Here, it is worthwhile to compare how the means are adapted in the linear and the quadratic PHMM. In the case of the linear PHMM, the mean $\boldsymbol{\mu}_i^{\boldsymbol{\theta}} \in \mathbb{R}^D$ of the Gaussian output distribution of a state i is depending on the parameter $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P) \in \mathbb{R}^P$ as given by

$$\boldsymbol{\mu}_i^{\boldsymbol{\theta}} = \bar{\boldsymbol{\mu}}_i + \boldsymbol{W}_i \, \boldsymbol{\theta}, \tag{4.1}$$

where $\boldsymbol{W}_i \in \mathbb{R}^{D \times P}$. For the linear PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ with $\boldsymbol{\theta} = (u, v) \in \mathbb{R}^{P'}$ the dependency can be rewritten as

$$\boldsymbol{\mu}_i^{(u,v)} = \boldsymbol{\mu}_i^{(\theta_1,\theta_2)} = \bar{\boldsymbol{\mu}}_i + \sum_{p=1}^{2} \boldsymbol{W}_i^p \theta_p = \bar{\boldsymbol{\mu}}_i + \boldsymbol{W}_i^1 u + \boldsymbol{W}_i^2 v,$$

where the matrix $[\boldsymbol{W}_i^1 | \boldsymbol{W}_i^2] = \boldsymbol{W}_i$ defines the linear dependency on the parameters u and v. In the case of the extended parameterization $\boldsymbol{\Theta}(u, v) = (\Theta_1, \ldots, \Theta_5)^\top = (u, v, u^2, v^2, uv)^\top$, the Equation (4.1) becomes for the quadratic PHMM:

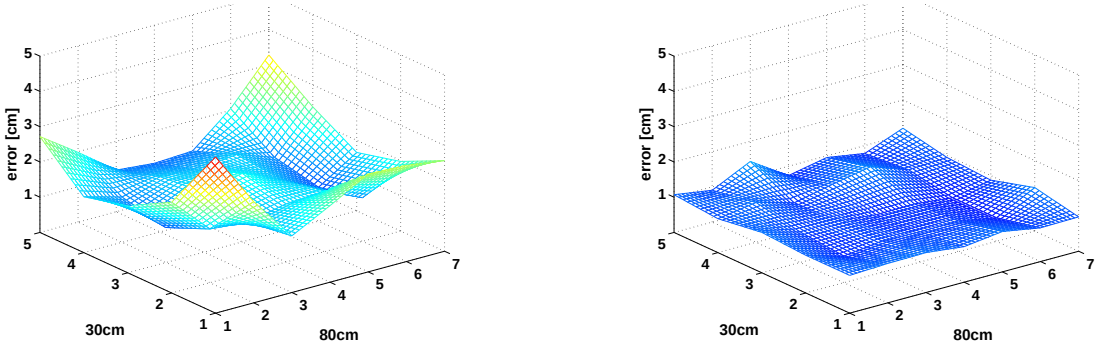
$$\boldsymbol{\mu}_i^{\boldsymbol{\Theta}(u,v)} = \boldsymbol{\mu}_i^{(\Theta_1,\ldots,\Theta_5)} = \bar{\boldsymbol{\mu}}_i + \sum_{p=1}^{5} \boldsymbol{W}_i^p \Theta_p = \bar{\boldsymbol{\mu}}_i + \boldsymbol{W}_i^1 u + \boldsymbol{W}_i^2 v + \boldsymbol{W}_i^3 u^2 + \boldsymbol{W}_i^4 v^2 + \boldsymbol{W}_i^5 uv, \tag{4.2}$$

where the row-vectors $\boldsymbol{W}_i^1, \ldots, \boldsymbol{W}_i^5$ of the matrix $\boldsymbol{W}_i = [\, \boldsymbol{W}_i^1 | \cdots | \boldsymbol{W}_i^5 \,] \in \mathbb{R}^{D \times P''}$ (with $P'' = 5$) are the coefficients of the quadratic polynomial in u and v.

For convenience, it is reasonable to neglect that a QPHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ with a 2-dimensional parametrization $\theta = (u, v)$ utilizes internally an LPHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}'}$ with a 5-dimensional parameterization $\boldsymbol{\theta}'$. The training of the QPHMM is performed as for the usual linear PHMM, but for each training sequence $\boldsymbol{X}^k$ the label $\boldsymbol{\theta}'_k = \boldsymbol{\Theta}(\boldsymbol{\theta}_k) = (u_k, v_k, u_k^2, v_k^2, u_k v_k)$ is used instead of the original sequence label $\boldsymbol{\theta}_k = (u_k, v_k)$. In order to synthesize a movement for a given parameter $\boldsymbol{\theta} = (u, v)$ on the basis of the state means, one has to evaluate first the means $\boldsymbol{\mu}_i^{\boldsymbol{\theta}'}$ for $\boldsymbol{\theta}' = \boldsymbol{\Theta}(u, v)$ (Equation (4.2)). The maximum likelihood estimate of the parameter $\boldsymbol{\theta} = (u, v)$ of a given sequence $\boldsymbol{X}$ is: $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{(u',v')} p(\boldsymbol{X} | \boldsymbol{\lambda}^{\boldsymbol{\Theta}(u',v')})$. The maximization can be performed, for example, with a gradient-ascend method.

4.2.4 Movement Synthesis

The performance of the synthesis is evaluated for each PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ and for each location $\boldsymbol{\theta}_0 = (u, v)$ of the raster points of the 7×5 raster in the same way. For the PHMM-based synthesis of a movement sequence with a target point $\boldsymbol{\theta}_0$ at the table-top, the sequence of means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N$ is extracted from the HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$, where the means are at the designated locations for the parameter $\boldsymbol{\theta}_0$. The sequence is then expanded to a sequence $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_{200}$ of 200 samples by using linear interpolation. The large number of samples is used to have a "smooth" representation of the trajectory that is given



(a) Pointing Syn., 2×2 LPHMM (b) Pointing Syn., 2×2 QPHMM

Figure 4.11: Synthesis Error (40-State PHMMs). ***Left:*** The plot shows how well the linear PHMM (LPHMM) can represent the pointing actions, which are available for the 7×5 (evaluation) raster. — ***Right:*** The plot shows how well the QPHMM (Section 4.2.3) can represent the pointing actions. Obviously, the synthesis errors of the QPHMM become quite small in comparison to the linear PHMM.

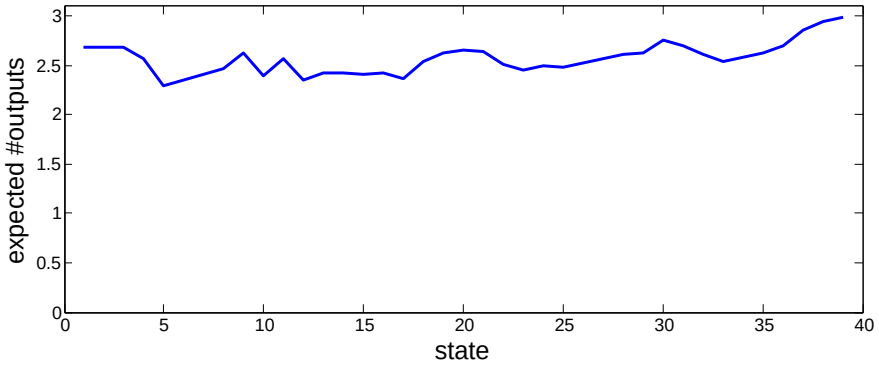


Figure 4.12: Time Durations of PHMM States. The figure shows for each state of an IPHMM the expected number of outputs. The 40 state IPHMM is trained on reaching actions with time restrictions $d_{\min} = 2$ and $d_{\max} = 3$.

by interpolating the means. This sequence is compared to the average of the 4 sequences (of the same movement type which is represented by the PHMM) that are available for the raster point for evaluation. The averaging of the 4 sequences is performed by using an 80 state left-right HMM (with time restrictions) that is trained on these sequences. The average sequence is then given by the means of the HMM. From this sequence, a sequence $\bar{\boldsymbol{X}} = \bar{\boldsymbol{x}}_1 \cdots \bar{\boldsymbol{x}}_{200}$ is generated with a large number of samples (200 samples). This sequence is called in the following a reference sequence. The samples $\boldsymbol{x}_i$ and $\bar{\boldsymbol{x}}_i$ are stacked vectors of the from $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^7)$, where the components $\boldsymbol{x}^i \in \mathbb{R}^3$ are the 3D locations of the elbow, wrist, etc. For the error calculation, both sequences, $\boldsymbol{X}$ and $\bar{\boldsymbol{X}}$, are time-warped (with the step pattern given by Equation (B.7)). The time-warped sequences are denoted as $\boldsymbol{X}' = \boldsymbol{x}_1 \cdots \boldsymbol{x}_{T'}$ and $\bar{\boldsymbol{X}}' = \boldsymbol{x}_1 \cdots \boldsymbol{x}_{T'}$. The overall error ε of the synthesized movement sequence $\boldsymbol{X}$ is then calculated as the root-mean-square error between the time-warped synthesis $\boldsymbol{X}'$ and the

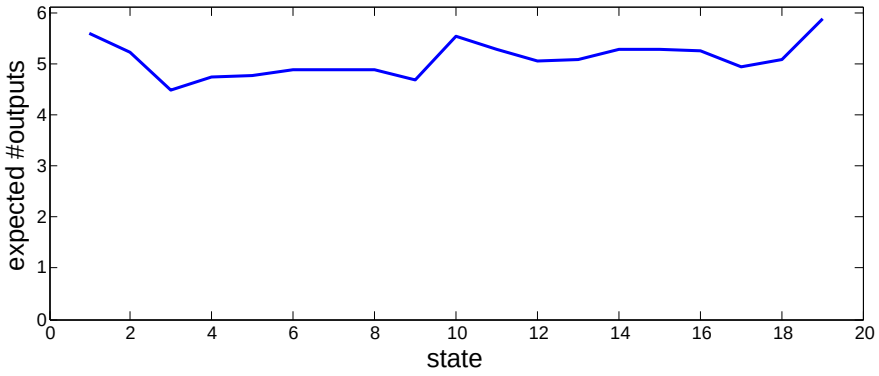


Figure 4.13: Time Durations of PHMM States. The figure shows for each state of an IPHMM the expected number of outputs. The 20 state IPHMM is trained on reaching actions with time restrictions $d_{\min} = 4$ and $d_{\max} = 6$.

reference $\bar{\boldsymbol{X}}'$:

$$\varepsilon = \sqrt{\frac{1}{0.70 \cdot T'} \cdot \frac{1}{7} \sum_{t=\lfloor 0.15 \cdot T' \rfloor}^{\lceil 0.85 \cdot T' \rceil} \sum_{i=1}^{7} (\boldsymbol{x}_t^i - \bar{\boldsymbol{x}}_t^i)^2}, \tag{4.3}$$

where $(\boldsymbol{x}_t^i - \bar{\boldsymbol{x}}_t^i)^2 = \|\boldsymbol{x}_t^i - \bar{\boldsymbol{x}}_t^i\|^2$. Due to the high variance of the base pose, the first 15% and the last 15% of the movements are neglected.

Figure 4.14 (a–d) shows the synthesis errors of the 20-state LPHMMs and IPHMMs trained on the pointing actions. The plots show the synthesis error for each raster point of the 7×5 evaluation raster. The synthesis errors of the reaching actions are very similar; these errors are neither shown nor discussed. For the 3×3 IPHMM, the synthesis for each raster point (i, j) with $i \in \{1, 4, 7\}$ and $j \in \{1, 5\}$ is based on one of the local HMMs of the IPHMM (the 3×3 IPHMM has 9 local HMMs). At these raster points, the synthesis error of the IPHMM indicates the smallest error which one would expect, since the synthesis does not rely on interpolation. The errors at the corners of the raster are approximately between 1.5cm and 2.5cm (Figure 4.14 (b)). At the center of the raster, the error is 1.1cm. One explanation of these errors is that the number of states of the PHMM is too small, but the errors are very similar when using 40 state PHMMs. Another explanation is the variance of the recorded performances for training and testing. In order to quantify this, the average overall distance between each movement pair of the movement data (including training and test data) is calculated. The distance between each pair is therefore calculated by Equation (4.3) after the sequences are aligned through time warping. The average distance for the pointing actions available for the center of the table-top region is 2.3cm with a standard deviation of 0.4cm. For the movements which belong to the corners, the average distance becomes comparably high. The highest value of one of the corners is 3.5cm with a deviation of 1.3cm.

For the 2×2 IPHMM, the errors increase significantly in the inner region due to interpolation. The errors increase in the inner region (if one neglects the region close to the boundaries of the table-top region) to 3.5cm, which seems to be still acceptable if one considers that the human performances can vary (at least at the corners) by several centimeters. In the inner region, the 3×3 LPHMM is more

accurate. The errors in the inner region are about 2.5cm. However, the linear constraints show their effect already for the 2×2 LPHMM at the corners.

4.2.4.1 Synthesizing from Quadratic PHMMs

The advantage of the linear PHMM in comparison to the 2×2 IPHMM or especially the 3×3 IPHMM (which performs better than the LPHMM) is that the LPHMM does not require that the training data is given for specific parameters which are on a raster/grid. (Note, the IPHMM requires this for the setup of the local HMMs.) The quadratic PHMM (QPHMM) is in this sense very similar to the linear PHMM. It is an extension of the linear PHMM as discussed in Section 4.2.3. The use of a second order polynomial for modeling the adaptation of the state means should enable the QPHMM to adapt better to the systematic variation of a parametric movement.

Figure 4.11 shows how well the QPHMM adapts to the pointing actions in comparison to the usual LPHMM. Both PHMMs are trained on the reference movements (see above) of the pointing actions which are available for the 7×5 evaluation raster. The error plot shows the errors between the synthesized movements and the references. The errors are calculated again by Equation (4.3). The behavior of the error surface of the linear PHMM (Figure 4.11) is similar to the surface of the 3×3 LPHMM (Figure 4.14 (d)). This shows that the LPHMM has a problem adapting to the systematic of the movements. Obviously, the quadratic PHMM adapts better. The errors are about 1cm over the whole region and vary only slightly (without a systematic), whereas the errors of the usual LPHMM surpasses 1cm almost everywhere.

4.2.4.2 Time Durations of PHMM States

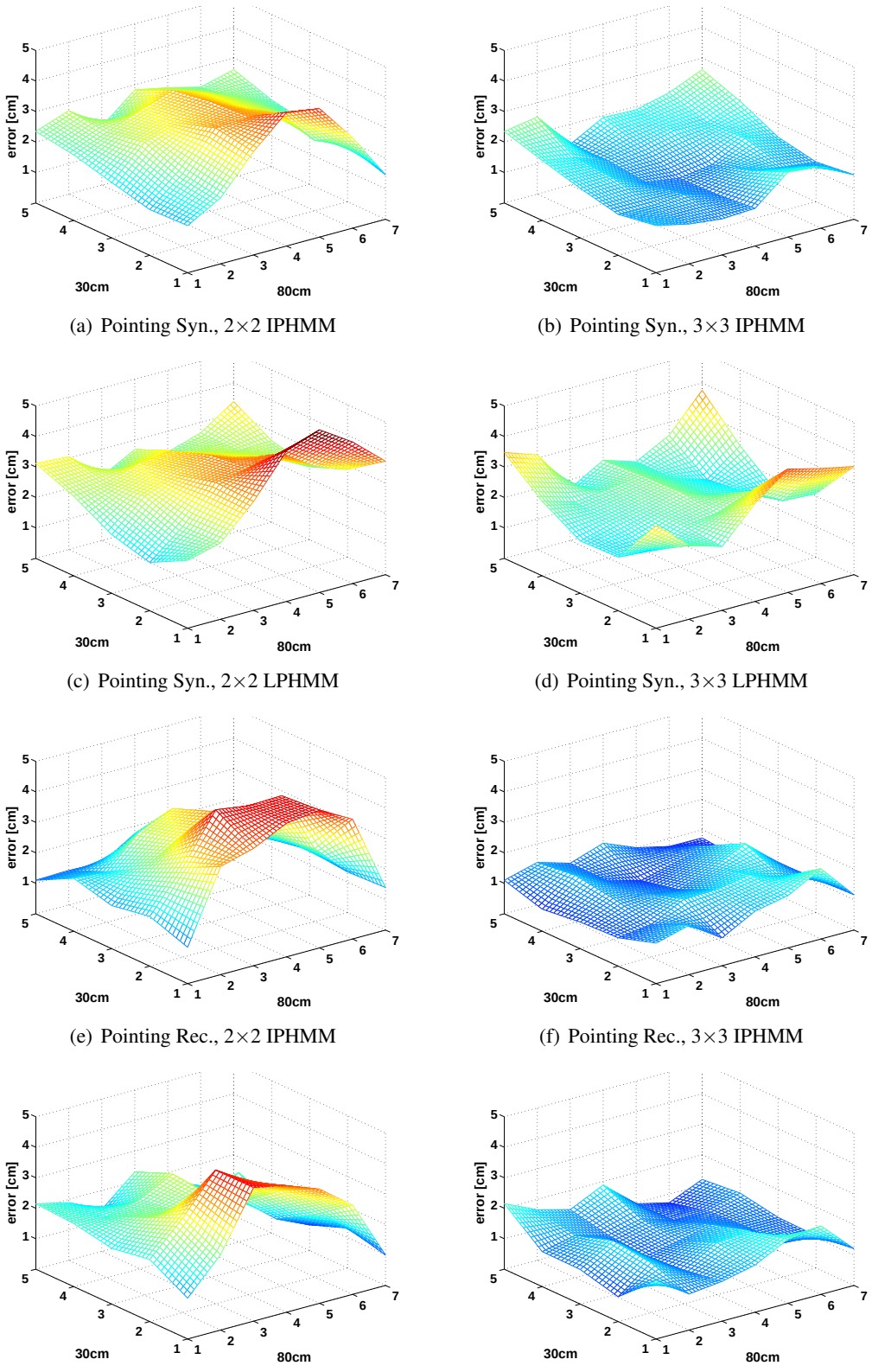
The Figures 4.12 and 4.13 show the expected number of outputs which state sequences (with nonzero probability) generate for each PHMM state. This is shown for a 20 state and a 40 state left-right PHMM. The PHMMs are trained with time restrictions. The training is performed on reaching actions. The expected number of outputs of a state is calculated by Equation (2.28). The expected number of outputs of a state can be interpreted as the duration during which the state is active (in average) when a sequence is generated (Section 2.2.3.2). If the temporal behavior of a synthesized sequence is important, then one should take the time durations of each state into account. However, obviously the output durations of the states are not changing dramatically.

4.2.5 Movement Recognition

Two aspects are considered in this section: 1. The recognition performance in terms of recognizing the correct parameter of a movement. 2. The classification of movements.

It is worthwhile to take first a look at Figure 4.15, which shows that the maximization of the log-likelihood $\mathcal{L}(u, v) = \log p(\boldsymbol{X}|\boldsymbol{\lambda}^{(u,v)})$ with respect to (u, v) for a given movement is solvable by standard optimization techniques (smoothness, and strict concavity). Moreover, the maximum is close to the true parameters of the movement.

Recognition. The recognition performance is evaluated for each IPHMM/LPHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ and for each raster point $\boldsymbol{\theta}_0$ of the 7×5 raster in the same way. For a single raster point $\boldsymbol{\theta}_0$ and a PHMM representing an action, the evaluation is performed as follows: the parameter $\boldsymbol{\theta} = (u, v)$ is estimated for each of the



(a) Pointing Syn., 2×2 IPHMM

(b) Pointing Syn., 3×3 IPHMM

(c) Pointing Syn., 2×2 LPHMM

(d) Pointing Syn., 3×3 LPHMM

(e) Pointing Rec., 2×2 IPHMM

(f) Pointing Rec., 3×3 IPHMM

(g) Pointing Rec., 2×2 LPHMM

(h) Pointing Rec., 3×3 LPHMM

Figure 4.14: Synthesis and Recognition Error (20-State PHMMs).

4 recorded actions available for evaluation. The error is the distance between the estimated table-top location (given by $\boldsymbol{\theta}$) and the true location of the raster point. The four errors are then averaged.

In Figure 4.14 (e–h), the errors are shown for the PHMMs trained on pointing actions. The errors for the reaching action are very similar, and are not shown.

As one can see, the recognition errors for the 2×2 IPHMM and 2×2 LPHMM correlate with the synthesis errors. Both models rely in the inner region heavily on interpolation (either bi-linear or linear), since both models are trained without using the training data available for the inner region. The recognition error of the 2×2 IPHMM is about 3.5cm at the center. The errors become up to 4.4cm at the boundary $v = 0$ of the region. The recognition errors of the 3×3 LPHMM and IPHMM are basically not above 2cm. The recognition error of the 3×3 PHMMs can be explained, to some degree, through the performance of the movements through the human. The person missed the true raster points slightly. The average deviation from the true location is in the middle of the raster about 0.8cm. In addition, one has to take into account that the parameters are estimated based on a whole performance. Since a performance starts and ends with the arm hanging down beside the person, it seems to be unavoidable that the person has to correct the arm motion during the performance in order to attain the right target location at the table-top. In such a case, the first part of the performance might be misleading for the estimation of the target location. However, it is interesting to see that the properly trained LPHMM (namely the 3×3 LPHMM) yields amazingly good recognition results. An explanation would be that the LPHMM can capture those parts of the systematic variation which it is incapable of modeling as variance in the Gaussian output distributions.

The robustness of the recognition has been tested by adding Gaussian noise to each component of the samples of the movements. Here, no significant influence for independently distributed noise could be detected when the noise is generated with $\sigma < 15\text{cm}$. Obviously, that is caused by the large number of samples of a sequence.

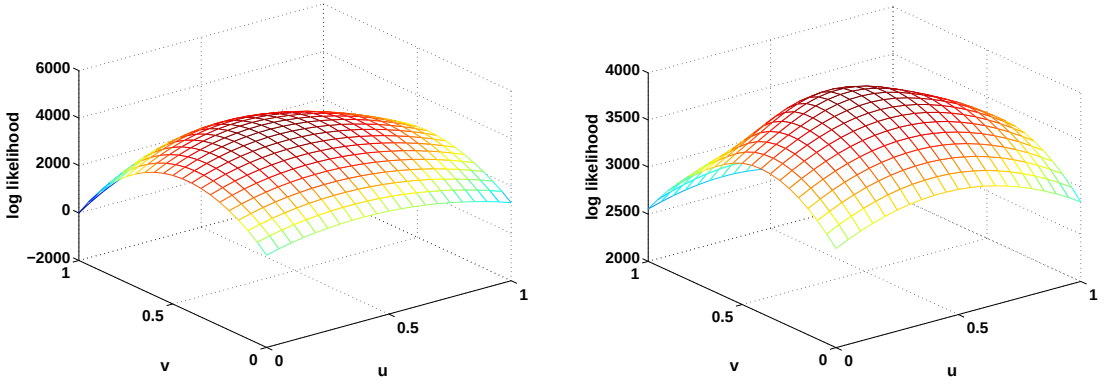


Figure 4.15: Plot of Log-Likelihood Functions. The plot on the left shows the log-likelihood function $\mathcal{L}(u, v) = \log p(\boldsymbol{X}|\boldsymbol{\lambda}^{(u,v)})$ of a 40-state IPHMM trained on pointing actions. On the right the plot is given for a LPHMM. The movement $\boldsymbol{X}$ is a pointing action with parameters $(u, v) = (0.5, 0.5)$. Both log-likelihood functions are smooth and concave, and have global maximums which are close to the true parameters $(u, v) = (0.5, 0.5)$ of the action.

Classification. The classification of a movement is performed as discussed in Section 4.1.2.3. The classification is performed here for each pair of PHMMs of same type, where one PHMM represents

pointing actions and the other reaching actions. The rate of correctly classified actions is shown in Table 4.1.

Table 4.1: Classification Rate. The table shows the classification rates for the PHMM-based classification of 280 pointing and reaching actions. The rates are given for different numbers of PHMM states.

States	2×2 IPHMM	2×2 LPHMM	3×3 IPHMM	3×3 LPHMM
20-States	0.9286	0.9000	0.9786	0.8821
40-States	0.9429	0.8929	0.9821	0.8643

4.3 Learning a Bi-Parametric Movement

In this section, the learning of a more complicated movement is exemplified. The movement is a parametric redirect action $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$ that is parameterized by two table-top locations $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$, where $\boldsymbol{\theta}_1 = (u_1, v_1)$ and $\boldsymbol{\theta}_2 = (u_2, v_2)$ determine the pointed to locations at a table-top. The table-top settings and the parameterization of the locations are the same as in the section above. The parameters u_i, v_i are normalized, i.e. $u_i, v_i \in [0, 1]$. The action is somehow similar to the parametric pointing action discussed in the section above. It begins and ends in the same base pose as the pointing action. But the redirect action points first to a location $\boldsymbol{\theta}_1$ and then to the location $\boldsymbol{\theta}_2$ in order to communicate: "Take this object here, and put it at that location." In Figure 4.16, the redirect action is shown, where the first pointed to location is $\boldsymbol{\theta}_1 = (0, 1)$. The second pointed to location is $\boldsymbol{\theta}_2 = (1, 1)$.

The training examples of the redirect action are generated by motion blending based on an 80 state 2×2 PHMM which is trained on the pointing actions of the section above. Based on the synthesized training examples, a linear PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ (left-right PHMM with time restrictions and 32 states) is trained, where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ is used as parameterization. The training set for the PHMM training consists of

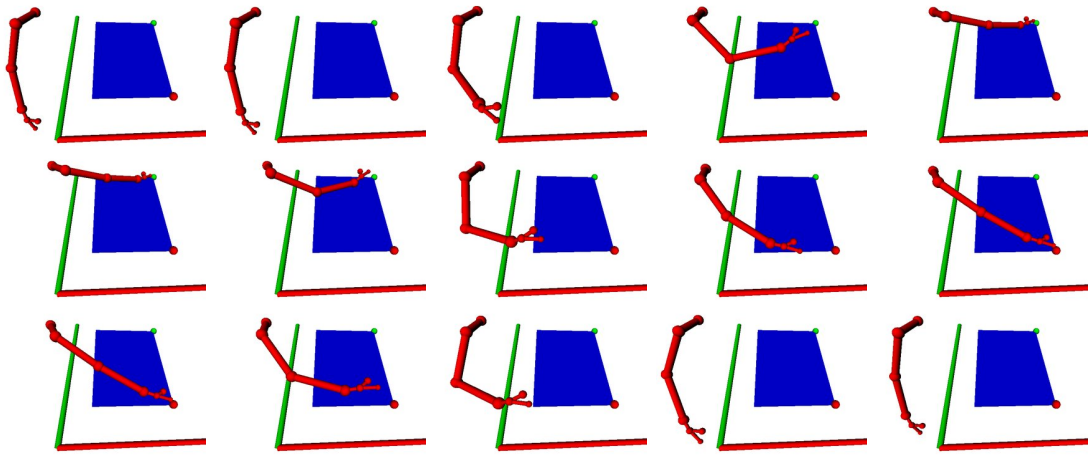


Figure 4.16: Redirect Action. The image sequence shows row-wise approximately every 6th sample of a redirect action which has 80 samples. The first pointed to location $\boldsymbol{\theta}_1$ is given by the green ball, the second pointed to location $\boldsymbol{\theta}_2$ is given by the red ball.

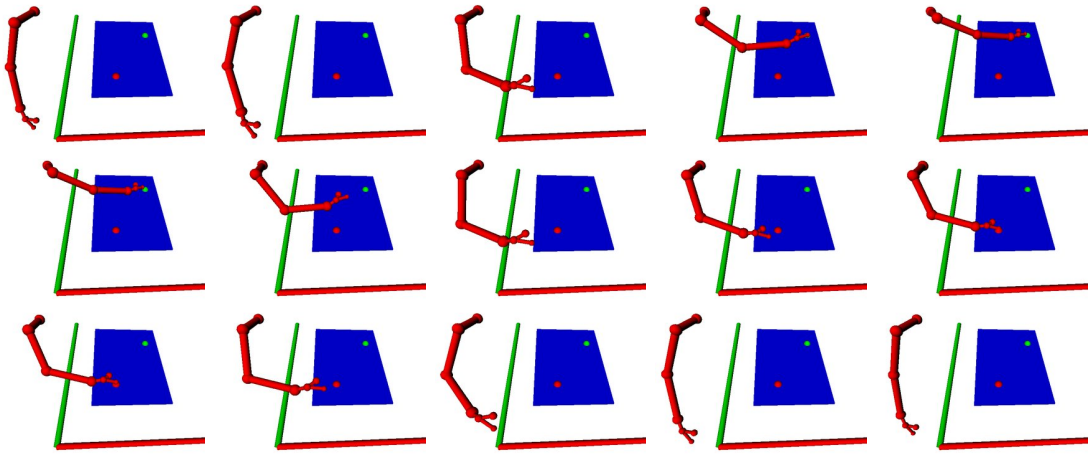


Figure 4.17: Synthesized Redirect Action. The image sequence shows approximately the same states of progress as Figure 4.16. But here the action is synthesized from a PHMM trained on redirect actions. The action is synthesized with a parameter $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$, where the pointed to locations $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are given by the green ball and the red ball, respectively.

10 training movements $\boldsymbol{X}^{\boldsymbol{\theta}=(\boldsymbol{\theta}_1,\boldsymbol{\theta}_2)}$, where $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \{(0,0),(1,0),(1,0),(1,1)\}$, i.e. one movement from each corner to each corner. Each training sequence has 80 samples.

In Figure 4.17, a synthesis of the parametric redirect action on the basis of the trained LPHMM is shown for a parameter $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ with $\boldsymbol{\theta}_1 = (0.25, 0.80)$ and $\boldsymbol{\theta}_2 = (0.80, 0.30)$. The locations $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ at the table-top are indicated by the green ball and the red ball, respectively.

4.4 Summary

In this chapter, different parametric hidden Markov models were evaluated for representing parametric human movements. The application of PHMMs for representing parametric movements was discussed (Section 4.1). The time-warping and noise modeling capabilities of the HMM framework combined with the adaptivity of the parametric extension of HMMs make the PHMM an appropriate model for learning parametric human movements from demonstrations: The adaptivity of the parametric extension enables the model to learn (with restrictions) the systematic variation of the human movement. The time-warping capability allows the model the handling of different progression rates of human movements, whereas the capability to model noise allows the model to capture the non-systematic spacial variation of the human performances. (Humans cannot perform the same movement twice with arbitrary precision.)

The PHMM representation can be used to recognize and synthesize a learned parametric movement. The representation of a parametric movement through a PHMM with a large number of states allows the synthesis of very accurate movements for arbitrary parameters, as shown for the synthetic parametric movement in Section 4.1.1. In this thesis the synthesis is understood as generating a good prototype movement (for a given parameter vector) as discussed in Section 4.1, which is clearly different from the PHMM-based synthesis that is performed in [132], where the goal is rather to synthesize movements which are within the variance of the training movements (for given parameter vector).

However, the ability of underlying model of a PHMM for adapting the means is crucial for rep-

resenting a parametric movement accurately. For example, the considered synthetic movement (Section 4.1.1) varies linearly depending on its parameter. Thus, the linear PHMM can accurately model the variation by the linear adaptation of the state means. In the synthesis experiments (Section 4.2) on the pointing and reaching actions, the quadratic PHMM could explain the systematic variation of the pointing action with an overall accuracy of about 1cm. One could argue that the inaccuracy is to some part still a result of the inaccuracy of the human's performances, which can also complicate the systematic variation. For the linear PHMM (especially for the inaccurately trained 2×2 LPHMMs) and the 2×2 interpolative PHMM, the synthesis errors were up to 3.5cm. It is worthwhile to note that the $80\text{cm} \times 30\text{cm}$ region of the table-top is very large (the person was not able to reach beyond the boundaries at the corner), for a smaller region the error would be much smaller as the errors of the 3×3 IPHMM indicate. In the inner region, the errors of the 3×3 IPHMM are about 2cm.

The performance in recognizing the target locations (parameters) of pointing and reaching actions is for the linear PHMM (3×3 LPHMM) fairly accurate: the error is here over the whole region basically not above 2cm, where approximately 1cm of the error could be explained through the inaccuracy of the human performer in reaching the actual target locations. The parameter recognition errors of the 2×2 IPHMM and the 2×2 LPHMM are in the same range as for the synthesis. As the performances of the pointing and reaching actions are very similar, the classification rate of up to 98% (in the case of the 3×3 IPHMM) seems to be fairly accurate. The LPHMMs achieved classification rates of up to 90%. It is worthwhile to note that the recognition and classification experiments are performed without further means as, for example, increasing the influence of the finger components of those states which model an arm pose, where the finger is at the target location. For parametric arm movements (similar to the actions considered here), classical HMMs achieve such a classification rate only on the basis of a tuned feature set [94].

Finally, Section 4.3 shows that more complicated bi-parametric movements can be learned by a bi-parametric HMM. The learned redirect action is parameterized by two target locations which are approached in sequence. The trained PHMM allows one to synthesize the action for arbitrary target locations. Note, such a bi-parametric HMM can be used to represent/learn bi-parametric action primitives, which can be used as basic building block for complex actions.

Chapter 5

Imitating Movements by Humanoid Robots

In this chapter the direct imitation of parametric human movements by humanoid robots is discussed.

Beside the even more complicate questions, such as how to acquire movements from observing a person or how to interpret demonstrations that show how one can accomplish a certain task (see Section 1.1.1), the direct imitation of basic arm actions, such as pointing or reaching actions that are demonstrated and learned in a supervised way (as in the chapter above), gives rise to further questions.

For the direct imitation, one has to decide how the acquired (recorded) movements are represented. This concerns, for example, the encoding of arm poses, e.g., as joint angles or locations (and orientations) of body-fixed points. On the other hand, it is not obvious how to imitate if the embodiment of the imitating robot differs from the demonstrator. In the case of an arm gesture such as waving the hand to communicate to another person "fare well", one could argue that the appearance of the movement should be similar. In this case, it might be sufficient to reproduce the observed sequence of joint angles of the demonstrator. But this is insufficient for an action such as reaching for an object. As an example, a demonstrator is standing in front of a table and reaches for an object at the table-top. If an imitator with shorter arms replaces the demonstrator in the same setting (objects etc. at the same locations), then the imitator would miss the object by reproducing the same sequence of joint angles. For imitating such an action, the crucial aspect seems to be the effect of reaching for a specific object rather than the reproduction of a motion with a similar appearance. As a consequence, the meaning/effect of a demonstrated movement and how the reproduction of the movement modifies the meaning/effect of the reproduced movement needs to be understood.

Another example emphasizing the problematic nature of the meaning of movements is a pointing action. If the pointing actions are demonstrated for objects that are close to the person and are demonstrated in such a way that the finger finally reaches the designated object, then the aspect which conveys the meaning of the action is the location that is reached (in order to identify the object). The imitation for the purpose of indicating an object should reproduce the meaning (the pointed to location) rather than reproducing a movement with the same appearance. But the meaning of a pointing action can be conveyed also in a different way. If the objects are far away, then the demonstrator would simply point in the direction of the object. The meaning is then not given by the location that is reached by the finger.

To summarize, there is no obvious way to conclude from a demonstrated movement the ef-

fect/meaning. If the effect/meaning is provided in a supervised manner for the demonstrated movements, the reproduction of the movement on another embodiment still may change the meaning/effect.

One option to approach this problem would be to establish a mapping of observed movements onto another embodiment in such a way that the effect/meaning is preserved. Another option is to model how the meaning/effect of a reproduced movement has to be re-interpreted. In the latter case, one could select a movement (if accessible) which has the desired effect when reproduced. Since the humanoid robot, which is considered in this chapter, is much smaller than the human demonstrator, the following approach is used: the reproduced movements preserve some characteristics of the movements such that the relation between the effect/meaning of the demonstrated movement and the effect/meaning of the reproduced movement becomes simple.

In Chapter 4, the learning of parametric movements through PHMMs and the synthesis of movements are discussed without considering a different embodiments. Section 5.1, the above considerations of mapping movements and re-interpreting their effect/meaning are elaborated in the context of parametric movement models which represent a movement in an effect/meaning dependent way. Subsequently, an implementation for the imitation of arm actions on the humanoid HOAP-3 is discussed in Section 5.2. This discussion includes the mappings, details of the robot, the reproduction of the synthesized movements on the robot, the imitation error, and a rule-learning application. A short summary is given in Section 5.3.

5.1 Imitating Parametric Movements on Different Embodiments

In the Section 4.2, a parametric hidden Markov model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ for a parametric movement is trained based on demonstrations performed by a single person. The parametrization $\boldsymbol{\theta}$ of the movement (model) is chosen in order to describe the effect/meaning of the movement in a suitable way. The model is then trained on a set of recorded sequences $\boldsymbol{X}^k = \boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_{T_k}^k$ and labels $\boldsymbol{\theta}_k$, where the vectors $\boldsymbol{x}_t^k$ encode the body poses in a chosen representation. The labels provide the effect/meaning of the demonstrations. The labels are provided by a supervisor. The trained movement model can then be used to synthesize a prototype trajectory $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$ for a parameter $\boldsymbol{\theta}$, where the trajectory lives in the space of the used representation for the body poses. If one assumes now that the model is precise, then the reproduction of the trajectory $\boldsymbol{x}^{\boldsymbol{\theta}}(t)$ on an embodiment as the one of the demonstrator results in a motion that has the effect/meaning as given by the parameter $\boldsymbol{\theta}$.

In the following considerations, it is not essential that the model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ is a PHMM. The model could be any model that can be used to synthesis precise prototype trajectories of a parametric movement for a given parameter $\boldsymbol{\theta}$. In the following, the transfer of the motion data and/or the models to another embodiment is considered. Therefore, it is assumed that the representation $\boldsymbol{x}$ of a body pose of the demonstrator can be mapped to a representation $\hat{\boldsymbol{x}}$ that is usable for another specific embodiment (in the following addressed as humanoid or robot). The mapping $\boldsymbol{x} \mapsto \hat{\boldsymbol{x}}(\boldsymbol{x})$ may be complicated (e.g., the conversion from body-fixed points of the demonstrator to the joint angles of the robot by using inverse kinematics) or may be as simple as the identity $\hat{\boldsymbol{x}}(\boldsymbol{x}) = \boldsymbol{x}$, which is, for example, the case when the body pose of the demonstrator is described in joint angles which are used without a further conversion to control the robot. Furthermore, it is assumed that there exists an underlying mapping $\boldsymbol{\theta} \mapsto \hat{\boldsymbol{\theta}}(\boldsymbol{\theta})$ which gives the meaning/effect $\hat{\boldsymbol{\theta}}$ of a trajectory $\hat{\boldsymbol{x}}(t)$ that is reproduced on the robot, where $\hat{\boldsymbol{x}}(t)$ is the result of a conversion (based on the mapping $\boldsymbol{x} \mapsto \hat{\boldsymbol{x}}(\boldsymbol{x})$) of a trajectory

$\boldsymbol{x}(t)$ that has the meaning/effect $\boldsymbol{\theta}$ on the demonstrator's embodiment. The mapping $\boldsymbol{\theta} \mapsto \hat{\boldsymbol{\theta}}(\boldsymbol{\theta})$ of the meaning/effect can be quite simple. For example, if the mapping of the body pose $\boldsymbol{x} \mapsto \hat{\boldsymbol{x}}(\boldsymbol{x})$ preserves the meaning/effect, the mapping is the identity $\hat{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \boldsymbol{\theta}$.

In the following, some different approaches are considered which allows one to reproduce a movement trajectory $\hat{\boldsymbol{x}}(t)$ on the humanoid which has the desired meaning/effect $\hat{\boldsymbol{\theta}}$. It is assumed in the following that a mapping $\boldsymbol{x} \mapsto \hat{\boldsymbol{x}}(\boldsymbol{x})$ of the body pose is given.

1. a) If the inverse mapping $\hat{\boldsymbol{\theta}} \mapsto \boldsymbol{\theta}(\hat{\boldsymbol{\theta}})$ of the mapping $\boldsymbol{\theta} \mapsto \hat{\boldsymbol{\theta}}(\boldsymbol{\theta})$ can be computed, then one can generate a movement on the robot with the desired effect/meaning $\hat{\boldsymbol{\theta}}$ as follows. First, $\boldsymbol{\theta} = \boldsymbol{\theta}(\hat{\boldsymbol{\theta}})$ is determined. Then, the movement trajectory $\boldsymbol{x}(t)$ for the parameter $\boldsymbol{\theta}$ is synthesized and converted to a trajectory $\hat{\boldsymbol{x}}(t)$ by using the mapping $\boldsymbol{x} \mapsto \hat{\boldsymbol{x}}(\boldsymbol{x})$. The trajectory $\hat{\boldsymbol{x}}(t)$ has then the desired effect/meaning $\hat{\boldsymbol{\theta}}$ when reproduced on the robot's embodiment.

   b) If the inverse mapping of the effect/meaning cannot be computed, then an iterative method might be still suitable. Assume the effect/meaning $\hat{\boldsymbol{\theta}}$ of a movement $\hat{\boldsymbol{x}}(t)$ can be predicted/measured. Then, one can evaluate the function $\hat{\boldsymbol{\theta}}(\boldsymbol{\theta})$ for a value $\boldsymbol{\theta}$ by synthesizing $\boldsymbol{x}(t)$ and predicting/measuring the effect/meaning $\hat{\boldsymbol{\theta}}$ of the converted movement $\hat{\boldsymbol{x}}(t)$. This allows one to use standard optimization techniques.

2. a) Alternatively to converting the trajectories, one could try to convert the model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ to a model $\hat{\boldsymbol{\lambda}}^{\boldsymbol{\theta}}$, which generates the desired trajectory $\hat{\boldsymbol{x}}(t)$ with effect/meaning $\hat{\boldsymbol{\theta}}$ if one uses the parameter $\boldsymbol{\theta} = \boldsymbol{\theta}(\hat{\boldsymbol{\theta}})$.

   b) Assume the mapping $\boldsymbol{\theta} \mapsto \hat{\boldsymbol{\theta}}(\boldsymbol{\theta})$ or $\hat{\boldsymbol{\theta}} \mapsto \boldsymbol{\theta}(\boldsymbol{\theta})$ is given. Then one could try to convert the model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ to a model $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{\theta}}}$, which directly generates a trajectory for the desired effect/meaning $\hat{\boldsymbol{\theta}}$ on the robot.

3. Assume the mapping $\boldsymbol{\theta} \mapsto \hat{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is given. Instead of converting the model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$, one could also train a model $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{\theta}}}$ if the training sequences $\boldsymbol{X}^k = \boldsymbol{x}_1^k \cdots \boldsymbol{x}_{T_k}^k$ with labels $\boldsymbol{\theta}_k$ are still available. The training data are then: $\hat{\boldsymbol{X}}^k = \hat{\boldsymbol{x}}(\boldsymbol{x}_1^k) \cdots \hat{\boldsymbol{x}}(\boldsymbol{x}_{T_k}^k)$ with labels $\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}(\boldsymbol{\theta}_k)$.

4. The latter approach, allows one also the use of demonstrations from several demonstrators with different embodiments if the mappings of the body pose and the effect/meaning for each demonstrator are known.

5. The last approach could be generalized in the following manner. The training sequences and labels (acquired for different embodiments) are converted to an intermediate representation (e. g., by the use of a reference model of the human body). Then an intermediate model $\bar{\boldsymbol{\lambda}}^{\bar{\boldsymbol{\theta}}}$ is trained based on the training data. This intermediate model could then be used for several target embodiments by using one of the approaches above.

The idea of using an intermediate representation is not new. In [8] a representation of motion data w.r.t. a reference model of the human body (the so called Master Motor Map) is used. Motion data can be converted from different sources to this representation. The representation can then be converted for different target embodiments. However, parametric movements are not considered.

In the following implementation Section 5.2, the approach 2. is used with a slight modification. A model $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ is converted to a model $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{\theta}}}$. A synthesized movement $\hat{\boldsymbol{x}}(t)$ for a parameter $\hat{\boldsymbol{\theta}}$ is then

converted by an additional mapping to a movement $\hat{\boldsymbol{x}}'(t)$ which is then reproduced on the embodiment. The additional mapping preserves the effect/meaning, i.e. $\hat{\boldsymbol{x}}'(t)$ has the effect/meaning $\hat{\boldsymbol{\theta}}$.

5.2 Imitating Arm Movements

In this section the imitation of a parametric human reaching action by the humanoid robot HOAP-3 is discussed. The implementation could be easily adapted to perform the imitation of certain other arm movements, for example, a pointing action, in which the robot would point at an object at an arbitrary table-top location. The humanoid platform HOAP-3 and the methodic to use the robot for reproducing a sequence of body configurations are described in Section 5.2.1. The used parametric reaching action and the representation through a parametric hidden Markov model are described in Section 5.2.2. The model is then converted to a model which is used to synthesize the sequences, which can be mapped to body configurations that can be reproduced by the humanoid. The model conversion is discussed in Section 5.2.3. The final mapping of the arm poses to the robot's embodiment is discussed Section 5.2.4. The reaching actions are then used to let the robot relocate objects (discussed in Section 5.2.5). This capability is finally used in a rule-learning application (Section 5.2.6), where the robot has to learn how to deal with several objects on a table-top. The accuracy of the approach to imitate the arm movements is discussed in Section 5.2.7. (The approach to imitation is published in the work Herzog et al. [50].)

Figure 5.1: HOAP-3 Performing a Tai Chi Exercise.

5.2.1 The Humanoid Robot

The HOAP-3 robot by HONDA is a small humanoid robot with a head height of about 60cm. The robot is shown in Figure 5.2. The robot has a backpack with a battery and personal computer (PC) control unit. The robot is, for example, able to walk (or even to perform tai chi exercises, see Figure 5.1) and manipulate objects. It can be controlled via wire and also wireless from a command PC, which is using real-time Linux. The robot has several input sensors and can articulate its body with 28 joints (each joint has one degree of freedom) by several servos in a very human like way. The head has two cameras (eyes), a distance sensor and a microphone at the top of the head and a speaker (in the upper chest). Other sensors are a grip sensor in the thumbs, foot bottom sensors, and a posture sensor consisting of an acceleration sensor and an angular velocity sensor which is in the upper half of the body. The sensors in the feet enables the robot to determine the distribution of the weight on the foot

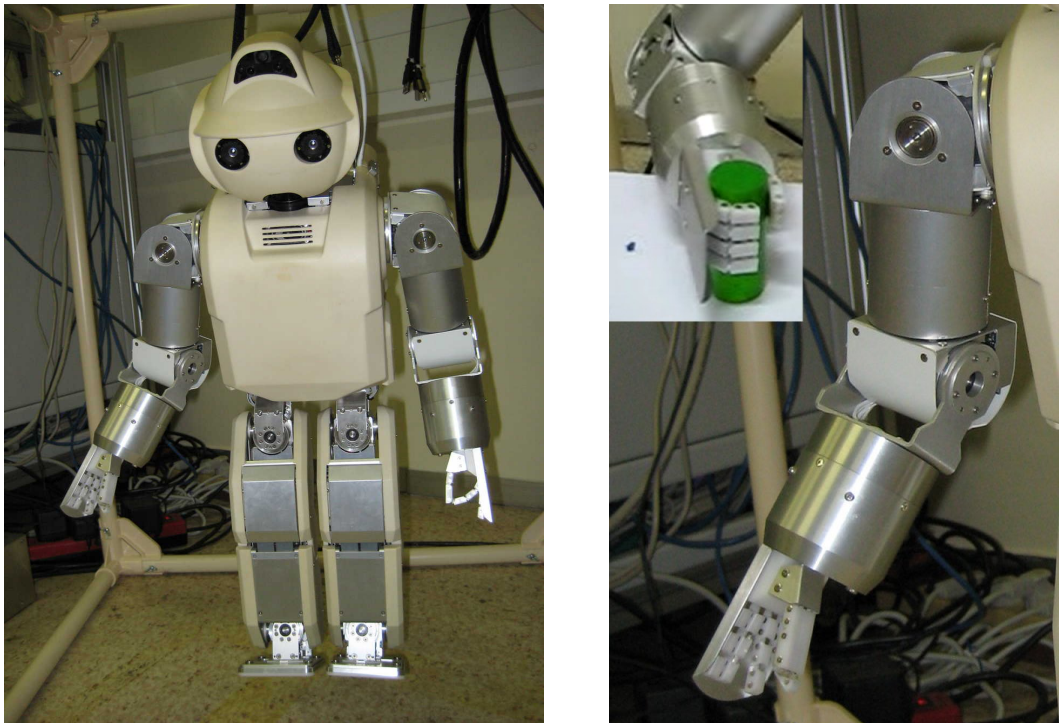


Figure 5.2: The Humanoid Robot HOAP-3. ***Left:*** The picture shows the humanoid robot HOAP-3 hanging at a lifting belt with no ground contact. — ***Right:*** The picture shows a close up of the robot's right arm. In the small picture, the robot has grasped a green cylinder. The thumb and the four finger of the right hand hold the object.

Table 5.1: Joint Angle Value Ranges of HOAP's Right Arm. The table shows the ranges of the joints of HOAP's right arm. The joints $\boldsymbol{j}_i$ are defined in Figure 5.3. In addition, the corresponding integer values are given which are used for controlling the robot. The integer values and maximum/minimum values (in degrees) of the joints $\boldsymbol{j}_1, \ldots, \boldsymbol{j}_4$ are taken from the robot's manual. For these the quantization factors to convert degrees to integers are ± 209. The maximum/minimum degrees of the joints $\boldsymbol{j}_5$ and $\boldsymbol{j}_6$ are taken from the manual, but the integer values have been determined empirically. The integer value 120 for $\boldsymbol{j}_6$ has the effect that the gripper is fully closed, the value 0 opens the gripper as far as possible. The degrees of the joint $\boldsymbol{j}_5$ to rotate the gripper are converted to integer values $0, 1, \ldots, 120$.

Joint	Min Degree	Min Int. Value	Max Degree	Max Int. Value
$\boldsymbol{j}_1$ (shoulder pitch)	$-91°$	-19019	$+151°$	+31559
$\boldsymbol{j}_2$ (shoulder roll)	$-96°$	-20064	$+1°$	+209
$\boldsymbol{j}_3$ (shoulder torsion)	$-91°$	+19019	$+91°$	-19019
$\boldsymbol{j}_4$ (elbow)	$-115°$	+24035	$+1°$	-209
$\boldsymbol{j}_5$ (hand torsion)	$-90°$	0	$+90°$	120
$\boldsymbol{j}_6$ (fingers)	$-60°$	0	$+60°$	120

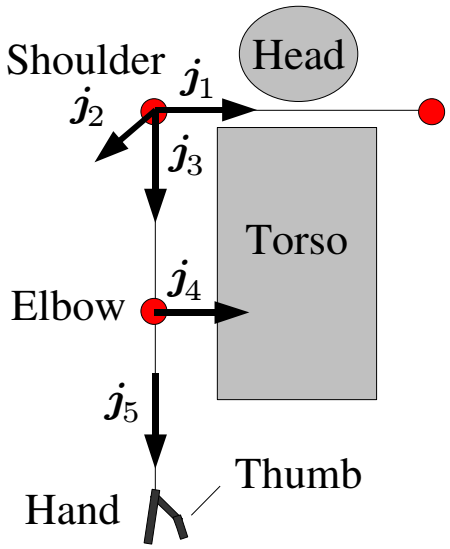


Figure 5.3: The Kinematic of HOAP's Right Arm. The figure shows the joints j_i of the right arm of the humanoid robot HOAP. The joints define a kinematic chain as indicated by the indices. The index numbers differ here from the robot's manual in order to have a concise notation for the right arm. The depicted pose is similar to the arm pose shown in Figure 5.2 (right), but the arm is hanging straight down. This pose is not the base pose of HOAP, where all angles α_i of the joints are zero. In the pose here, the angle α_1 of the shoulder joint j_1 (pitch) is set to $\alpha_1 = +90°$.

plates. The shoulders, head/neck, and the hip joints of the legs have 3 degrees of freedom each. The elbows and knees have only 1 degree of freedom. The ankles have 2 degrees of freedom. In addition to the hip joints, the robot has a joint between the legs and the torso, which allows the robot to lean forward. The wrists which take about 2/3 of the forearms have one degree of freedom each. By the wrist joint (called hand joint in the HOAP manual), the robot can change the orientation of the hand, see Figure 5.2. The hand has a gripper that consists of four fingers and a thumb which can be controlled only simultaneously. In Figure 5.2 (right), the gripper is closed as far as possible for the encompassed object.

Since the right arm of HOAP is used to imitate the arm movements, it is discussed in more depth. The kinematic chain of the right arm is shown in Figure 5.3, the so called finger joint j_6 for controlling the gripper is not shown in the figure. The ranges of the joint angles are listed in Table 5.1 together with the corresponding integer values which are used for the control of the robot. As one can see from the integer ranges, the shoulder joints j_1, j_2, j_3 as well as the elbow joint j_4 can be set very precisely. The wrist joint j_5 (hand torsion) and the the "joint" j_6 for controlling the gripper can only be set coarsely. The servo of the wrist is less accurate than the shoulder and elbow servos. It misses a given values slightly, which is also the case for the gripper. The articulation of the arm via the kinematic chain is discussed in the following subsection.

The control of the robot via a kernel module on the real-time Linux PC is performed as follows. After the initialization phase of the robot, a state vector can be transmitted to the kernel module. The state vector describes (basically) the full set of joint values of the robot. The joint values are specified

as integer values as given in Table 5.1 for the right arm. After the transmission of the state vector, the robot adapts rapidly to the given joint values. Beside, the access of the sensor readings, it is also possible to check whether a currently set target pose (specified by the state vector) has been reached. It is not required to wait until the robot has achieved the new target pose, instead it is possible to update the target pose with 1000Hz. This allows the robot to perform very smooth movements. The robot is able to damage itself when one sets an inappropriate target pose. The robot can also damage itself when the transition from the current body pose to a new target pose results in an inappropriate sequence of poses. Therefore, it is advisable to control the transition from one pose to another by updating the target pose frequently in order to define a smooth movement for the transition.

The methodology that is used in this thesis to control the robot for movement imitation is the following. The goal is to update the state vector for the robot with the full update frequence (1000 Hz). This is accomplished by using an additional process with a buffer for a large set of state vectors. This process takes every 1ms one state vector from the buffer (if available) and updates the target pose of the robot. The process that generates motion sequences can push the generated state vectors into the buffer. In order to generate smooth motions on the robot, the buffer must be prevented from running out of state vectors. In the following, the procedure for producing an arm action, as the relocation action (Section 5.2.5), is considered. The movement sequence consists of several sections. The generating process can start to generate a sequence of state vectors for the first section of the movement sequence. The sequence of state vectors is then pushed into the buffer. Since the additional process is responsible for updating the state vector for the robot, the generating process can start to work on generating the state vectors for the next section of the movement sequence. The procedure for generating the state vectors for a section of a movement sequence is basically the following. First, a sequence of arm poses is generated. The sequence has fewer samples than the final sequence of state vectors. Each arm pose is a vector of floating point numbers which define the values for the joints $\boldsymbol{j}_1, \ldots, \boldsymbol{j}_6$. This sequence of arm poses is then expanded to a longer sequence of complete state vectors. Therefore, the joint values are interpolated and converted into the required integer format for the robot. The additional components of the complete state vectors of the robot (which also includes the values for the left arm etc.) are filled with the values which correspond to the base pose of the arm actions.

The Kinematic Chain of the Robot Arm

The joints $\boldsymbol{j}_i$ of the right arm of the robot HOAP-3 (see Figure 5.3) have been discussed already in the section above. Here, the meaning of the kinematic chain of the robot's arm is discussed more closely. The articulation of the gripper as well as the body parts which do not belong to the right arm are not considered here. As a consequence, the state of the right arm is specified through the joint angles $\boldsymbol{\theta} \equiv (\alpha_1, \ldots, \alpha_5)$ of the joints $\boldsymbol{j}_1, \ldots, \boldsymbol{j}_5$. The default pose of the arm is assumed to be $\boldsymbol{\theta} = (0, 0, 0, 0, 0)$.

To understand the arm kinematics, it is worthwhile to consider to which location $\boldsymbol{q}$ a point $\boldsymbol{p}$ at the robot's hand is moved in a certain arm state. It is assumed in the following that the joints and the point $\boldsymbol{p}$ at the hand are defined in a reference frame for the default pose of the robot's arm. The joints are regarded as arbitrarily located and oriented rotation axes. A joint $\boldsymbol{j}_i$ can be defined as $\boldsymbol{j}_i = (\boldsymbol{q}_i, \boldsymbol{v}_i)$, where $\boldsymbol{q}_i$ is a point on the axis and $\boldsymbol{v}_i$ is a vector pointing in the direction of the oriented axis. Both the point and the vector are defined in the reference frame for the default pose of the robot's arm. This is indicated by using the notation $\boldsymbol{j}_i^{\mathbf{0}}$. Indeed, the location and orientation of the joints change if the arm

is articulated. Therefore, a joint axis for an arbitrary arm pose $\boldsymbol{\theta}$ is denoted by $\boldsymbol{j}_i^{\boldsymbol{\theta}}$.

If a point $\boldsymbol{p}$ is rotated by an angle α around an arbitrary located rotation axis $\boldsymbol{j} = (\boldsymbol{q}, \boldsymbol{v})$, then the point is at a location $\boldsymbol{p}'$ (in the same reference system). The transformation is denoted in the following as $\boldsymbol{p}' = \mathcal{R}_{\boldsymbol{j}}^{\alpha}(\boldsymbol{p})$ (see Section D.2.4). The location $\boldsymbol{q}$ of the point $\boldsymbol{p}$ at the robot hand (defined in the reference frame and the default pose $\boldsymbol{\theta} = \mathbf{0}$) is calculated for an arbitrary arm state $\boldsymbol{\theta} \equiv (\alpha_1, \ldots, \alpha_5)$ by following the kinematic chain: First, the location $\boldsymbol{p}' = \mathcal{R}_{\boldsymbol{j}_5^{\mathbf{0}}}^{\alpha_5}(\boldsymbol{p})$ is calculated. The point $\boldsymbol{p}'$ is the new location of the point at the hand in the reference frame, where only the wrist is rotated by α_5, which corresponds to an arm state $(0, 0, 0, 0, \alpha_5)$. Since $\boldsymbol{p}'$ is the location of the point for the already rotated hand, the location $\boldsymbol{p}''$ for the articulated elbow and rotated wrist can be calculated by $\boldsymbol{p}'' = \mathcal{R}_{\boldsymbol{j}_4^{\mathbf{0}}}^{\alpha_4}(\boldsymbol{p}')$. The point $\boldsymbol{p}''$ is then the location of the point at the hand for the arm state $(0, 0, 0, \alpha_4, \alpha_5)$. Following this argument, one can calculate the locations $\boldsymbol{p}''' = \mathcal{R}_{\boldsymbol{j}_4^{\mathbf{0}}}^{\alpha_3}(\boldsymbol{p}'')$ (for the arm state $(0, 0, \alpha_3, \alpha_4, \alpha_5)$) and $\boldsymbol{p}^{(4)} = \mathcal{R}_{\boldsymbol{j}_4^{\mathbf{0}}}^{\alpha_4}(\boldsymbol{p}''')$, and finally the location $\boldsymbol{q} = \boldsymbol{p}^{(5)} = \mathcal{R}_{\boldsymbol{j}_5^{\mathbf{0}}}^{\alpha_5}(\boldsymbol{p}''')$ of the point at the hand for the fully articulated arm state $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_5)$. The location $\boldsymbol{q}$ of the point at the hand can be written as a concatenation of functions:

$$\boldsymbol{q} = \left(\mathcal{R}_{\boldsymbol{j}_5^{\mathbf{0}}}^{\alpha_5} \circ \mathcal{R}_{\boldsymbol{j}_4^{\mathbf{0}}}^{\alpha_4} \circ \cdots \circ \mathcal{R}_{\boldsymbol{j}_1^{\mathbf{0}}}^{\alpha_1} \right) (\boldsymbol{p}),$$

which corresponds to the kinematic chain. The calculation of the points in the order $\boldsymbol{j}_5^{\mathbf{0}}, \ldots, \boldsymbol{j}_1^{\mathbf{0}}$ of the joints is important here in order to use simply the joint axis $\boldsymbol{j}_i^{\mathbf{0}}$ as defined in the default arm pose. If one starts with calculating first a location $\boldsymbol{p}'$ of the point $\boldsymbol{p}$ for the articulated shoulder joint $\boldsymbol{j}_1$ (corresponding to the state $\boldsymbol{\theta}_1 = (\alpha_1, 0, 0, 0, 0)$, then one would be required to calculate the new location and orientation of the joint $\boldsymbol{j}_2^{\boldsymbol{\theta}_1}$ in order to calculate the location $\boldsymbol{p}''$ for the state $\boldsymbol{\theta}_2 = (\alpha_1, \alpha_2, 0, 0, 0)$.

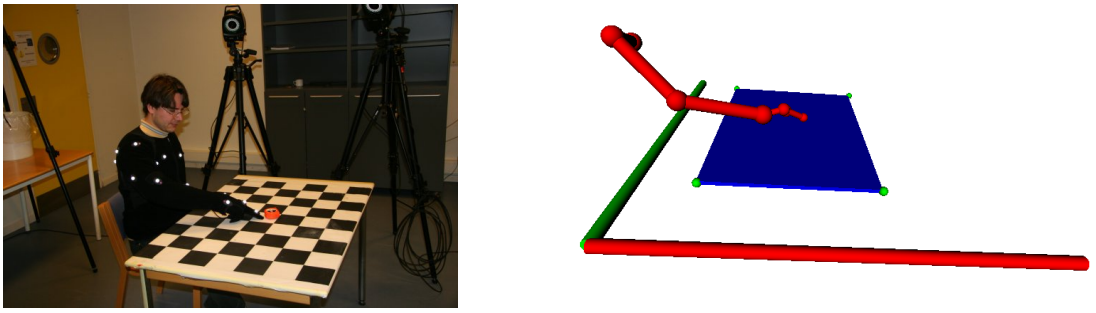


Figure 5.4: Reaching Action. ***Left:*** The figure shows the table-top scenario in which the arm actions are recorded. Here, a pointing action is shown. — ***Right:*** The virtual environment visualizes a recorded reaching action to a similar object location as shown in the picture on the left. The blue rectangle visualizes the active table-top region for which actions are recorded. The red and green axes show the x-axis and y-axis of the reference frame. The axes have a length of 1m and intersect in the origin, the z-axis is not shown but points upwards.

5.2.2 Modeling the Reaching Actions

In this section some properties of the recorded reaching actions and the representation through an interpolative PHMM are discussed. The interpolative PHMM (IPHMM) is parametrized by the 3D target location $\boldsymbol{t}$ which is specifying the location of an object. The 3D parametrization is necessary

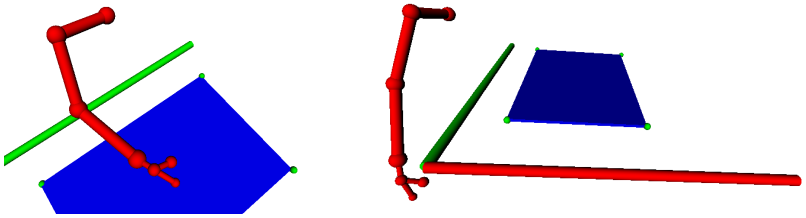


Figure 5.5: Recorded Body-Fixed Points and The Base Pose. ***Left:*** The figure shows the 7 body-fixed points which are recorded for the demonstrated arm actions, the locations are indicated through the balls between the sticks of the red arm. — ***Right:*** The figure shows the base pose. All reaching actions start and end in this base pose.

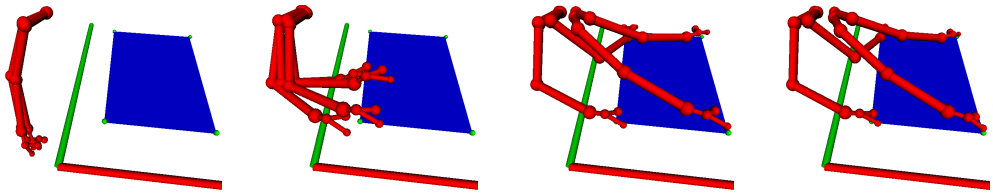


Figure 5.6: HMM States Representing Reaching Actions. The figures show the means of corresponding states of four of the local HMMs of an interpolative PHMM trained on reaching actions. Each local HMM represents reaching actions to a corner of the table-top region. The local HMMs have 80 states. The figures from left to right show the means of the states 1 (the base pose), 21 (the object is still approached), 30 (the object has been reached but not grasped) and 45 (the object is grasped and released after this state). The retreating part of the actions (until the base pose is finally reached in state 80) is not shown.

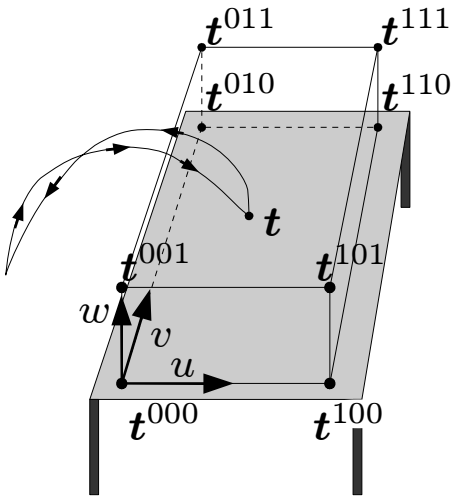


Figure 5.7: Target Points of the Actions. The figure shows the target points of the reaching actions represented through the 8 local HMMs of the IPHMM. In addition, a finger trajectory of an action with target point $\boldsymbol{t}$ is shown. The coordinates (u, v, w) are used in the IPHMM for the interpolation of the local HMMs.

to enable the robot to grasp an object at a certain height at an arbitrary table-top location. This is for example necessary to adjust the action to the height of the table-top. In addition, it is discussed why a synthesized action for a location $\boldsymbol{t}$ is always an action which reaches for an object at the location $\boldsymbol{t}$. This is not obvious since the interpolative model cannot model the systematic of a parametric movement with arbitrary precision. An important aspect of the reaching actions is the representation of the body pose, since finally the representation has to be converted into joint angles for the robot (Section 5.2.4). However, the settings for the model training are very similar to the settings in Section 4.2.

The reaching actions of the demonstrator are recorded in a table-top scenario shown in Figure 5.4. An arm pose where the object is grasped is shown in Figure 5.4 (right). Each reaching action consists of basically three parts: approaching the object, interacting with the object, and the withdrawing motion. Further details are given in Figure 5.6. The actions begins and ends in the same base pose, where the arm is hanging down beside the demonstrator. The base pose is shown in Figure 5.5 (right). The recorded movements are represented by the 3D locations of 7 body-fixed points, which are shown in Figure 5.5 (left). The locations are: the sternum, and the shoulder, elbow, thumb, index finger, and its knuckle. A vector describing a pose is a 21 dimensional vector $\boldsymbol{x}$ storing the seven 3D locations $\boldsymbol{x} = (\boldsymbol{p}_1^\top, \ldots, \boldsymbol{p}_7)$. Each 3D point $\boldsymbol{p}_i$ is represented in a global reference system, which is shown in Figure 5.4 (right).

As mentioned above, an interpolative PHMM $\boldsymbol{\lambda}^{\boldsymbol{t}}$ is used to represent the reaching actions. The parametrization of the IPHMM is a 3D target location $\boldsymbol{t} \in \mathbb{R}^3$. Each training action is a movement, where the demonstrator reaches for one of the objects that are placed at the target locations $\boldsymbol{t}^{ijk}$. The target points $\boldsymbol{t}^{ijk}$ are shown in Figure 5.7. Bases on the training movements, 8 local HMMs are setup, where each local HMM $\boldsymbol{\lambda}^{\boldsymbol{t}^{ijk}}$ models reaching actions for the target $\boldsymbol{t}^{ijk}$ $(i, j, k \in \{0, 1\})$. Figure 5.6 shows the poses that are represented by the state means of the four local HMMs $\boldsymbol{\lambda}^{\boldsymbol{t}^{ijk}}$ with $k = 0$. The training procedure for an IPHMM has been discussed in Section 3.4, and the settings for the training are very similar to those discussed in Section 4.2. However, here the trained IPHMM is an 80 state PHMM (which is trained with time restrictions similar to Section 4.2).

Internally, the interpolative PHMM uses a parametrization (u, v, w) with normalized interpolation coordinates $u, v, w \in [0, 1]$. However, the conversion of a target location $\boldsymbol{t}$ into the representation (u, v, w) is simple and discussed in the following. For brevity, it seems to be reasonable to use the notation $\boldsymbol{\lambda}^{\boldsymbol{t}}$. The interpolation coordinates are defined such that a vector $(u, v, w) = (i, j, k)$ with $i, j, k \in \{0, 1\}$ corresponds to the target point $\boldsymbol{t}^{ijk}$. It is worthwhile to note that the points $\boldsymbol{t}^{ijk}$ are the corners of an (undistorted) cuboid which simplifies the calculation of the u, v, w interpolation coordinates of a target point $\boldsymbol{t}$. The (u, v, w) coordinates for the point $\boldsymbol{t}$ are calculated by the formulas

$$\begin{aligned} u &= \frac{\boldsymbol{t} - \boldsymbol{t}^{000}}{|\boldsymbol{t} - \boldsymbol{t}^{000}|} \cdot \frac{\boldsymbol{t}^{100} - \boldsymbol{t}^{000}}{|\boldsymbol{t}^{100} - \boldsymbol{t}^{000}|}, \\ v &= \frac{\boldsymbol{t} - \boldsymbol{t}^{000}}{|\boldsymbol{t} - \boldsymbol{t}^{000}|} \cdot \frac{\boldsymbol{t}^{010} - \boldsymbol{t}^{000}}{|\boldsymbol{t}^{010} - \boldsymbol{t}^{000}|}, \\ w &= \frac{\boldsymbol{t} - \boldsymbol{t}^{000}}{|\boldsymbol{t} - \boldsymbol{t}^{000}|} \cdot \frac{\boldsymbol{t}^{001} - \boldsymbol{t}^{000}}{|\boldsymbol{t}^{001} - \boldsymbol{t}^{000}|}, \end{aligned} \tag{5.1}$$

where "$\cdot$" is the dot product. The correctness of the formulas above becomes apparent though the following considerations.

In the following, the interpolation formulas for generating an HMM for the interpolation coordinates (u, v, w) are considered. Since the synthesis is based only on the state means, the interpola-

tion formulas are given explicitly for the means. The interpolation formulas are applied on all states $i = 1, \ldots$ in the same way:

$$\boldsymbol{\mu}_i^{uvw} = \bar{w}\boldsymbol{\mu}_i^{uv0} + w\boldsymbol{\mu}_i^{uv1}, \tag{5.2}$$
$$\boldsymbol{\mu}_i^{uv0} = \bar{v}(\bar{u}\boldsymbol{\mu}_i^{000} + u\boldsymbol{\mu}_i^{100}) + v(\bar{u}\boldsymbol{\mu}_i^{010} + u\boldsymbol{\mu}_i^{110}), \tag{5.3}$$
$$\boldsymbol{\mu}_i^{uv1} = \bar{v}(\bar{u}\boldsymbol{\mu}_i^{001} + u\boldsymbol{\mu}_i^{101}) + v(\bar{u}\boldsymbol{\mu}_i^{011} + t\boldsymbol{\mu}_i^{111}), \tag{5.4}$$

where

$$\bar{w} = 1 - w, \quad \bar{v} = 1 - v, \quad \bar{u} = 1 - u,$$

The Equations (5.3) and (5.4) are a compact notation of the bi-linear interpolation formulas used in Section 3.4.3. Based on the Equations (5.3) and (5.4), the means $\boldsymbol{\mu}_i^{uv0}$ and $\boldsymbol{\mu}_i^{uv1}$ are calculated for the interpolation parameters $(u, v, 0)$ and $(u, v, 1)$. Equation (5.2) interpolates the mean $\boldsymbol{\mu}^{uvw}$ for the arbitrary interpolation parameters (u, v, w). The mean $\boldsymbol{\mu}_n^{ijk}$ with $i, j, k \in \{0, 1\}$ is the mean of the r-th state of the local HMM which represents the actions for a target location $\boldsymbol{t}^{ijk}$.

In the following, it is assumed that an arbitrary target point $\boldsymbol{t}$ is given. The interpolation parameters (u, v, w) are assumed to be given on the basis of the equations in (5.1). In the following the value $\boldsymbol{t}^{uvw}$ given by (5.2) with the corners $\boldsymbol{t}^{ijk}$ in place of the means $\boldsymbol{\mu}^{ijk}$ is calculated. It is shown that $\boldsymbol{t}^{uvw} = \boldsymbol{t}$. If one calculates first the value $\bar{u}\boldsymbol{t}^{0jk} + u\boldsymbol{t}^{1jk}$ for $j, k \in \{0, 1\}$, then one obtains

$$\begin{aligned}
\bar{u}\boldsymbol{t}^{0jk} + u\boldsymbol{t}^{1jk} &= (1-u)\boldsymbol{t}^{0jk} + u\boldsymbol{t}^{1jk} \\
&= \boldsymbol{t}^{0jk} + u(\boldsymbol{t}^{1jk} - \boldsymbol{t}^{0jk}) \\
&= \boldsymbol{t}^{0jk} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}),
\end{aligned}$$

since $\boldsymbol{t}^{1jk} - \boldsymbol{t}^{0jk} = \boldsymbol{t}^{100} - \boldsymbol{t}^{000}$. This is true, since the cuboid with the corners $\boldsymbol{t}^{kij}$ is not distorted (see Figure 5.7). If one plugs these results into the Equation (5.3) (the case $k = 0$) or Equation (5.4) (the case $k = 1$), then one gets

$$\begin{aligned}
\boldsymbol{t}^{uvk} &= \bar{v}(\boldsymbol{t}^{00k} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000})) + v(\boldsymbol{t}^{01k} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000})) \\
&= (1-v)(\boldsymbol{t}^{00k} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000})) + v(\boldsymbol{t}^{01k} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000})) \\
&= (1-v)\boldsymbol{t}^{00k} + v\boldsymbol{t}^{01k} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) \\
&= \boldsymbol{t}^{00k} + v(\boldsymbol{t}^{01k} - \boldsymbol{t}^{00k}) + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) \\
&= \boldsymbol{t}^{00k} + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}),
\end{aligned}$$

where in the last step $\boldsymbol{t}^{01k} - \boldsymbol{t}^{00k} = \boldsymbol{t}^{010} - \boldsymbol{t}^{000}$ (for $k \in \{0, 1\}$) is used, which is true for the corners of the undistorted cuboid. If one plugs these results into Equation (5.2), then one gets for $\boldsymbol{t}^{uvs}$:

$$\begin{aligned}
\boldsymbol{t}^{uvw} &= \bar{w}\boldsymbol{t}^{uv0} + w\boldsymbol{t}^{uv1} \\
&= (1-w)\boldsymbol{t}^{uv0} + w\boldsymbol{t}^{uv1} \\
&= (1-w)(\boldsymbol{t}^{000} + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000})) \\
&\quad + w(\boldsymbol{t}^{001} + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000})) \\
&= (1-w)\boldsymbol{t}^{000} + w\boldsymbol{t}^{001} + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) \\
&= \boldsymbol{t}^{000} + w(\boldsymbol{t}^{001} - \boldsymbol{t}^{000}) + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) \\
&= \boldsymbol{t}^{000} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + w(\boldsymbol{t}^{001} - \boldsymbol{t}^{000})
\end{aligned}$$

Finally, one has

$$\boldsymbol{t}^{uvw} = \boldsymbol{t}^{000} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + w(\boldsymbol{t}^{001} - \boldsymbol{t}^{000}). \tag{5.5}$$

If one would plug the definitions (5.1) of u, v, w into this equation, then one can see that:

$$\boldsymbol{t}^{uvw} = \boldsymbol{t}^{000} + u(\boldsymbol{t}^{100} - \boldsymbol{t}^{000}) + v(\boldsymbol{t}^{010} - \boldsymbol{t}^{000}) + w(\boldsymbol{t}^{001} - \boldsymbol{t}^{000}) = \boldsymbol{t}. \tag{5.6}$$

This shows that the choice of the formulas (5.1) is reasonable. However, this has an interesting consequence for the generation of a sequence of means by using the interpolation formula (5.2). The mean μ_n^{ijk} of a local HMM represents the arm poses though a stacked vector $(\boldsymbol{p}_1^\top, \ldots, \boldsymbol{p}_7^\top)$, which contains the 7 body-fixed 3D points. If one assumes now that the finger component $\boldsymbol{p}_6$ in the sequence of means of a local HMM λ^{ijk} reaches exactly the target point $\boldsymbol{t}^{ijk}$, and if one assumes further that this occurs for all the local HMMs in the same state n, then the interpolated arm pose μ_n^{uvw} (Equation (5.2)) for interpolation coordinates (u, v, w) of a location $\boldsymbol{t}$ represents again an arm pose, where the finger component $\boldsymbol{p}_6$ is exactly at the target location $\boldsymbol{t}$. Figure 5.6 shows that the object is grasped for each local HMM in the same state. (A reason therefore is the training with time restrictions.) Thus one can assume that a movement, which is generated through interpolating the local HMMs for a certain target $\boldsymbol{t}$, is a movement, where the finger and thumb very accurately reach the target $\boldsymbol{t}$.

5.2.3 Converting the Action Model

In this section, the conversion of the action model $\boldsymbol{\lambda}^{\boldsymbol{t}}$ (Section 5.2.2) to a model $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{t}}}$ is discussed. The model $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{t}}}$ can then be used to synthesize reaching actions for the robot, where each synthesized pose can be easily mapped to joint angles (Section 5.2.4) that are required to control the robot.

In Section 5.2.1, the embodiment of the robot was discussed. It differs from the demonstrator's embodiment in the size and in the degree of freedom of the body. The robot's wrist can, for example, only rotate the hand. The distance between the elbow and the location, where the robot can grasp a certain object with its end-effector, is fixed (see Figure 5.2 (left)). If the object is a cylinder with a diameter of about 2cm, then the distance between elbow and center of the cylinder is $b = 133\text{mm}$. This distance is addressed in the following as forearm length. The distance between the shoulder and elbow joints is $a = 110\text{mm}$. The robot is not able to move the shoulders forward and backward without moving the upper body. Obviously, this is possible for a human.

The basic idea of the conversion of the model $\boldsymbol{\lambda}^{\boldsymbol{t}}$ is to accommodate the restrictions of the robot but to preserve the correspondence between the target point for which an action is synthesized and the actually reached location. As only the means of the output distributions of the HMMs are required for the synthesis, only the mapping $\boldsymbol{x} \mapsto \hat{\boldsymbol{x}}$ of a pose vector $\boldsymbol{x} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_7)$ onto a new pose vector $\hat{\boldsymbol{x}}$ is considered. The mapping is applied on each mean $\boldsymbol{\mu}_m^{ijk}$ of the 8 local HMMs of the IPHMM $\boldsymbol{\lambda}^{\boldsymbol{t}}$ in the same way. For brevity, the notation $\boldsymbol{x}^n = (\boldsymbol{p}_1^n, \ldots, \boldsymbol{p}_7^n)$ is used to address each mean $\boldsymbol{\mu}_n^{ijk}$, i.e. $\{\boldsymbol{x}^n\}_{n=1}^{N'} = \{\boldsymbol{\mu}_l^{ijk}\}$, where $N' = N \cdot 8 = 80 \cdot 8$.

In a preprocessing step, several attributes are extracted from the old model $\boldsymbol{\lambda}^{\boldsymbol{t}}$. The average location $\bar{\boldsymbol{p}} = \sum_n \boldsymbol{p}_2^n / N'$ of the shoulder is calculated. The elbow location of a pose $\boldsymbol{x}^n$ is denoted in the following by $\tilde{\boldsymbol{q}}^n \equiv \boldsymbol{p}_3^n$. A point $\tilde{\boldsymbol{r}}^n = \sum_{i=5}^{7} \boldsymbol{p}_i^n / 3$ is calculated for each pose $\boldsymbol{x}^n$. The point $\tilde{\boldsymbol{r}}^n$ should correspond to the location of the object's center in the gripper of the robot. The point $\tilde{\boldsymbol{r}}^n$ is here the average of the locations of the thumb ($\boldsymbol{p}_7^n$), the index finger ($\boldsymbol{p}_6^n$) and the knuckle of the

finger ($\boldsymbol{p}_5^n$). In addition, a vector from index finger to thumb is calculated: $\tilde{\boldsymbol{o}}^n = \boldsymbol{p}_7^n - \boldsymbol{p}_6^n$. It is used to mimic the articulation of the demonstrator's hand. To accommodate the different sizes of the demonstrator and the robot, the maximal effective overall arm length of the demonstrator is calculated as $l' = \max_n(a'^n + b'^n)$, where $a'^n = \|\tilde{\boldsymbol{q}}^n - \bar{\boldsymbol{p}}\|$ is the upper arm length and $b'^n = \|\tilde{\boldsymbol{r}}^n - \tilde{\boldsymbol{q}}^n\|$ is the forearm length (distance from elbow to the location where an object is grasped). The value for the demonstrator is $l' = 679.1\text{mm}$, where $a' = \max_n a'^n = 277.8\text{mm}$ and $b' = \max_n b'^n = 408.2\text{mm}$. The values differ obviously from the overall arm length of the robot $l = a + b = 110\text{mm} + 133\text{mm} = 243\text{mm}$, and also the proportion of upper arm and forearm differ ($a'/b' \approx 0.68$ and $a/b \approx 0.83$). The values a' and l' are biased, since the fixed shoulder location $\bar{\boldsymbol{p}}$ has been used.

Finally, a pose vector $\boldsymbol{x}^n$ is converted into a pose vector $\hat{\boldsymbol{x}}^n = \hat{\boldsymbol{x}}^n(\boldsymbol{x}^n)$ that is

$$\hat{\boldsymbol{x}}^n(\boldsymbol{x}^n) = (\hat{\boldsymbol{p}}^n = (\bar{\boldsymbol{p}} - \bar{\boldsymbol{p}}) \cdot s = \boldsymbol{0},\ \hat{\boldsymbol{q}}^n = (\tilde{\boldsymbol{q}}^n - \bar{\boldsymbol{p}}) \cdot s,\ \hat{\boldsymbol{r}}^n = (\tilde{\boldsymbol{r}}^n - \bar{\boldsymbol{p}}) \cdot s,\ \hat{\boldsymbol{o}}^n = \tilde{\boldsymbol{o}}^n \cdot s), \tag{5.7}$$

where $s = l/l'$ is used to accommodate the smaller arm length of the robot. The shoulder, elbow and end-effector locations are rescaled and represented with respect to the shoulder (as a consequence, the fixed shoulder location is $\hat{\boldsymbol{p}}^n = \boldsymbol{0}$). This allows one to find for each pose $\hat{\boldsymbol{x}}^n$ an arm pose of the robot where the gripper is at its location $\hat{\boldsymbol{r}}^n$.

The covariances of the IPHMM $\boldsymbol{\lambda}^{\boldsymbol{t}}$ are not important for the synthesis of prototype movements and are neglected. After the conversion of the means of the local HMMs as given by Equation (5.7), one gets 8 local HMMs $\hat{\boldsymbol{\lambda}}^{\boldsymbol{t}^{(i,j,k)}}$, which can be used as IPHMM to generate movements for the robot. A final concern is the parametrization of the IPHMM. Due to the transformation Equation (5.7), the local HMM $\hat{\boldsymbol{\lambda}}^{\boldsymbol{t}^{(i,j,k)}}$ represents now a reaching action for a target location $\hat{\boldsymbol{t}}^{(i,j,k)} = (\boldsymbol{t}^{(i,j,k)} - \bar{\boldsymbol{p}}) \cdot s$ (not $\boldsymbol{t}^{(i,j,k)}$). However, for a target location $\hat{\boldsymbol{t}}$, the interpolation of the local HMMs with the interpolation parameters (u, v, w), which are calculated by Equation (5.1) by using the target points $\hat{\boldsymbol{t}}^{(i,j,k)}$ in place of $\boldsymbol{t}^{(i,j,k)}$, results in an action for the target $\hat{\boldsymbol{t}}$. The IPHMM based on the converted local HMMs is denoted as $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{t}}}$, which emphasizes that the points $\hat{\boldsymbol{t}}^{(i,j,k)}$ are used to calculate the interpolation parameters for a target point $\hat{\boldsymbol{t}}$.

In the following, the mapping of a pose $\hat{\boldsymbol{x}}$ to the joint angles of the robot is discussed. The mapping utilizes the maximal and minimal distances $o_{\min} = \min_n \|\hat{\boldsymbol{o}}^n\|$ and $o_{\min} = \max_n \|\hat{\boldsymbol{o}}^n\|$ between the (rescaled) finger and thumb.

5.2.4 Mapping an Arm Pose to Joint Angles

The conversion, discussed in the section above, results in a model $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{t}}}$. An arm pose which is generated from the adapted model is in the following denoted as $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{o})$. (See also Equation (5.7).) The points $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ define the proposed locations of the robot's shoulder, elbow, and end-effector, where the shoulder is the reference, i.e. $\boldsymbol{p} = \boldsymbol{0}$. The vector $\boldsymbol{o}$ is basically a vector that points from the "index finger tip" to the "thumb".

The conversion of the pose $\boldsymbol{x}$ into joint angles has two concerns: 1. The joint angles of the robot's shoulder and elbow need to be calculated based on the points $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$. 2. The orientation of the hand and the gripper need to be set on the basis of $\boldsymbol{o}$.

The angles of the four joints of the shoulder and the elbow of the right arm of the robot are calculated such that the robot's end-effector is at the location $\boldsymbol{r}$. This is necessary to preserve the effect of the arm actions. Under this constraint it is not possible to set the elbow of the robot at the location

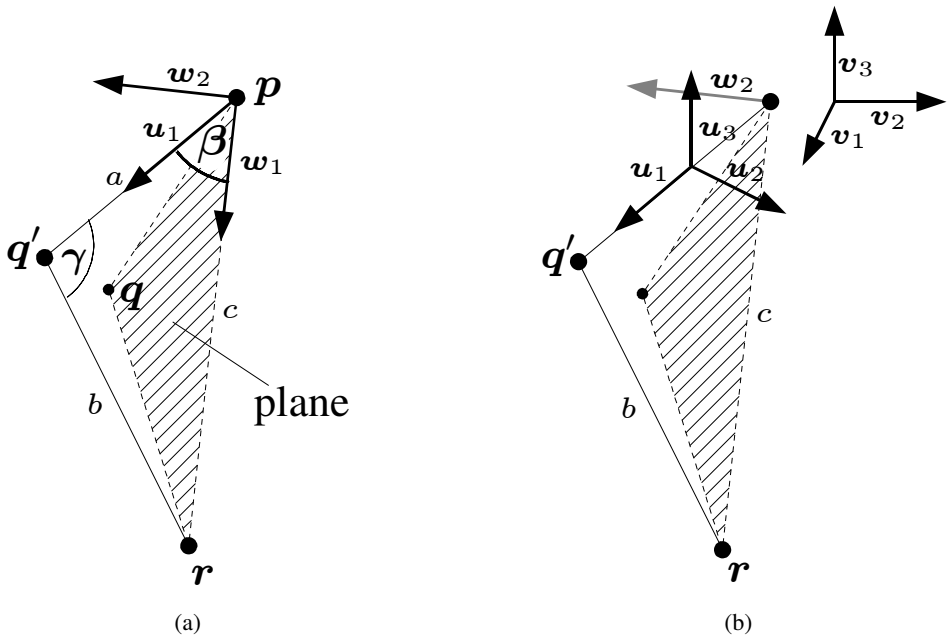


Figure 5.8: Mapping an Arm Pose for the Robot. In figure (a), the points $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$, which represent the shoulder, elbow, and end-effector locations, define a plane, on which the elbow $\boldsymbol{q}'$ of the robot should be. Figure (b) shows the two reference coordinate systems $(\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)$ and $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$. The system $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ is a fixed reference system of the robot's shoulder. The system $(\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)$ is a local reference system of the upper arm. The origin of both systems is supposed to be at the point $\boldsymbol{q}$. (The origins are translated only to improve the readability of the illustration.)

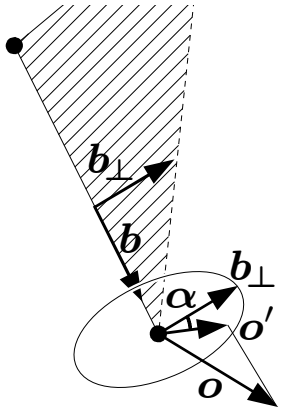


Figure 5.9: Rotation of the Robot's Wrist. The twist angle $\alpha = \measuredangle(\boldsymbol{b}_\perp, \boldsymbol{o}')$ of the wrist is calculated in the twist plane.

$\boldsymbol{q}$, since the arm proportions of the demonstrator and robot are different. A new elbow location $\boldsymbol{q}'$ has to be chosen. The constraints for the point $\boldsymbol{q}'$ are $\|\boldsymbol{q}' - \boldsymbol{p}\| = a$ and $\|\boldsymbol{q}' - \boldsymbol{r}\| = b$, where a and b are the lengths of the upper arm and the forearm of the robot. In order to preserve the appearance of the given arm pose, a valid location $\boldsymbol{q}'$ of the robot's elbow is chosen which is close to $\boldsymbol{q}$ and lies on the

plane defined by the three points $\boldsymbol{p}$, $\boldsymbol{q}$, $\boldsymbol{r}$. The plane and the points $\boldsymbol{p}$, $\boldsymbol{q}$, $\boldsymbol{r}$, and $\boldsymbol{q}'$ are illustrated in Figure 5.8 (a).

Note that the 3D coordinates of the points $\boldsymbol{p}$, $\boldsymbol{q}'$, $\boldsymbol{q}$, and $\boldsymbol{r}$ specify their locations in the reference system $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ of the shoulder. The reference system $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ is shown in Figure 5.8 (b)), where the origin of the system is supposed to be located at the location $\boldsymbol{p} = \boldsymbol{0}$ of the shoulder. Therefore, it makes sense to define $\boldsymbol{v}_1 = (1, 0, 0)$, $\boldsymbol{v}_2 = (0, 1, 0)$, and $\boldsymbol{v}_3 = (0, 0, 1)$, since the reference system used during the recordings is aligned in the same way. For the pose defined by the points $\boldsymbol{p}$, $\boldsymbol{q}'$ and $\boldsymbol{r}$ (see Figure 5.8 (a)), the angles of the three shoulder joints are defined through the rotation between the reference system $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ of the shoulder and the system $(\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)$ of the upper arm (see Figure 5.8 (b)). In order to compute the rotation, the vectors $\boldsymbol{u}_1, \boldsymbol{u}_2$ and $\boldsymbol{u}_3$ have to be determined in the reference system $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$.

First, the angles α and β (Figure 5.8 (a)) are determined. These angles are determined on the basis of Figure 5.8 (a), where a and b are (as mentioned above) the length of the upper arm and the forearm of the robot. The value c is $c = \|\boldsymbol{r} - \boldsymbol{q}\|$. By the law of cosines one achieves

$$\beta = \arccos\left(\frac{a^2 + c^2 - b^2}{2ac}\right), \tag{5.8}$$

$$\gamma = \arccos\left(\frac{a^2 + b^2 - c^2}{2ab}\right), \tag{5.9}$$

where γ defines the angle of the elbow joint (the correct value that has to be set in the state vector for robot control is $\gamma' = 180 - \gamma \cdot (180/\pi)$).

Now, $\boldsymbol{u}_1$ is given by

$$\boldsymbol{u}_1 = \cos\beta \cdot \boldsymbol{w}_1 \; + \; \sin\beta \cdot \boldsymbol{w}_2, \tag{5.10}$$

where

$$\boldsymbol{w}_1 = (\boldsymbol{r} - \boldsymbol{p})/\|\boldsymbol{r} - \boldsymbol{p}\|, \tag{5.11}$$

$$\boldsymbol{w}_2 = \boldsymbol{w}_2'/\|\boldsymbol{w}_2'\| \tag{5.12}$$

$$\boldsymbol{w}_2' = (\boldsymbol{q} - \boldsymbol{p}) - \langle \boldsymbol{q} - \boldsymbol{p}, \boldsymbol{w}_1 \rangle \, \boldsymbol{w}_1. \tag{5.13}$$

The vectors $\boldsymbol{w}_i$ are shown in Figure 5.8 (a). The vector $\boldsymbol{w}_2$ is orthogonal to $\boldsymbol{w}_1$ and lies in the plane defined by the points $\boldsymbol{p}$, $\boldsymbol{q}$ and $\boldsymbol{r}$.

Finally, the vectors $\boldsymbol{u}_2$ and $\boldsymbol{u}_3$ of the reference system of the upper arm (see Figure 5.8 (b)) can be calculated by

$$\boldsymbol{u}_2 = \boldsymbol{u}_2'/\|\boldsymbol{u}_2'\| \tag{5.14}$$

$$\boldsymbol{u}_2' = -\boldsymbol{w}_2 - \langle -\boldsymbol{w}_2, \boldsymbol{u}_1 \rangle \, \boldsymbol{u}_1 \tag{5.15}$$

$$\boldsymbol{u}_3 = \boldsymbol{u}_1 \times \boldsymbol{u}_2. \tag{5.16}$$

The Cardan angles[1] (ϕ, θ, ψ) of the shoulder joints are calculated on the basis of the rotation matrix

[1] Cardan angles ϕ, θ, ψ define a rotation $\boldsymbol{R}$ as $\boldsymbol{R} = \boldsymbol{R}_\psi^{\boldsymbol{z}} \boldsymbol{R}_\theta^{\boldsymbol{y}} \mathbf{R}_\phi^{\boldsymbol{x}}$.

$\boldsymbol{R}$ between the coordinate systems[2]:

$$\phi = \arctan2(r_{32}, r_{33}) \tag{5.17}$$
$$\theta = -\arcsin(r_{31}) \tag{5.18}$$
$$\psi = \arctan2(\mathrm{r}_{21}, \mathrm{r}_{11}) \tag{5.19}$$

where

$$\boldsymbol{R} = (r_{ij}) \;=\; [\boldsymbol{u}_1|\boldsymbol{u}_2|\boldsymbol{u}_3]^\top\,[\boldsymbol{v}_1|\boldsymbol{v}_2|\boldsymbol{v}_3]\,. \tag{5.20}$$

Finally, the orientation of the gripper of the robot need to be calculated from the direction vector $\boldsymbol{o}$ (finger→thumb). Since the wrist of the HOAP-3 has only one degree of freedom (twist), only the projection of the vector $\boldsymbol{o}$ onto the rotation plane of the wrist is used to calculate the twist angle α. This is illustrated in Figure 5.9. The angle α is

$$\alpha = \arccos\left(\frac{\langle \boldsymbol{b}_\perp, \boldsymbol{o}'\rangle}{|\boldsymbol{b}_\perp|\,|\boldsymbol{o}'|}\right) \cdot \mathrm{sgn}(\langle \boldsymbol{o}', \boldsymbol{b} \times \boldsymbol{b}_\perp\rangle), \tag{5.21}$$

where

$$\boldsymbol{o}' = \boldsymbol{o} - \frac{\langle \boldsymbol{o}, \boldsymbol{b}_\perp\rangle}{\|\boldsymbol{b}_\perp\|^2}\boldsymbol{b}_\perp, \tag{5.22}$$
$$\boldsymbol{b} = (\boldsymbol{r} - \boldsymbol{q})/\|\boldsymbol{q} - \boldsymbol{r}\| \tag{5.23}$$
$$\boldsymbol{b}_\perp = -\boldsymbol{u}_1 - \langle -\boldsymbol{u}_1, \boldsymbol{b}\rangle\boldsymbol{b}. \tag{5.24}$$

The normalized "joint" value $\xi \in [0, 1]$ for opening/closing the gripper is simply given by $\xi = \|\mathbf{o}\|/(o_{\max} - o_{\min})$. (The values $o_{\min}$ and $o_{\max}$ are defined in the section above.)

The conversion of the joint values into the integer representation that is used in the state vector for the robot control is trivial and can be deduced from Table 5.1.

5.2.5 Using The Reaching Actions to Relocate Objects

In the following the notation $\boldsymbol{\lambda}^{\boldsymbol{t}}$ is used for the adapted reaching model (formerly, the notation $\hat{\boldsymbol{\lambda}}^{\hat{\boldsymbol{t}}}$ was used in the Sections 5.2.2 and 5.2.3). By the use of the adapted reaching model $\boldsymbol{\lambda}^{\boldsymbol{t}}$, it is possible to generate a sequence of pose vectors $\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_{80}$ (where $\boldsymbol{x}_n = (\boldsymbol{p}_n, \boldsymbol{q}_n, \boldsymbol{r}_n, \boldsymbol{o}_n)$) for a reaching action for an arbitrary target $\boldsymbol{t}$, where the pose vectors are simply the means of the PHMM for the interpolation parameters (u, v, w) that are computed for $\boldsymbol{t}$.

The pose vectors can then be mapped (Section 5.2.4) to a sequence $\boldsymbol{\Theta} = \boldsymbol{\theta}_1 \cdots \boldsymbol{\theta}_{80}$ of state vectors, where a state vector $\boldsymbol{\theta}_n$ provides the joint values for the right arm of the robot. The sequences $\boldsymbol{\Theta}$ can then be interpolated and re-sampled to a sequence of adequate length T. If one wants that the sequence reproduced on the robot has the same duration d as the original sequences of the demonstrator, then one has to use $T = 1000 \cdot d$ samples. (The state vector is updated for the robot control with 1000 Hz.)

The reaching action has two practical concerns: 1. The demonstrator of the actions has grasped the object not only from the side but also slightly from the top. The imitated actions let the object tip

[2] Contrary to $\arctan(a/b)$, $\arctan2(a, b) \in (-\pi, \pi]$ respects the signs of a and b.

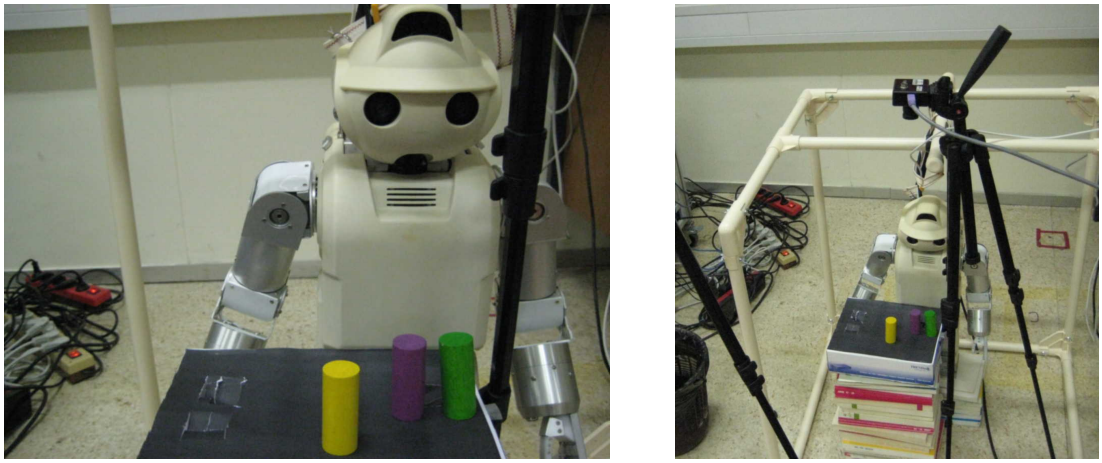


Figure 5.10: Table-Top Scenario. The scenario includes some colored objects which can be grasped by the robot and tracked by the camera (right). The black table-top has small holes (as can be seen on the left) into which the objects can be put.

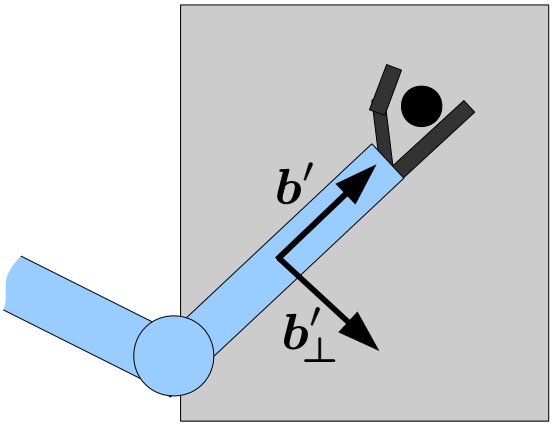


Figure 5.11: Gripper Approaching an Object.

over when the robot's thumb or fingers hit the object. 2. The reaching action is parameterized only by the target location of the object, without allowance for relocation of the object. Both problems are addressed below.

The following describes how to use the reaching action in order to perform a relocation action (which addresses the concerns 1. and 2.). However, this requires some further considerations.

The pose vector $\boldsymbol{x}_n$ and the state vector $\boldsymbol{\theta}_n$ (see above) can be computed for an arbitrary reaching target $\boldsymbol{t}$. As a consequence, the pose vector and state vector can be regarded as a function of $\boldsymbol{t}$. This is indicated in the following through the notation $\boldsymbol{x}_n^{\boldsymbol{t}}$ and $\boldsymbol{\theta}_n^{\boldsymbol{t}}$.

The following approach to the relocation action relies on the fact that the corresponding states of the local HMMs represent the same state of progress in the movement, see Figure 5.6. For the state number $T = 37$, each mean $\boldsymbol{\mu}_{n=T}^{ijk}$ represents a pose, where the gripper has reached the target location $\boldsymbol{t}^{ijk}$ but the gripper is still open. In the following states $n = T+1, T+2, \ldots$ the object is grasped and finally released in state $T' = 45$. The pose and state vectors $\boldsymbol{x}_n^{\boldsymbol{t}}$ and $\boldsymbol{\theta}_n^{\boldsymbol{t}}$ have the same semantic.

The relocation action (described below) is assumed to takes place in a table-top scenario as shown

in Figure 5.10, where the objects are placed on the table. A performance of the relocation action by the robot can be seen in Figure 5.12. The relocation action consists of three parts: 1. The approaching of an object at a location $\boldsymbol{t}_1$. 2. Grasping the object and placing it at another target location $\boldsymbol{t}_2$. 3. The object is released and the arm goes back to the base pose.

The first part of the relocating action: 1. *reaching for the object*

- First, a target location $\boldsymbol{t}'_1$ is determined from which the object can be approached by the gripper without tipping over the object. Therefore, the arm pose $\boldsymbol{x}_T^{\boldsymbol{t}_1} = (\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{o})$ is calculated, which provides the arm pose when the gripper would grasp the object at the location $\boldsymbol{t}_1$. This pose is illustrated in Figure 5.11. The target location $\boldsymbol{t}'_1$ is now chosen such that the gripper is placed before the object (and slightly to the side in the direction of the vector $-\boldsymbol{b}_\perp$, see Figure 5.11). The location is defined as $\boldsymbol{t}'_1 = \boldsymbol{t} - 0.03 \cdot \boldsymbol{b}' + 0.01 \cdot \boldsymbol{b}'_\perp$ (3cm before the object, 1cm to the right). The vector $\boldsymbol{b}' = (b_1, b_2, 0)/\sqrt{{b_1}^2 + {b_2}^2}$ is the projection of the vector $\boldsymbol{b} = \boldsymbol{r} - \boldsymbol{q}$ onto the plane of the table-top. (The points $\boldsymbol{q}$ and $\boldsymbol{r}$ are the locations of the elbow and the gripper.) The vector $\boldsymbol{b}'_\perp$ can be calculated by the cross product of $\boldsymbol{b}$ with the normal of the table-top plane.

  The sequence $\boldsymbol{\theta}_1^{\boldsymbol{t}'} \cdots \boldsymbol{\theta}_T^{\boldsymbol{t}'}$ is reproduced on the robot as a motion with a duration of about 3 seconds.
- For the state vectors which are generated in the following, the gripper is aligned to the plane of the table-top. This is achieved as follows: a state vector $\boldsymbol{\theta}$ is always generated based on a pose vector $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{o})$. The vector $\boldsymbol{o}$ is replaced in the vector $\boldsymbol{x}$ by a vector $\boldsymbol{o}'$ which is aligned to the table-top plane. (The vector $\boldsymbol{o}$ is used to calculate the rotation angle of the robot's wrist.) This alignment is done until the interaction with the object is over.
- Finally, the object is approached by using a sequence of states $\boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_0} \boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_2} \cdots \boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_K}$, where $\bar{\boldsymbol{t}}_i = \boldsymbol{t}'_1 + \alpha_i(\boldsymbol{t}_1 - \boldsymbol{t}'_1)$ with $\alpha_i = i/K$. The state vectors are generated from the same means μ_T^{ijk} of the model. The means represents a pose where the gripper has reached the target. Over the sequence $\boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_0} \boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_2} \cdots \boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_K}$ only the parameter $\bar{\boldsymbol{t}}$ varies. In the last state $\boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_K}$ of the sequence the gripper is at the target location $\bar{\boldsymbol{t}}_K = \boldsymbol{t}_1$. (However, it turned out that the object is grasped better if the target location $\boldsymbol{t}_1$ is replaced by a target location $\tilde{\boldsymbol{t}}_1 = \boldsymbol{t}_1 + 0.01 \cdot \boldsymbol{b}'_\perp$, where the gripper is placed slightly to the right of the object ($\approx 1\mathrm{cm}$).)
- During the entire sequence $\boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_0} \boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_2} \cdots \boldsymbol{\theta}_T^{\bar{\boldsymbol{t}}_K}$ the gripper is kept open.

The second part of the relocating action concerns the *relocation of the object*. The gripper is closed around the object and kept closed until the object is released. The object is then lifted 1.5cm. Then a location is approached where the gripper is 1.5cm above the target location $\boldsymbol{t}_2$ of the grasped object. Finally the gripper is moved to the target location $\boldsymbol{t}_2$. The generation of these sequence parts is performed by using $\boldsymbol{\theta}_T^{\boldsymbol{t}}$, while the parameter $\boldsymbol{t}$ is varied, so that $\boldsymbol{t}$ describes a curve (lifting, moving, and putting the object down).

The final part, *retreating from the object*, is very similar to the first part in a reverse order. First the gripper is opened. Then, the gripper is moved (and kept open) 0.6cm to the right of the object (therefore the direction $\boldsymbol{b}'_\perp$ is calculated for the new location $\boldsymbol{t} = \boldsymbol{t}_2$). Then, the gripper is moved to the location $\boldsymbol{t}'_2$ (this location is computed for $\boldsymbol{t}_2$ in the same way as $\boldsymbol{t}'_1$ is computed for $\boldsymbol{t}_1$). Finally, the sequence $\boldsymbol{\theta}_T^{\boldsymbol{t}'_2} \boldsymbol{\theta}_{T'+1}^{\boldsymbol{t}'_2} \cdots \boldsymbol{\theta}_{80}^{\boldsymbol{t}'_2}$ is used to generate the movement which brings the arm back to the base pose. The control of the gripper and its orientation is here extracted again from the PHMM.

As a final remark, the movement part until the gripper has approached the location $\boldsymbol{t}'_1$ and the part when the gripper leaves the location $\boldsymbol{t}'_2$ are a pure imitation of the reaching action.

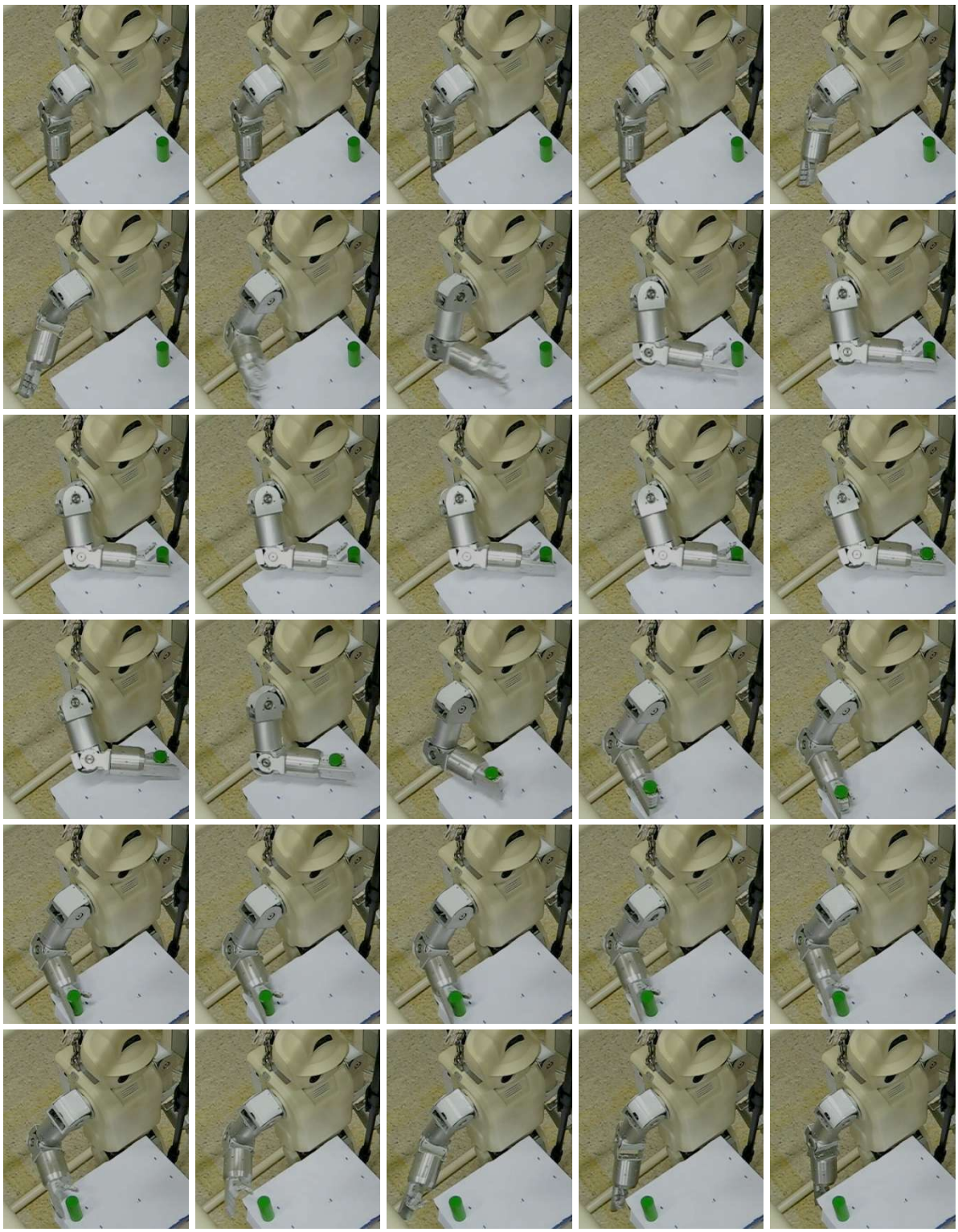


Figure 5.12: Object Relocation Action. The robot HOAP-3 relocates an object. The rows 1–2 show the reaching action to a point close to the object. In row 3, the object is approached and finally grasped. Row 4 shows the object relocation. The entire arm retreating sequence is shown in row 5–6.

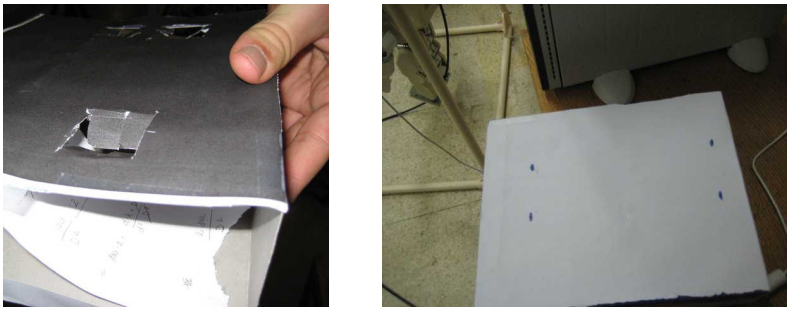


Figure 5.13: Tool Box. The pictures show the tool box which is used for the rule learning application. On one side the box has openings (left). On the other side the box has no openings (right).

5.2.6 A Rule-Learning Application

The effectiveness of the imitation-based approach to relocate objects (discussed in the section above) has been tested by implementing a task for the robot involving a number of objects. The basic idea of the task is a children's tool box with some objects. The box has different openings with different shapes, and an object can be pushed through an opening only if the shape of the object matches the shape of the opening. The setup for robot consists of a tool box (Figure 5.13) with four openings and 3 identically shaped but differently colored cylinders (Figure 5.10). Robot's task is to learn which object belongs to which opening. This is demonstrated to the robot by an instructor. After the rules are learned, the task is to put each object into the right opening. The objects are placed at arbitrary tool-box/table-top locations, and the instructor indicates which object is to be removed next.

A test run of the implementation is shown in Figure 5.12. The objects and hand of the person are tracked by a color tracker in a camera view, see Figure 5.10 (right). The steps of the implementation are discussed in the following more closely:

1. The PHMM for the reaching actions are learned from the demonstrator, the model is converted into a model λ^t which is used to synthesis reaching or relocation actions on the robot. An object location is specified by a value t of the parametrization of the HMM. The object target t is defined in the reference frame of the robot's shoulder. The height at which objects are grasped is adjusted to the current height of the working area of the table-top (see Figure 5.10 (right)).
2. *Camera Calibration.* The robot has to know the mapping between a tracked object location in the camera view and the object location in the robot's reference frame. The robot places one object at four different target locations (the corners of the working area) that are defined in the robot's reference frame (the height of the target 3D location of an object is always the height determined in step 1). The object is grasped at a certain location in the robot's reference frame with the help of the instructor and is placed at one corner of the working area, see Figure 5.14 (row 1). For the other three corners, the relocation action (as discussed in the section above) is used. The four object locations (given by the color tracker) at the corners are used to determine the mapping between object locations given by the tracker and the reference frame of the robot. The mapping is a homography between two planes, which can be estimated based on four point correspondences.

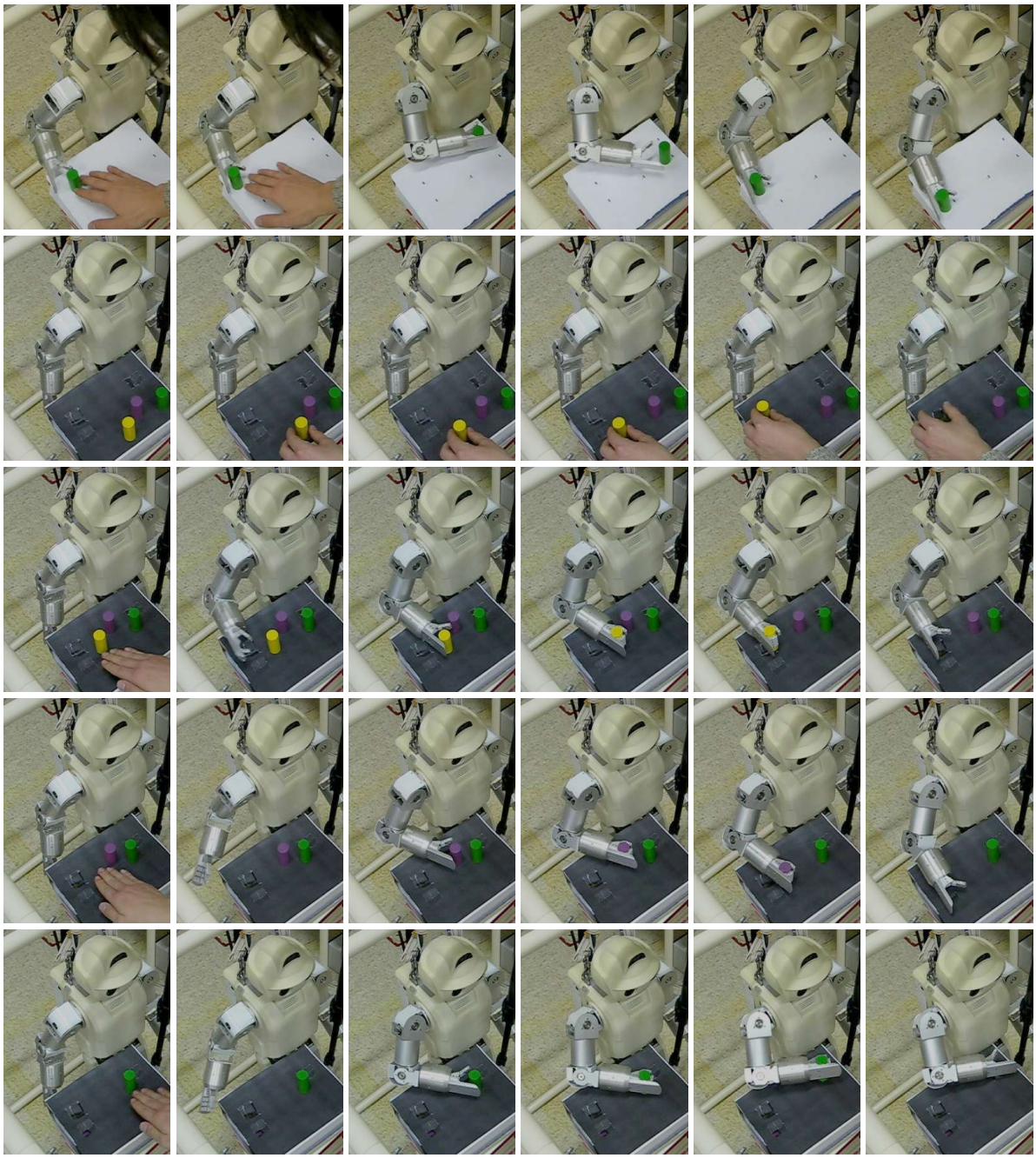


Figure 5.14: Rule-Learning Application. In this rule-learning application the humanoid HOAP-3 is shown some rules about where to put some objects (colored cylinders). First (see row 1), an object is given in the robots hand, and then the robot places the object at 4 designated target points in order to understand the mapping between the working area (a table-top with some openings) and the camera system (see Figure 5.10). Then it is demonstrated to the robot the openings into which the objects have to be placed. In row 2, it shown to the robot into which hole the yellow object belongs. After the demonstration of these rules for the three object (not shown), the robot is able to place the objects (after pointing to the object) into the right openings. This is shown in the rows 3–5 for each object.

3. *Rule Learning.* First, the tool-box is rotated so that the black side of the box with the openings is on the top. This does not change the height of the table-top. As a consequence, the mapping between the object location in the camera view and the robots reference frame is not affected. Then, the instructor shows for each object (color) the corresponding target opening (the target location). Each object is taken and placed in the opening, see Figure 5.14 (row 2). The robot identifies the color of an object when it starts to move and remembers the object location shortly before the object disappears.
4. *Applying the Rules.* First, the objects are distributed at the table-top at arbitrary locations. When the hand of the instructor points to an object the robot puts the object in the corresponding opening. The pointing is here identified by thresholding the distance between the color-tracked hand and the closest (tracked) object. If this threshold is maintained for some time, the robot removes the object and puts it into the opening learned for the color of the object. Therefore, again the relocation action is used. This is shown in Figure 5.14 (row 3–5) for the 3 objects.

The objects were accurately placed (as in the test run in Figure 5.14) into the correct openings. This shows that the robot can accurately reach for an object and place it. The inaccuracy in placing an object at a certain location would also result in an inaccurate camera calibration.

5.2.7 Discussion

The rule-learning application indicates that the parameter $\boldsymbol{t}$ of the reaching action very accurately defines the target location of the gripper (or more precisely the tool kit point of the gripper).

The accuracy is especially important in the rule-learning application. First, the relocation actions are used to place an object at 4 corners in order to determine the calibration. Secondly, the synthesis of the relocation action is performed for learned rules. The learning of the rules relies in return on the calibration. Here it is worthwhile to note that the relocation actions are generated for the object locations without any further adjustments during the execution on the robot (e.g., by tracking the gripper and adjusting the target location such that the target is reached more precisely). In addition, the rule-learning application indicates that the color tracker and the robot control are operating so precisely such that the application could succeed.

To some extent, the high accuracy of the reached at targets and the target locations where the objects are placed is not too surprising due to the use of the interpolative PHMM. The argument is therefore: Since the tool center point of the gripper is part of the pose vector that is represented by the state means of the PHMM, the tool center point will exactly reach the location $\boldsymbol{t}$ in a sequence that is generated from the IPHMM with the parameter $\boldsymbol{t}$. This has been discussed for the interpolative PHMM in Section 5.2.2. Of course, the local HMMs are trained on demonstrations and suffer from slight inaccuracies. In addition, the tool center point of the gripper is calculated simply as the average of the finger-tip, its knuckle and the thumb of the imitated human hand.

Another aspect about the accuracy of reaching at objects and placing them somewhere else is the appearance of the imitated movements. The appearance is important for the humanoid robots, since the movements of a humanoid are also supposed to be observed or interpreted by a human (or even another humanoid). This would be especially important if the movement conveys a certain meaning rather than serving the purpose of manipulating the environment. As one can see in Figure 5.12, the reaching and retreating part of the actions (based on imitating the human action) appear to be performed by human. However, this cannot be quantified easily.

In the following, the precision of the generated reaching actions is discussed, wherein the inaccuracy of the robot in achieving a certain pose configuration is neglected. The movements are encoded in the approach to imitation through the pose vector $\boldsymbol{x}$ over time, where a certain pose vector $\boldsymbol{x}$ includes the 3D elbow location $\boldsymbol{q}$ and the 3D location of the tool center point $\boldsymbol{r}$ of the gripper. In the mapping (discussed in Section 5.2.4), the elbow location $\boldsymbol{q}$ is corrected to a valid position $\boldsymbol{q}'$ so that the configuration of the robot's arm can be calculated. The elbow and gripper of the robot are then exactly set at the locations $\boldsymbol{q}'$ and $\boldsymbol{r}$. Here are two things to discuss. First, the interpolative PHMM generates the location $\boldsymbol{q}$ and $\boldsymbol{r}$ by interpolation. This results in an error in $\boldsymbol{q}$ and $\boldsymbol{r}$ that affects also the refined elbow location $\boldsymbol{q}'$. Second, the mapping sets the elbow of the robot at the location $\boldsymbol{q}'$, which is necessary due to the different proportions of the demonstrator's and robot's embodiments. However, even though necessary, this results in a pose which differs from the demonstrator's pose. The following analysis of the imitation error is done by using some local HMMs which are trained for the specific reaching targets for which the error is investigated. These local HMMs are trained and transformed (including the rescaling to the robot's size) in the same way as the local HMMs of the interpolative PHMM. The local HMMs are then used as a reference. All errors are calculated in the following in the scale of the robot.

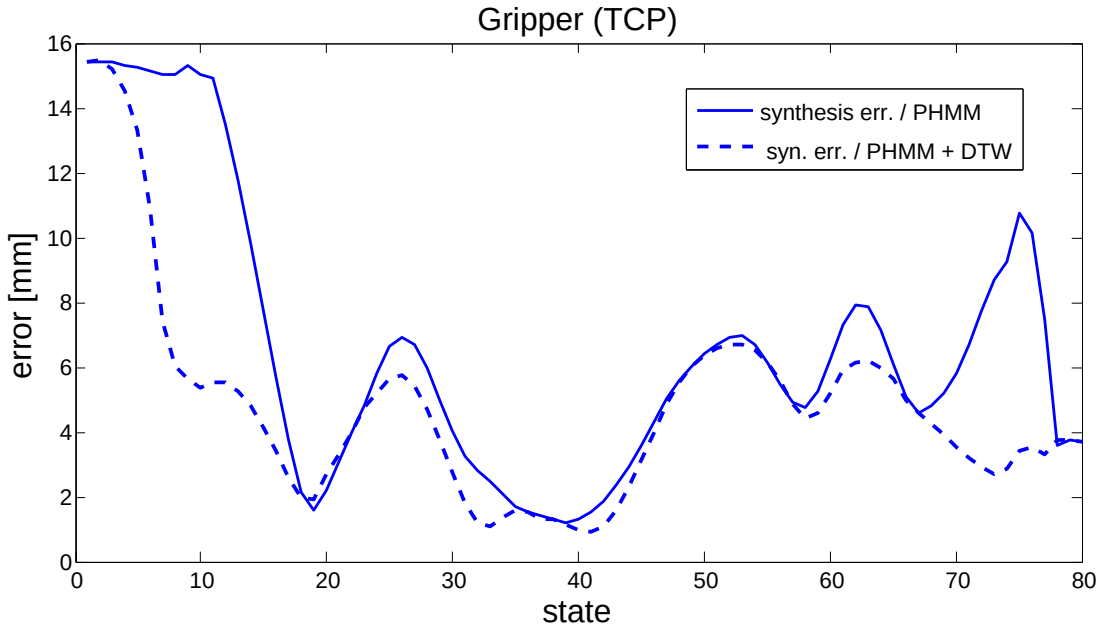


Figure 5.15: Synthesis Error of Gripper for Table-Top's Center. The plot shows the synthesis error of the gripper as in Figure 5.16 but for a target location $\boldsymbol{t}_1$ with $(u, v, w) = (0.5, 0.5, 0)$ at the center of the table-top. The sequence of gripper locations synthesized by the PHMM is directly used for the imitation. The dotted lines shows the same errors, where the sequences are aligned by DTW before calculating the error.

If one considers, for example, a reaching target $\boldsymbol{t}_1$ with interpolation parameters $(u, v, w) = (0.5, 0.5, 0)$ which correspond to the center of the work space, then one gets from each interpolated mean of the interpolative PHMM the locations $\boldsymbol{q}$ and $\boldsymbol{r}$. This results in sequences $\boldsymbol{q}_1 \cdots \boldsymbol{q}_{80}$ and

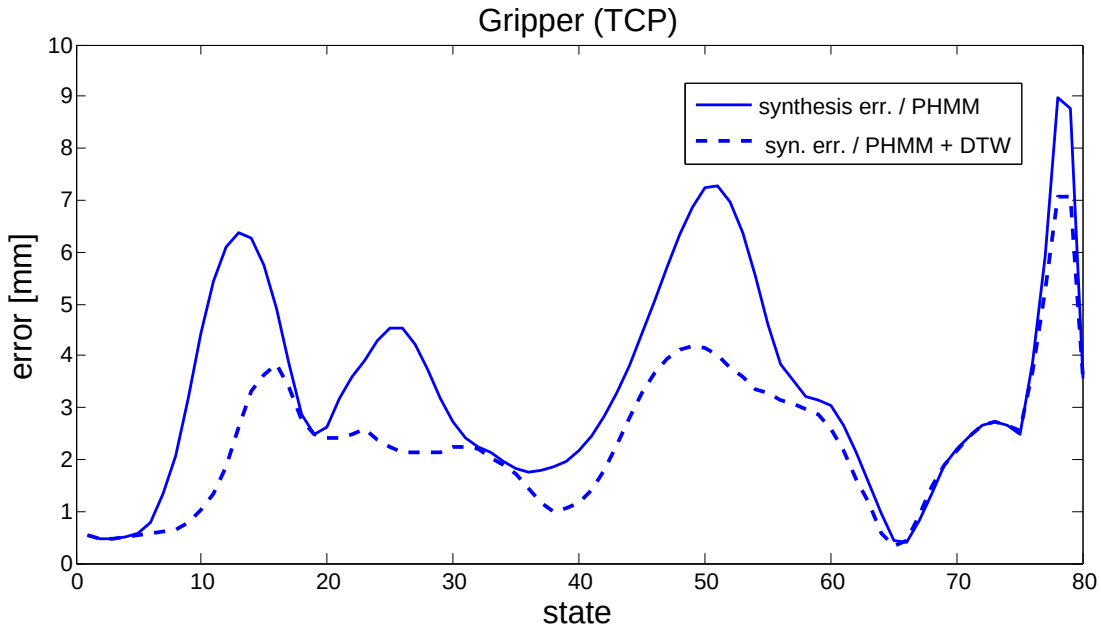


Figure 5.16: Synthesis Error of Gripper for Table-Top's Right Side. The plot shows the synthesis error of the gripper for a target location $\boldsymbol{t}_1$ with $(u, v, w) = (0.5, 1.0, 0)$.

$\boldsymbol{r}_1 \cdots \boldsymbol{r}_{80}$ of shoulder and gripper locations, where 80 is the number of states of each local HMM. In addition, the sequence $\boldsymbol{q}'_1 \cdots \boldsymbol{q}'_{80}$ of the refined elbow locations can be calculated. The corresponding sequences given by the reference HMM trained for the target $\boldsymbol{t}_1$ are denoted by $\hat{\boldsymbol{q}}_1 \cdots \hat{\boldsymbol{q}}_{80}$, $\hat{\boldsymbol{q}}'_1 \cdots \hat{\boldsymbol{q}}'_{80}$, and $\hat{\boldsymbol{r}}_1 \cdots \hat{\boldsymbol{r}}_{80}$. This allows one to calculate the synthesis error of the gripper location for the interpolative PHMM as $\|\boldsymbol{r}_i - \hat{\boldsymbol{r}}_i\|$ for the states $i = 1, \ldots, 80$. This error is shown by the solid line in Figure 5.15. Since the sequences represent basically trajectories in time, the calculated error can be a result of different progression rates, which results in a spacial misalignment of the sequences. Therefore, the sequences are aligned through dynamic time warping with a step pattern that is given in Equation (B.8) (see Section B.2). The error for the aligned sequences is shown in Figure 5.15 by the dashed line. Since the base pose of the arm has not been normalized for the local HMMs, the base poses differ slightly from each other. This explains the large error for the first 15 to 20 states and the last 15 to 20 states (see Figure 5.15 and Figure 5.16). In Figure 5.16, the PHMM synthesis error of the gripper is shown for a reaching target $\boldsymbol{t}_2$ with $(u, v, w) = (0.5, 1.0, 0)$, which is on the right of the working area. As expected, the error becomes very small (less than 2mm), when the gripper is very close to the reached at target location in the state 37 (see also Figure 5.15). This effect of the IPHMM has been discussed above. The error of the gripper location $\boldsymbol{r}$ for both reaching targets $(0.5, 0.5, 0)$ and $(0.5, 1.0, 0)$ is less than 8mm for the significant states 20–60, and less than 7mm for the warped trajectories.

Since the elbow location $\boldsymbol{q}$ is refined to a location $\boldsymbol{q}'$, one can look at different errors. The synthesis error $\|\boldsymbol{q}_n - \hat{\boldsymbol{q}}_n\|$ between the unrefined locations of the elbow as given by the PHMM and the reference HMM is the error due to the interpolation that is used in the IPHMM framework. This error is shown by the blue line in Figures 5.17 and 5.18 for the target locations $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ at the center and the right side of the working area. One can see that this error (up to 25mm) cannot be neglected for the target

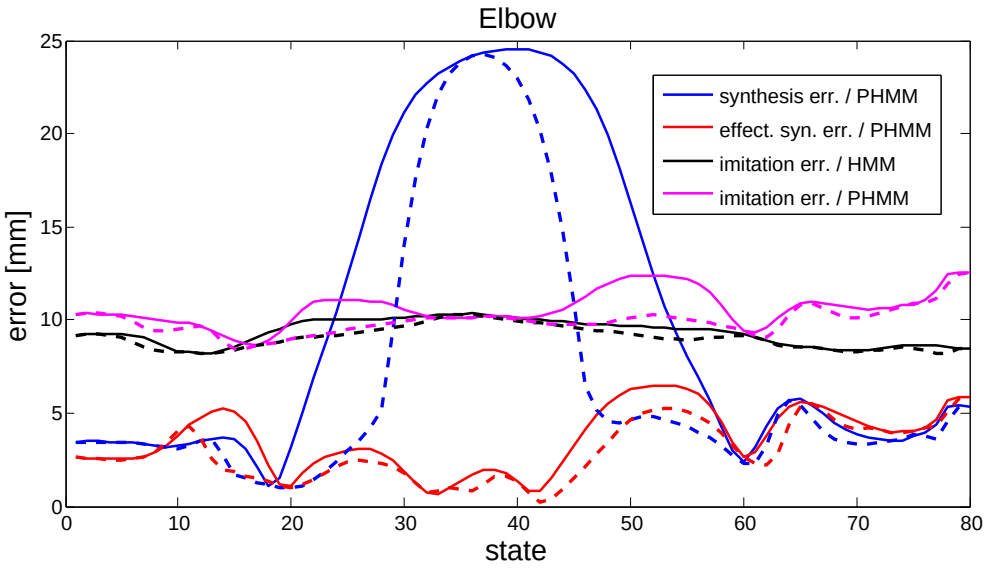


Figure 5.17: Imitation Error of Elbow for Table-Top's Center. The plot shows the synthesis and imitation errors of the elbow for the target location $\boldsymbol{t}_1$ at the center of the table-top. The effective synthesis error (red) shows how much the synthesis error (blue) affects the elbow location $\boldsymbol{q}'$ that is finally used during imitation. The imitation errors are the distance between the used elbow location $\boldsymbol{q}'$ and the true (rescaled) elbow location $\hat{\boldsymbol{q}}$ of the demonstrator. The dotted lines show the same type of errors, where the sequences are aligned by DTW before the errors are calculated.

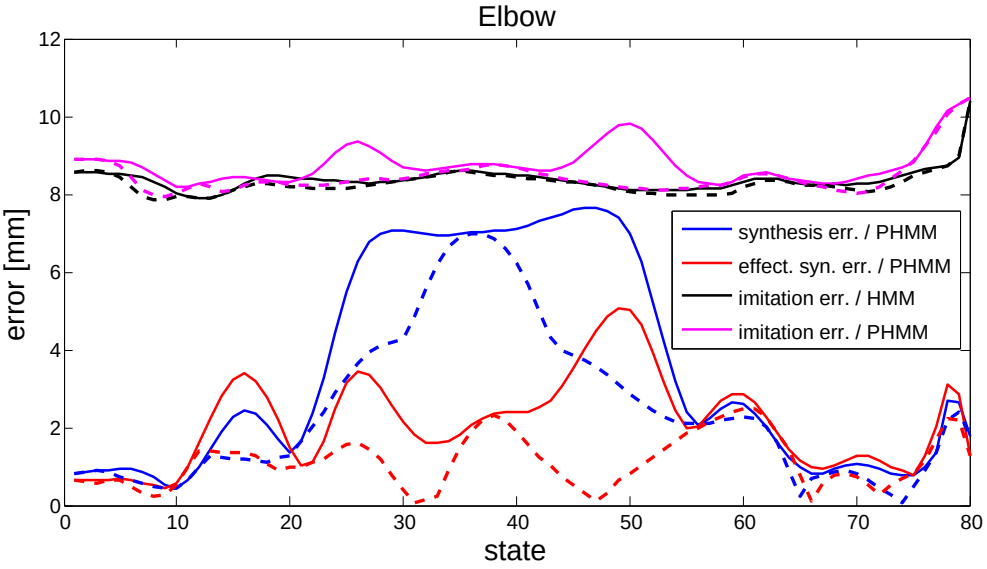


Figure 5.18: Imitation Error of Elbow for Table-Top's Right Side. The plot shows the same type of errors as Figure 5.17 but for the target $\boldsymbol{t}_2$ at the right of the table-top.

location $\boldsymbol{t}_1$, which is far away from the corners of the working area for which the local HMMs of the IPHMM are learned. For the target point $\boldsymbol{t}_2$ at the right side of the working area the error is up to 8mm. Interestingly, the effective error induced by the PHMM for the actually used elbow location is much smaller. This error is calculated as $\|\boldsymbol{q}'_n - \hat{\boldsymbol{q}}'_n\|$ and is shown in the plots (Figures 5.17 and 5.18) by the red line. The effective error of the elbow $\boldsymbol{q}'$ for both targets, $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, is less than 6mm. It is less than 5mm for the time warped sequences (for the states 20–60).

Finally, the distance between the used elbow location $\boldsymbol{p}'$ and the location $\hat{\boldsymbol{p}}$ given by the demonstrator might be used as a measurement for the quality of the imitation. This distance is called in the following the imitation error. The imitation error $\|\hat{\boldsymbol{p}}' - \hat{\boldsymbol{p}}\|$ of an imitated action from a local HMM shows the best one could expect for the chosen mapping to the robot's embodiment. This imitation error is shown in the plots (Figures 5.17 and 5.18) as black lines. The imitation error $\|\boldsymbol{p}' - \hat{\boldsymbol{p}}\|$ for the PHMM is shown in magenta. The imitation error for the local reference HMM is about 10mm for the location $\boldsymbol{t}_1$ and about 8.5mm for the location $\boldsymbol{t}_2$. The imitation errors for the PHMM are not greater than 12.5mm and 10mm for the locations $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, respectively. Interestingly, when time warping is used, the imitated elbow location for the PHMM synthesis is as good as for the local reference HMM (for the states 20–60).

As a summary, the locations of the gripper and the elbow are preserved during the IPHMM-based imitation of an action with an accuracy of 1cm and the gripper reaches for the target location with a very high accuracy (2mm). The inaccuracy of 1cm is necessary to preserve the effect on the embodiment of the robot which differs from the demonstrator's embodiment. A more precise PHMM would not improve the imitation significantly.

5.3 Summary

In this chapter the imitation of parametric movements with a certain meaning/effect (defined by the parametrization) has been discussed in general and for a specific application.

In the specific application of the reaching actions, it has been shown how the parametric movement model can be used to create a more sophisticated relocation action. This relies on the fact that the pose vector is a function of the time and the parameters of the object location. This allows one to generate a certain pose (where the time is considered as constant) for arbitrary locations. The pose can then be varied by modifying the parameters of the location. Each pose is then still a pose that is basically provided by the demonstrations.

In the discussion concerning the mapping of a parametric movement, several concerns have been considered. One is that the mapping of the movement onto another embodiment may affect the meaning of the parameterization. If the parameterization defines the effect as an object location to be reached for, then the reached location can differ and one might have to reinterpret the parameterization. Therefore, different approaches have been discussed. Another concern is the training of the movement model which can be complicated when demonstrations of different persons are used. An approach is to use an intermediate embodiment for representing and learning the movements. This includes also the representation of the effect/meaning of the movements with respect to this intermediate embodiment.

In the specific application, the mapping has been chosen such that the effect/meaning is preserved (if one is neglecting the rescaling to the robot's size, which is compensated for by rescaling the pa-

rameterization as well). The appearance is preserved under this constraint as closely as possible. The maximal imitation error is about 1cm which includes the inaccuracy of the PHMM-based synthesis, whereas the effect (the reached target location) is preserved with an accuracy of about 2mm. Here it is worthwhile to note that the inaccuracy of 1cm is necessary to preserve the effect on the robot's embodiment, which differs from the demonstrator's embodiment. A more precise PHMM would not improve the imitation significantly. In other applications, the appearance of movements or the dynamics of the movements depending on the parameters [126] might be more important. However, by emphasizing the importance of the effect, the learned action model could be used to grasp objects at arbitrary locations in order to relocate the objects. For the considered reaching action, an IPHMM has been utilized.

Chapter 6

Tracking in Action Space

6.1 Introduction

Human communicative actions such as pointing ("Give me this.") or object grasping are typical examples of human actions in a human-to-human communication problem [7, 65]. These actions are usually context-dependent and their parametrization defines an important piece of their information [134, 7]. To capture these communicative actions is challenging because a) the capturing should be independent of scene parameters such as viewing direction or viewing distance and b) one needs complex action models that allow to recover *what* action is performed and *which* parameters it has. The observation that the parameters of an action carry important information about its *meaning* is discussed in the introduction (Chapter 1).

A strategy for recognizing such actions [134, 98] is to first track the human movements using a 3D body tracker. In a second step, these tracks are fed into an action recognition engine, such as HMMs [75, 136] or even parametric HMMs (PHMMs) [134]. Considering the first ingredient of this strategy, it was recently pointed out [70] that 3D tracking and pose estimation, especially from monocular views, is still a non-trivial problem. Common approaches to motion capturing are model-based generative ones [26, 81, 116, 117], which compare a synthesized candidate pose with the image data.

This chapter follows the argument that such a two-step approach (discussed above) is an unnecessary complication. Human actions are usually goal-directed and are performed within a certain context, as eating, or cooking, etc. Furthermore, actions are often performed on objects [40, 64] which leads to the observation that the objects can prime the actions performed on them (e. g., reaching, pointing, grasping, pushing-forward) [44, 22]. Thus, the suggestion is to look at 3D human body tracking from an object and context-driven perspective: Instead of estimating the whole set of joint angles that makes a human model fit to the observation, one looks for an action causing a pose that fits to the observation. By replacing in a particle filter approach the propagation model for joint angles [81, 26] with a propagation model for human actions, it is possible to re-formulate the 3D tracking problem as a problem of recognizing the action itself (including its parameters). In other words, instead of having to estimate the high-dimensional parameter vector of the human body model, one samples the action and its parameters in the low-dimensional space of the actions and their parameters (*Action Space*). By using a generative model for the action representation, one can deduce the appropriate 3D body pose from the sampled action parameters and compare it to the input data. This approach is named here: *Tracking in Action Space*.

In this thesis, parametric hidden Markov models (PHMMs, Chapter 3) are used as the generative action model. While classical HMMs can recognize essentially only a specific trajectory or movement, PHMMs are able to model different parametrizations of a particular movement. For example, while a common HMM would be able to recognize only one specific pointing action, PHMMs are able to recognize pointing action into different directions [134]. Furthermore, PHMMs are generative models which means that for recognizing an observation they compare it with a model of the expected observation.

In the experiments in this chapter, the action space spans over the human arm actions (pointing, taking an object etc.), the corresponding parameterization (e. g., the point to location), plus a timing parameter. One might argue that such an approach cannot be used for general 3D body tracking because the action space will always be too limited. However, following the arguments in [103, 102, 37, 60, 38] that human actions are composed out of motor primitives similarly to human speech being composed out of phonemes, support the hypothesis that the limited action space considered here can be generalized to the space spanned by the action primitives, where a stochastic action grammars could be used as in [58, 37, 38] to model even complex actions through such a basic set of primitives. Furthermore, the work [38] explains how a language for human actions can be generated based on grounded concepts: kinetology, morphology and syntax.

For estimating the action and action parameters during the tracking, the classical Bayesian propagation over time is used. The propagation over time provides an excellent embedding for the tracking in action space, including the use of primitives and action grammars. Indeed, if one considers the action space to consist of action primitives and if one assumes complex actions to be composed out of these primitives according to some grammatical description, then the Bayesian propagation estimates at each time step the likelihood of each action primitive, and, while moving from one action primitive to the next one as the complex action progresses, the corresponding action grammar specifies the propagation density in the Bayesian propagation.

Key Contributions. The key contributions of the outlined approach are:

- the introduction of the concept, Tracking in Action Space, by posing the 3D human pose estimation problem as one of action and action parameter recognition
- a concise formulation in the particle filtering framework
- the concept of approaching action recognition as a context and object dependent problem
- recovery of 3D pose and action recognition plus action parameters in a singe framework
- robust tracking that is content with monocular video data
- processing in real-time

Chapter Outline. The chapter is structured as follows: The introduction is completed with the related work section (Section 6.2). In Section 6.3 the concept, Tracking in Action Space, is discussed. Section 6.4 explains the details on using the PHMMs for modeling the action space and the application of particle filtering to establish the tracking in this space. In Section 6.6, the calculation of the observation likelihood of a particle for a given image is discussed. The discussion includes the human model, the mapping of the particle to a model pose, and the image preprocessing which precedes the evaluation of the likelihood. Aspects of the implementation, which enables the implemented framework to perform the tracking in real-time (online), are discussed in Section 6.5. Section 6.7 provides numerous experiments with different actions. Section 6.8 concludes the chapter with final comments.

6.2 Related Work

As it can be seen in the survey [80], early deterministic methods, as gradient based methods, (e.g., [20]), have been overcome by stochastic methods due to problems as depth disambiguations, occlusions, etc. The methods rage from the basic particle filtering, as described in [57], to, e.g., belief propagation [41, 70]. Efficient techniques for particle filtering [26, 117, 31, 81] in combination with (simple) motion models [116] to constrain the particle propagation or the state space [128] are investigated since the number of required particles generally scale exponentially with the degree of freedom. Novel frameworks use, for example, multistage approaches ([70] considers the stages: coarse tracking of people, body part detection and 2D joint location estimation, and 3D pose inference) or implement various constrains ([41] considers the constrains concerning self-occlusion, kinematic, and appearance and uses belief propagation to infer the pose within a graphical model). Contrary to the simple motion models, which roughly approximate the state space used by certain motions (e.g., through a linear subspace [128]), advanced approaches, as, for example, locally linear embedding allow to learn [28] the intrinsic low dimensional manifold, or aim at the learning of nonlinear dynamic system (NLDS) on motion data, as it is approached through the Gaussian process model developed in [131] in a more efficient way. Interestingly (in the context of this thesis), the learning of NLDS requires a vast amount of training data [131], whereas *"classic"* HMMs can be easily trained but are limited in their expressiveness for complex motions. In this chapter, the *parametric extension* of HMMs is utilized.

6.3 The Approach to Tracking in Action Space

In this section, the approach to Tracking in Action Space, is discussed in detail. The discussion begins with the classical Bayesian propagation over time (see Appendix C):

$$p_t(\boldsymbol{\omega}_t) \propto p(\boldsymbol{I}_t|\boldsymbol{\omega}_t) \int p(\boldsymbol{\omega}_t|\boldsymbol{\omega}_{t-1}) p_{t-1}(\boldsymbol{\omega}_{t-1}) d\boldsymbol{\omega}_{t-1}. \tag{6.1}$$

The variable $\boldsymbol{I}_t$ denotes the current observation at time step t. The density $p_t(\boldsymbol{\omega}_t)$ denotes the posterior density of the random variable (r.v.), $\boldsymbol{\omega}_t$, at time step t. The propagation density $p(\boldsymbol{\omega}_t|\boldsymbol{\omega}_{t-1})$ approximates the current r.v. $\boldsymbol{\omega}_t$ given a previous state $\boldsymbol{\omega}_{t-1}$ without the knowledge of the current observation. The likelihood $p(\boldsymbol{I}_t|\boldsymbol{\omega}_t)$ of $\boldsymbol{\omega}_t$ introduces the knowledge of the current observation.

If applied for 3D human body tracking, the r.v. $\boldsymbol{\omega}$ usually specifies the set of joint angles for some human body model (see, e.g., [81]). The propagation density is used to constrain the r.v. $\boldsymbol{\omega}_t$ to the most likely pose values at each time step t [81, 31, 116, 26]. In order to compute the likelihood $P(\boldsymbol{I}_t|\boldsymbol{\omega}_t)$, a human body model is synthesized using the pose values from $\boldsymbol{\omega}_t$ and is then compared with the input image $\boldsymbol{I}_t$.

For the tracking in action space, the r.v. $\boldsymbol{\omega}$ is used to control some parametric action models with $\boldsymbol{\omega} = (a, \boldsymbol{\theta}, \tau)$. Here, the parameter a identifies which action it is, $\boldsymbol{\theta}$ is a vector specifying the parameters of action a, and τ is a timing parameter which specifies the progress of action a. The action models are in this work PHMMs. Consider, for example, that the action $a = 1$ is a pointing action which is modeled by a parametric HMM $\boldsymbol{\lambda}_a^{\boldsymbol{\theta}}$ then $\boldsymbol{\theta}$ specifies the pointed to location. The parameters $a = 1$, $\boldsymbol{\theta}$, and τ specify then basically a body pose, where τ specifies the PHMM state of $\boldsymbol{\lambda}_{a=1}^{\boldsymbol{\theta}}$ which defines the body pose. Hence, the likelihood $P(\boldsymbol{I}_t|\boldsymbol{\omega}_t)$ can be computed by synthesizing the body model in the pose $\boldsymbol{\omega} = (a, \boldsymbol{\theta}, \tau)$ and comparing the synthesized model with the observation $\boldsymbol{I}_t$.

In the case of parametric actions, the propagation density $p(\boldsymbol{\omega}_t|\boldsymbol{\omega}_{t-1})$ can be considerably simplified. Assuming that a human finishes one action primitive before starting a new one, the action identifier a is constant until an action is finished. If an action grammar model for complex human actions [38, 37] is given, then the corresponding action grammar can be used to control the progression of a. The timing parameter τ changes according to the action model (e.g., by using the transition matrix of the PHMM $\boldsymbol{\lambda}_a^{\boldsymbol{\theta}}$). Likewise, the action parameters $\boldsymbol{\theta}$ are approximately constant until the action primitive is completed. However, usually at least some of the parameters of the parameter vector $\boldsymbol{\theta}$ are not known in advance and need to be estimated during the tracking. In the case of bi-parametric primitives that are parameterized by $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ and start with the hand at a location $\boldsymbol{\theta}_1$ and end with the hand at a location $\boldsymbol{\theta}_2$, then the parameter $\boldsymbol{\theta}_1'$ of a following primitive is given by the parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ of the previous primitive, i.e. $\boldsymbol{\theta}_1' = \boldsymbol{\theta}_2$ (Section 6.7.3).

In the following sections, the application of PHMMs in the tracking scheme is discussed (Section 6.4). The basic framework is then completed with the discussion of the computation of the observation likelihood (Section 6.6).

6.4 PHMM-Based Action Tracking

The parametric actions are modeled through PHMMs. In each tracking scenario a set $\mathcal{A} = \{1, \ldots, M\}$ of M actions are modeled. In a very simple scenario the number of actions is $M = 1$. In the following, the training, parametrization, and utilization of the PHMMs are discussed. The utilization of particle filtering is discussed in Section 6.4.2 and the sections which follow.

6.4.1 Aspects of the Action Models

For each parametric action a, a PHMM is trained on demonstrations of the action a. The PHMM is then the model of the parametric action a. The settings of the PHMM training on motion data are described in Section 4.1.2. The PHMMs that are used are usually QPHMMs (Section 4.2.3), which are trained as left-right no-skip PHMMs with time restrictions and a finish state.

The training sequences are represented in a similar way as discussed in Section 4.2.1. Each sample $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})$ of a training sequence represents an arm pose via a set of body-fix points, which are the locations of the shoulder, elbow, and the hand. The location of the hand corresponds to the end of the hand. These points are represented in a reference frame that is used during the recording of the demonstrations via the Vicon system. Since the location of the shoulder of the demonstrator's body and the orientation of the body is known in the resting pose of the demonstrator, the coordinates of the points can be computed for the reference system of the shoulder of the human body model. This allows one to use an arm pose $\boldsymbol{x}$ to set the arm pose of the model even when the global pose (rotation and translation) of the model in the reference system of the scene (Figure 6.1) is arbitrary. However, the different representations of the pose $\boldsymbol{x}$ is neglected in the following. The important aspect is only that an arm pose $\boldsymbol{x}$, which drawn from the model, can be used to set the arm pose of the human body model. The setting of the arm pose is discussed in Section 6.6.1.2, where $\boldsymbol{x}$ is assumed to be given in the system of the shoulder. For simplicity, the orientation of the reference system which is used during the tracking experiments has the same orientation of the system used during the recording with the Vicon system. Only the origins of these systems differ. The setup used for tracking is shown in Figure 6.1. Usually, the PHMMs are parameterized by table-top location(s). A table-top location is

described through normalized coordinates $(u, v) \in [0, 1]^2$. The u, v coordinates describe the region of the table-top for which demonstrations of the actions are available. The coordinates (u, v) are chosen such that u and v describe the directions of the x-axis and y-axis of the tracking/recoding system. (But this is only true since the orientation of the body model matches the orientation of the demonstrator during the recording of the motion data.) A PHMM that is parametrized by a single table-top location uses the parametrization $\boldsymbol{\theta} = (u, v)$, but the parameters (u, v) of the modeled action can be converted easily to the corresponding 3D location in the reference frame used during the experiments.

In the following it is worthwhile to consider the synthesis of a pointing action for a location $\boldsymbol{z}_0$ at the table-top. The global pose of the body model is predefined. A PHMM $\boldsymbol{\lambda}^{\boldsymbol{\theta}}$ models the parametric pointing action. The location $\boldsymbol{z}_0$ corresponds to a parameter $\boldsymbol{\theta}_0 = (u, v)$. If now a sequence of arm poses $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ (a prototype action) is generated from the HMM $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\boldsymbol{\theta}_0}$, for example, by taking the state means, then the sequence can be animated through the model by using the sequence of arm poses to set the arm poses in the image frames $t = 1, \ldots, T$. The animation is then similar to the performance of an pointing action by a person when the person is sitting at the same location and is pointing to the location $\boldsymbol{z}_0$. However, the animated prototype action can differ from the performance in two ways: the temporal progress can be different, and the performance can differ spatially. The spacial variation of the action is basically encoded in the covariance matrices of the PHMM states. In the tracking experiments it turned out that this variance cannot be neglected, since otherwise the tracking becomes unstable. Therefore, the evaluation of the likelihood function (discussed below) makes use of the covariance matrices. The temporal progress is encoded in the transition matrix of the PHMM. In a first approach to tracking in action space [51], the transition matrix was used to propagate the time parameter τ, where τ specifies the state of the PHMM. However, the propagation of the time parameter from one frame to the next frame should correspond to the average progress of an action, which is recorded during the tracking with a certain frame rate. If the frame rate during the recording of the demonstrations (50fps) does not correspond with the frame rate during the tracking (the real time tracking works with 8fps), this correspondence is not given. Therefore, a uniform parametrization $\tau \in [0, 1]$ of the progress of an action is chosen. The propagation of the parameter τ is then adopted to the actual frame rate that is used during the tracking (see Section 6.4.4). The mean vector $\boldsymbol{\mu}_\tau^{\boldsymbol{\theta}}$ and the covariance matrix $\boldsymbol{\Sigma}_\tau^{\boldsymbol{\theta}}$ are computed for a parameter $\tau \in [0, 1]$ by linear interpolation:

$$\boldsymbol{\mu}_\tau^{\boldsymbol{\theta}} = (1 - \tau')\boldsymbol{\mu}_{i_1}^{\boldsymbol{\theta}} + \tau'\boldsymbol{\mu}_{i_2}^{\boldsymbol{\theta}}, \tag{6.2}$$

$$\boldsymbol{\Sigma}_\tau^{\boldsymbol{\theta}} = (1 - \tau')\boldsymbol{\Sigma}_{i_1}^{\boldsymbol{\theta}} + \tau'\boldsymbol{\Sigma}_{i_2}^{\boldsymbol{\theta}}, \tag{6.3}$$

$$\tau' \equiv \tau(N - 1) - (i_1 - 1), \tag{6.4}$$

$$i_1 \equiv 1 + \lfloor \tau(N - 1) \rfloor, \quad \text{(“floor”)} \tag{6.5}$$

$$i_2 \equiv 1 + \lceil \tau(N - 1) \rceil, \quad \text{(“ceiling”)} \tag{6.6}$$

where N is the number of states of the PHMM. The vector $\boldsymbol{\mu}_{i_j}^{\boldsymbol{\theta}}$ and the matrix $\boldsymbol{\Sigma}_{i_j}^{\boldsymbol{\theta}}$ are the mean and covariance matrix of the states i_j, which depend on the PHMM parameter $\boldsymbol{\theta}$. Note, for a fixed $\boldsymbol{\theta}$ the vector $\boldsymbol{\mu}_\tau^{\boldsymbol{\theta}}$ describes a prototype trajectory of the action, where $\tau \in [0, 1]$ is the time parameter.

6.4.2 Particle Filtering

The particle filter framework [57] is explained in Appendix C. Here, only the essentials of the three particle filtering steps of an iteration $t - 1 \to t$ are explained. The state density $p_{t-1}(\boldsymbol{\omega})$ is represented

through a set $\mathcal{S}_{t-1} = \{(\boldsymbol{\omega}_{t-1}^n, \pi^n)|\ n = 1, \ldots, N\}$ of π^n-weighted samples. From the sample set $\mathcal{S}_{t-1}$ a set of samples $\{\boldsymbol{\omega}_t^n | n = 1, \ldots, N\}$ is generated. Each samples is propagated (Section 6.4.4), which results in a sample set $\{\hat{\boldsymbol{\omega}}_t^n | n = 1, \ldots, N\}$. For each sample $\hat{\boldsymbol{\omega}}_t^n$, the likelihood needs to be evaluated. Based on the likelihood measurements, new weights π_t^n are attached to each sample $\hat{\boldsymbol{\omega}}_t^n$. The weighted sample set $\mathcal{S}_t = \{(\hat{\boldsymbol{\omega}}_{t-1}^n, \pi^n)|\ n = 1, \ldots, N\}$ is then an approximation of the density $p_t(\boldsymbol{\omega})$. Based on the set $\mathcal{S}_t$ different features of the density $p_t(\boldsymbol{\omega})$ can be approximated (Section 6.4.5). In the experiments $N = 400$ samples are used, which is a trade off between computation time and tracking quality.

6.4.3 Evaluating the Likelihood

In principle the likelihood of the state $\boldsymbol{\omega}_t^n = (a, \boldsymbol{\theta}, \tau)$ can be computed for the current observation $\boldsymbol{I}_t$ simply by $p(\boldsymbol{I}_t|\boldsymbol{\omega}_t) = b_{a,\tau}^{\boldsymbol{\theta}}(\boldsymbol{I}_t)$ if the observation space is the same as the state space of the arm poses. However, here $\boldsymbol{I}_t$ is the data of the recoded image(s) of the current frame, and $p(\boldsymbol{x}|\boldsymbol{\omega}) = b_{a,\tau}^{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{a,\tau}^{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{a,\tau}^{\boldsymbol{\theta}})$ defines the density of the arm pose. For an arm pose $\boldsymbol{x}$ the likelihood $p(\boldsymbol{I}_t|\boldsymbol{x})$ can be computed by comparing the model in the pose with the image(s). This is discussed in Section 6.6.4. The likelihood $p(\boldsymbol{I}_t|\boldsymbol{\omega})$ can be written as

$$p(\boldsymbol{I}_t|\boldsymbol{\omega}_t) = \int_x p(\boldsymbol{I}_t|\boldsymbol{x}) p(\boldsymbol{x}|\boldsymbol{\omega}_t)\ \mathrm{d}x, \tag{6.7}$$

which could be approximated through stochastic sampling. In the implementation only a very coarse approximation of $p(\boldsymbol{I}|\boldsymbol{\omega}) \approx p(\boldsymbol{I}|\boldsymbol{x})$ is made by drawing a single pose $\boldsymbol{x}^n$ from the density $p(\boldsymbol{x}|\boldsymbol{\omega}_t^n) = b_{a,\tau}^{\boldsymbol{\theta}}(\boldsymbol{x})$. The intention is here to use preferably a large particle set, instead of evaluating the likelihood $p(\boldsymbol{I}_t|\boldsymbol{\omega}_t^n)$ very accurately. As a consequence, a single pose $\boldsymbol{x}^n$ corresponds to a particle $\boldsymbol{\omega}_t^n$. The use of M samples for evaluation of the likelihood would result in $N \cdot M$ evaluations of the likelihood $p(\boldsymbol{I}_t|\boldsymbol{x})$, where N is the number of particles. It worthwhile to note that the evaluation of $p(\boldsymbol{I}_t|\boldsymbol{x})$ (Section 6.6.4) is the computationally expensive operation not the remaining operations of the particle filtering.

6.4.4 Particle Propagation

A state $\boldsymbol{\omega} = (a, \boldsymbol{\theta}, \tau)$ is propagated to a state $\hat{\omega} = (\hat{a}, \hat{\boldsymbol{\theta}}, \hat{\tau})$. As mentioned in Section 6.4.2, the action type a is considered usually as constant, i.e. $\hat{a} = a$.

An optimal prediction $\hat{\tau}$ of the temporal progress τ would be $\hat{\tau} = \tau + f/T_a$, where f is the frame rate of the tracking and T_a is the average duration of the action a. However, an actual performance of an action can have different progression rates and a different overall duration. Therefore, a diffusion is applied, i.e. $\hat{\tau} = \tau + f/T_a(1 + x)$, where x is drawn from the normal distribution with $\sigma = 0.7$.

As mentioned in Section 6.3, the action parameters $\boldsymbol{\theta} = (u, v)$ of a pointing action could be considered approximately as constant. However, when the first part of an action is tracked the parameters are usually not known. Therefore, the following diffusion is performed $\hat{u} = u + \Delta u$ and $\hat{v} = v + \Delta v$, where Δu and Δv are drawn from a normal distribution with a standard deviation $\sigma = \sigma(\tau)$. A plot of the function $\sigma(\tau)$ is shown in Figure 6.9. For $\tau \approx 0$ the diffusion with very large, which reflects the assumption that the action parameters are not known when the first part of an action is tracked. The diffusion becomes smaller with the progress of the action. In addition the range of the parameters $\hat{u}$ and $\hat{v}$ is constrained to the interval $[-0.2, 1.2]$.

However, when the tracking process is initialized one has to generate first the samples $\boldsymbol{\omega}^n$. In the case of a single pointing action, a sample $\boldsymbol{\omega}^n = (a, \boldsymbol{\theta} = (u, v), \tau)$ is generated by setting $a = 1$ and $\tau = 0$. The values u and v are chosen from the interval $[0, 1]$ randomly.

6.4.5 Making Estimates

The posterior density $p_t(\boldsymbol{\omega}_t)$ is represented through the π^n-weighted samples $\boldsymbol{\omega}^n = (a^n, \boldsymbol{\theta}^n, \tau^n)$. For each sample $\boldsymbol{\omega}^n$, there is a corresponding body pose vector $\boldsymbol{x}^n$. Different properties of the density $p_t(\boldsymbol{\omega}_t)$ can be computed (approximatively):

$$p_t(a) = \sum_{n \in \mathcal{N} : a^n = a} \pi^n, \tag{6.8}$$

$$p_t(a, \tau \in [\tau_1, \tau_2]) = \sum_{n \in \mathcal{N} : a^n = a, \tau \in [\tau_1, \tau_2]} \pi^n, \tag{6.9}$$

$$\boldsymbol{\mu}_{\boldsymbol{\theta},a} = \mathbb{E}[\boldsymbol{\theta}|a] \approx \frac{1}{N_a} \sum_{n \in \mathcal{N} : a^n = a} \pi^n \boldsymbol{\theta}^n, \tag{6.10}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta},a} = \mathrm{cov}[\boldsymbol{\theta}|a] \approx \frac{1}{N_a} \sum_{n \in \mathcal{N} : a^n = a} \pi^n (\boldsymbol{\theta}^n - \boldsymbol{\mu}_{\boldsymbol{\theta},a})^\top (\boldsymbol{\theta}^n - \boldsymbol{\mu}_{\boldsymbol{\theta},a}), \tag{6.11}$$

$$N_a \equiv \sum_{n \in \mathcal{N} : a^n = a} \pi^n, \tag{6.12}$$

$$\mathcal{N} \equiv \{1, \ldots, N\}. \tag{6.13}$$

Note, the weights π^n are normalized, i.e. $\sum \pi^n = 1$. A high posterior probability $p_t(a) \approx 1$ of a certain action $a = a_0$ indicates that the currently tracked action is likely the action a_0. This can be seen in the experiments. However, assume the case that two action models are used which describe actions that are starting from the same base pose. If the tracked person is still in the base pose, then most particles have $\tau^n \approx 0$. A value $p_t(a) \approx 1$ is then not very indicative, since no action has been performed. In such an recognition experiment, the probability $p_t(a, \tau \in [\tau_1, \tau_2])$ used, where the interval $[\tau_1, \tau_2]$ is smaller than the interval $[0, 1]$ (note, $\tau \in [0, 1]$).

The means $\bar{u}$ and $\bar{v}$ and deviations $\tilde{u}$ and $\tilde{v}$ of the action parameters u and v of a specific action a are given by

$$\bar{u} = [\boldsymbol{\mu}_{\boldsymbol{\theta},a}]_1, \ \bar{v} = [\boldsymbol{\mu}_{\boldsymbol{\theta},a}]_2, \ \ \tilde{u} = \sqrt{[\boldsymbol{\Sigma}_{\boldsymbol{\theta},a}]_{1,1}}, \ \tilde{v} = \sqrt{[\boldsymbol{\Sigma}_{\boldsymbol{\theta},a}]_{2,2}}. \tag{6.14}$$

A value which is similar to both deviations $\tilde{u}$ and $\tilde{v}$ is given by

$$\widetilde{uv} = \sqrt[4]{\det \boldsymbol{\Sigma}_{\boldsymbol{\theta},a}}. \tag{6.15}$$

Two different methods to estimate the pose are to select the best observation $\boldsymbol{x}_{\max}$ or to take the mean $\bar{\boldsymbol{x}}$ of the poses, i.e.

$$\boldsymbol{x}_{\max} = \arg\max_{\boldsymbol{x}^n} \pi^n, \tag{6.16}$$

$$\bar{\boldsymbol{x}} = \sum_{n=1}^{N} \pi^n \boldsymbol{x}^n. \tag{6.17}$$

The best observation $\boldsymbol{x}_{\text{max}}$ provided good pose estimates in the experiments. However, the sequence of the pose estimates of a tracked sequence is not very smooth. Whereas, the sequence of pose means is smooth. But these estimates are very imprecise. Sometimes the mean pose lags behind the true pose, and sometimes the mean pose hurries ahead of the true pose. Therefore, a mixture of both approaches is used in the experiments: First, an estimate $\boldsymbol{x}_0 = \boldsymbol{x}_{\text{max}}$ is selected. The estimate is then refined to

$$\boldsymbol{x}_i = \frac{1}{W} \sum_n w_n \boldsymbol{x}^n, \tag{6.18}$$

$$w_n \equiv \pi^n \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu} = \boldsymbol{x}_{i-1}, \boldsymbol{\Sigma} = \sigma_i \boldsymbol{I}), \tag{6.19}$$

$$W \equiv \sum_n w_n, \tag{6.20}$$

where in the iterations $i = 1, 2, 3$ the values $\sigma_i = 0.15, 0.10, 0.05$ are used. Each iteration is basically a kernel regression. The pose $\boldsymbol{x}_3$ is then the finally estimated pose.

6.5 Real Time Aspects

The implementation works with a single camera view and 400 particles in real time (8 frames per second) on a single workstation. The work station has a Nvidia Geforce 295 GTX graphics card and 8 Intel Xeon cores running at 2.53GHz. The crucial tasks are performed by 8 threads. The programming of the threads follows a producer consumer scheme with multiple stages.

One thread is performing the particle filtering including the particle propagation, sampling etc. This thread generates also the pose states. A rendering thread performs the rendering of the model for each pose state and stores the rendered silhouettes in a buffer. A camera thread is responsible for reading the image(s) of the frames from the camera(s) and performing the image pre-processing (see Section 6.6.4) that is required for the contour matching. This thread has two buffers for all images it needs to store for one frame. The thread is supposed to start with capturing and pre-processing of the next frame, while all other threads are working on the current frame. Thus the thread is supposed to be done with the processing of the current frame when the rendering thread generates the first rendered silhouettes for the current frame. Since the matching for the evaluation of the likelihood of the silhouettes requires more processing time (see Section 6.6.4), 5 threads are performing the evaluation of the likelihood. Theses threads start to operate when the pre-processing of the current frame is indeed done by the camera thread. These threads consume the rendered silhouettes and compute the likelihoods. This includes the generation of the contours, the matching of the contours and finally the computation of the likelihood (see Section 6.6.4). Once all likelihoods are computed for the current frame, all buffers for the current frame can be reused and the particle process can perform the next iteration of the particle framework.

Even though not all cores are 100% utilized, the frame work cannot be improved easily. The rendering thread can be seen, for example, as a bottle neck, the rendering utilizes also the CPU. However, the rendering in two parallel threads reduced the rendering throughput.

6.6 The Observation Model

The approach to tracking is an analysis-by-synthesis approach. The particle filter produces for each frame a number of hypotheses $\boldsymbol{x} = \boldsymbol{x}^n$ of the body pose which need to be evaluated on the basis of the image data. The captured image data $\boldsymbol{I}$ can be a captured image of a single camera or the images from different camera views (see Figure 6.1). The scene with the body model of the arm is then rendered in the camera view(s). Therefore, the camera parameters and the body pose $\boldsymbol{x}$ are applied to the model. The synthesis of the view(s) is then compared with the actual observation $\boldsymbol{I}$ to compute the likelihood measure $P(\boldsymbol{x}|\boldsymbol{I})$. The measurement that is used in the implementation is based on the contour(s) of the rendered silhouette(s) and the gradients in the image(s). The gradient intensities are interpreted as edge features. A simple approach to evaluate the silhouette cue is discussed in Section 6.6.3. However, the edge features are often very weak. Therefore, a shape matching approach was developed (Section 6.6.4) to achieve a more robust tracking. The scene model and the human body model are discussed in Section 6.6.1. An important aspect is how the arm pose $\boldsymbol{x}$ is applied to the model (Section 6.6.1.2). One aspect is that the arm pose $\boldsymbol{x}$ is drawn from a density of the pose state space and is not necessarily a valid pose. Short notes on the rendering and the contour extraction from the silhouette are given in Section 6.6.2.

6.6.1 Scene Model

A picture of the scene for the experiments is shown in Figure 6.1. The chess pattern is removed after the calibration process. A proper model of the scene is necessary to evaluate a hypothesis in an analysis-by-synthesis approach. The whole scene model must model the important aspects of the scene such

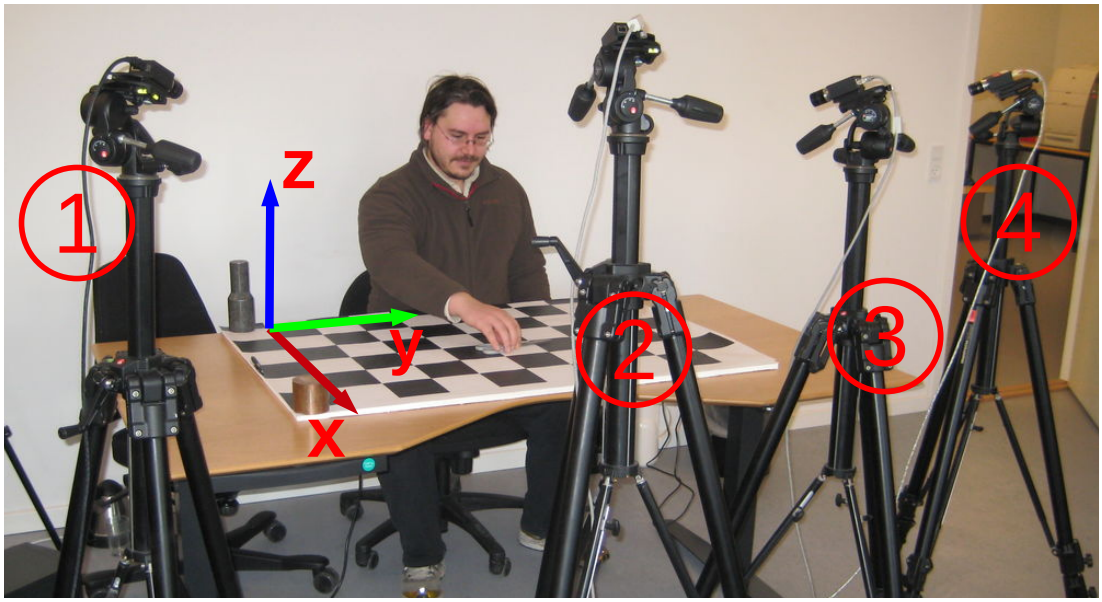


Figure 6.1: Scene for Tracking. The picture shows the scene as it is during the tracking. The locations and the numbers to address the cameras are shown. The chess pattern is removed after the calibration of the cameras. The chess pattern defines the reference frame of the scene as indicated in the picture.

that it makes sense to compare the rendered scene with the observed scene. Basics to scene modeling are discussed in Appendix D. The main aspects of the scene are the reference frame of the scene, the cameras and the body of the human. The scene is described as an OpenSG [3] scene graph, where the root node corresponds to the reference frame of the scene. (A short description of scene graphs w.r.t. OpenSG is given in Section D.4.) The scene graph of the upper body model (Section 6.6.1.1) is attached to the root node with an additional transformation node in between, which allows setting the pose of the upper body such that the torso of the model matches the torso of the observed person. The reference frame is defined by the calibration pattern as shown in Figure 6.1. The calibration pattern is used to determine the external and internal camera parameters (see Section D.3) of each camera. This is done by using the Camera Calibration Toolbox for Matlab [1] and some camera images of the chess pattern. The internal and external parameters of a camera are used to set the projection matrix (Section D.3) during the rendering of the scene. This results in a synthesized view of the body model. Under optimal circumstances the rendered body parts in the synthesized image should spatially match with those in the observed view. However, the cameras used are not optimal and show, for example, slight distortion effects. Further aspects of the rendering are discussed in Sections 6.6.1.2 and 6.6.2.

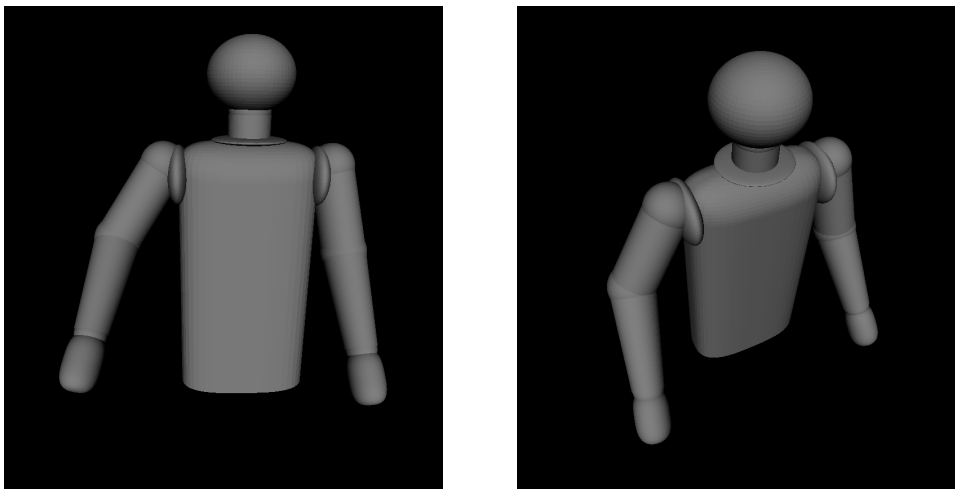


Figure 6.2: Upper Body Model. The pictures show two views on the upper body model in an articulated state.

6.6.1.1 The Human Model

The model is represented as an OpenSG[3] scene graph (see Section D.4). The kinematic structure of the model is similar to the one used in [26]. Actually only the upper body is modeled in this thesis. An illustration of the graph is shown in Figure 6.3. The graph is structured in such a way that the basic geometries are composed to define the body segments. The segments are composed in a hierarchical structure which includes the kinematic chains of the joints of a human, e. g., the right hand is articulated by the chain: wrist joint, elbow joint, shoulder joint, etc. The transformation nodes in Figure 6.3 which represent the joints of the right arm have a green color. These joint nodes are basically homogenous 4×4 transformation matrices (see Section D.1) which include 3×3 rotation matrices. The rotation matrices can be set to change (articulate) the state/pose of the model. However, in order to articulate

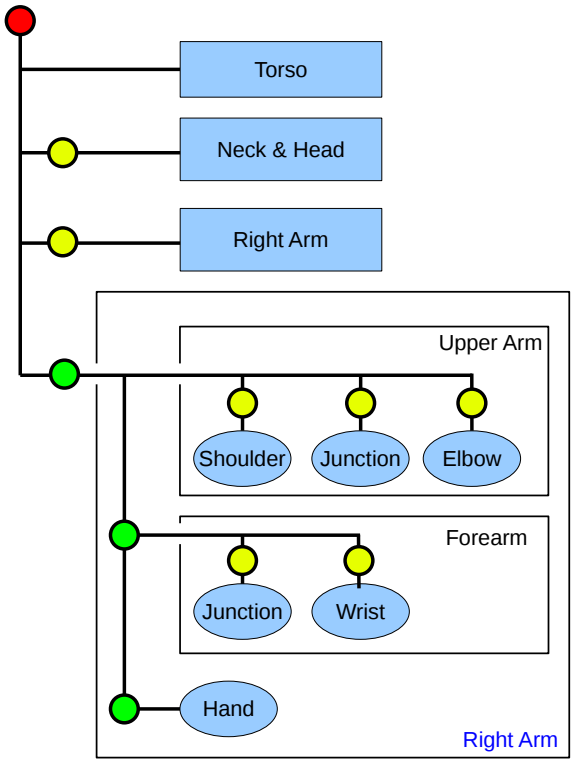


Figure 6.3: Simplified Scene Graph of the Upper Body Model.

the model by hand, rotation axes similar to the joint axes of the humanoid robot (see Section 5.2.1) have been established. For example, the rotation of a shoulder joint node is then set on the basis of three joint axes (abduct, flection, and twist). The segments of the model are composed of superquadratics [121], sections of superquadratics, cylinders and balls. Pictures of the rendered model are shown in Figure 6.2.

6.6.1.2 Setting the Arm Pose

The arm pose needs to be set for the rendering as given by a hypothesis $\boldsymbol{x}$. The arm pose is defined as $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})$, where $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ are 3D vectors which specify the location of shoulder, elbow, and hand of the right arm. The origin of the 3D reference system for $\boldsymbol{p}$, $\boldsymbol{q}$ and $\boldsymbol{r}$ is the initial location of the shoulder of the model.

For a given $\boldsymbol{x}$ the shoulder is moved to the location $\boldsymbol{p}$. As mentioned above, the vector $\boldsymbol{x}$ is drawn from a density, i.e. the observation density of an HMM state. Thus, the lengths of the upper arm $|\boldsymbol{p}-\boldsymbol{q}|$ and forearm $|\boldsymbol{q}-\boldsymbol{r}|$ are not preserved. The elbow location $\boldsymbol{q}$ is then refined with respect to the model's arm lengths to the nearest possible point $\boldsymbol{q}'$ that is on the plane defined through $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$. The arm of the model can then be articulated such that the elbow and hand locations of the model match the locations $\boldsymbol{q}'$ and $\boldsymbol{r}$. This procedure of refining $\boldsymbol{q}$ to $\boldsymbol{q}'$ and the computation of the joint rotations is basically the same procedure that has been established in Section 5.2.4 for the mapping of an arm pose to the robot's embodiment. However, here the shoulder moves to the location $\boldsymbol{p}$ and the rotation matrix of the shoulder can be copied directly into the joint node of the shoulder of the body model. The decomposition into three shoulder joints (see Section 5.2.4) is not necessary. The rather unlikely

case that $|\boldsymbol{p} - \boldsymbol{r}|$ is greater than the overall arm length of the model is mapped to an arm pose, where the stretched arm points from the shoulder location $\boldsymbol{p}$ in the direction of the point $\boldsymbol{r}$.

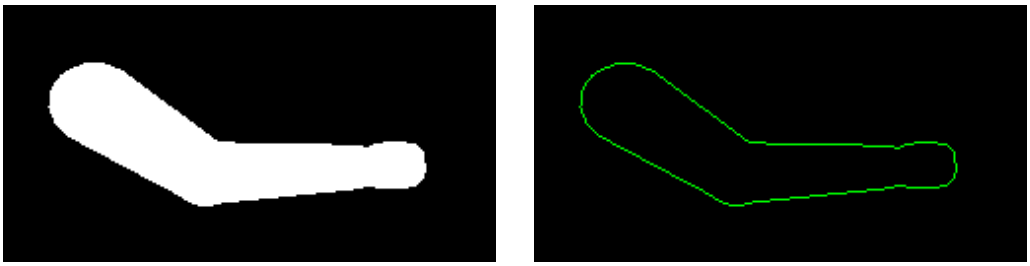


Figure 6.4: Silhouette and Contour. The picture on the right shows the contour of the rendered silhouette (left) of the arm of the upper body model. The picture (left) is a cropped version of the rendered 320×240 image.

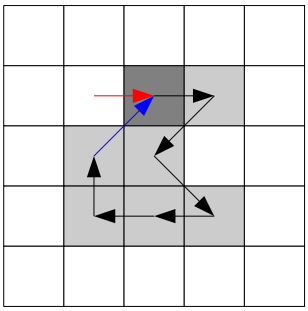


Figure 6.5: Iteration Steps of the Contour Algorithm.

6.6.2 Rendering and Contour Extraction

In the experiments, usually only the arm of the model is rendered. The rest of the model is hidden. Since the model is rendered on the graphics card, the image of the rendered arm needs to be transferred for further processing into the main memory of the computer. The transfer from graphic card to main memory is slower than the memory of both components. Therefore, the rendered images are first converted into a single channel black/white image on the graphics cards and copied afterward. Both operations are performed through a CUDA [2] implementation on the graphics card. A black/white image of the rendered arm is shown in Figure 6.4 (left). For the evaluation of the likelihood function, the contour of the rendered arm silhouette is needed. A straight forward approach to find the contour pixels is to test for each foreground pixel if it has a neighboring background pixel. The collection of such pixel locations defines then the set $\mathcal{C}$ of contour pixels (see Figure 6.4 (right)) that can be used to compute the likelihood as described in the following Section 6.6.3.

However, the contour matching approach to define a more robust likelihood measure (discussed in Section 6.6.4) requires determination of the contour pixels as a sequence $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N$ of pixel locations $\boldsymbol{p}_i = (x_i, y_i) \in \mathbb{N}^2$, which define the contour as a closed loop. The approach determines the contour pixels iteratively starting from a pixel location $\boldsymbol{p}_1$. The first contour pixel $\boldsymbol{p}_1$ is determined by performing a row wise scan from top to bottom. Each row is scanned from left to right. The lo-

cation of the first foreground pixel is then stored as $\boldsymbol{p}_1$. In order to start the iteration, a dummy pixel location $\boldsymbol{p}_0$ is initialized with $\boldsymbol{p}_0 = (x_1 - 1, y_1)$. One iteration $i \rightarrow i + 1$ is performed as follows: the eight neighboring pixels to the pixel location $\boldsymbol{p}_i$ are searched for a foreground pixel. The search is performed counterclockwise around the location $\boldsymbol{p}_i$. The search is started with the pixel that follows the neighboring pixel $\boldsymbol{p}_{i-1}$ in the counterclockwise order. The location of the first foreground pixel is stored as $\boldsymbol{p}_{i+1}$, which completes the iteration step. The iterations are continued until $\boldsymbol{p}_{i+1} = \boldsymbol{p}_1$. An example of the iteration process is shown in Figure 6.5, where the foreground pixels are gray. The first pixel location $\boldsymbol{p}_1$ is dark gray.

6.6.3 Basic Approach to Computing the Likelihood

In this thesis only definitions of the likelihood function $P(\boldsymbol{I}|\boldsymbol{x})$ are implemented, where the likelihood is evaluated based on the silhouette(s) of the model and edge information in the observed image(s). In the following it is assumed that a single camera view is used. The observation $\boldsymbol{I}$ is a 2D intensity image. The contour of the silhouette of the rendered model in the pose $\boldsymbol{x}$ is given as a set $\mathcal{C}$. The following approach to compute the likelihood for the contour of the silhouette is similar to the approach proposed in [26]: First an edge image is generated using an edge filter. This image could be, for example, a gradient intensity image $g(\boldsymbol{p}) = \|\boldsymbol{\nabla I}\|$. The image $g(\boldsymbol{p})$ is then smoothed with a Gaussian filter mask and normalized such that the resulting image $G(\boldsymbol{p})$ has the property: $G(\boldsymbol{p}) \in [0, 1]$. The likelihood is then computed based on the following definition:

$$P(\boldsymbol{I}|\boldsymbol{x}) = \exp\left(-\frac{1}{2\gamma^2}\frac{1}{|\mathcal{C}|}\sum_{\boldsymbol{p}\in\mathcal{C}}(1 - G(\boldsymbol{p}))^2\right). \tag{6.21}$$

The definition is similar to [26, 57]. Since $G(\boldsymbol{p})$ is a smoothed edge image, the value $d(\boldsymbol{p}) \equiv 1 - G(\boldsymbol{p})$ of a pixel location $\boldsymbol{p}$ can be interpreted as distance measure to edge features. The value $d(\boldsymbol{p}) \approx 0$ indicates that $\boldsymbol{p}$ is close to intensive edges. The value $d(\boldsymbol{p}) = 1$ indicates that no edges are in the vicinity of $\boldsymbol{p}$. The sum over the contour $\mathcal{C}$ is basically a sum of squared errors. The value $\gamma = 0.15$ turned out to be reasonable in the experiments. A slightly smoothed image $G(\boldsymbol{p})$ is shown on the top right of Figure 6.6. An extension to multiple camera views is straight forward:

$$P(\boldsymbol{I}|\boldsymbol{x}) = \exp -\frac{1}{2\gamma^2}\sum_i \frac{1}{|\mathcal{C}_i|}\sum_{p\in\mathcal{C}_i}(1 - G_i(\boldsymbol{p}))^2, \tag{6.22}$$

where $\mathcal{C}_i$ and G_i are the corresponding contour sets and edge images of each view i.

However, a major problem concerning the particle filtering is that likelihood function $P(I|x)$ needs to be smooth and needs to provide some useful information about how well the hypothesis $\boldsymbol{x}$ matches with the observation. This is explained by the following example. A likelihood function $P(I|x)$ could be defined such that it is usually $P(I|\boldsymbol{x}) = 0$ and becomes $P(I|\boldsymbol{x}) = 1$ only if the hypothesis $\boldsymbol{x}$ produces a perfect match of the contour of the model with the edges in the observed image. This likelihood function $P(I|\boldsymbol{x})$ would provide for the most hypotheses $\boldsymbol{x}$ no useful information about a good guess of $\boldsymbol{x}$ and one would have to evaluate a huge number of hypotheses until one finally evaluates some hypothesis $\boldsymbol{x}$ with $P(I|\boldsymbol{x}) = 1$. In this thesis only a small set of particles is used to achieve real time performance. Therefore, it is required that the likelihood function $P(\boldsymbol{I}|\boldsymbol{x})$ is smooth.

This can be accomplished for the likelihood function in Equation (6.21) by using a edge image $G(\boldsymbol{p})$ which is smoothed with a large Gaussian kernel and then normalized. Such edge images are

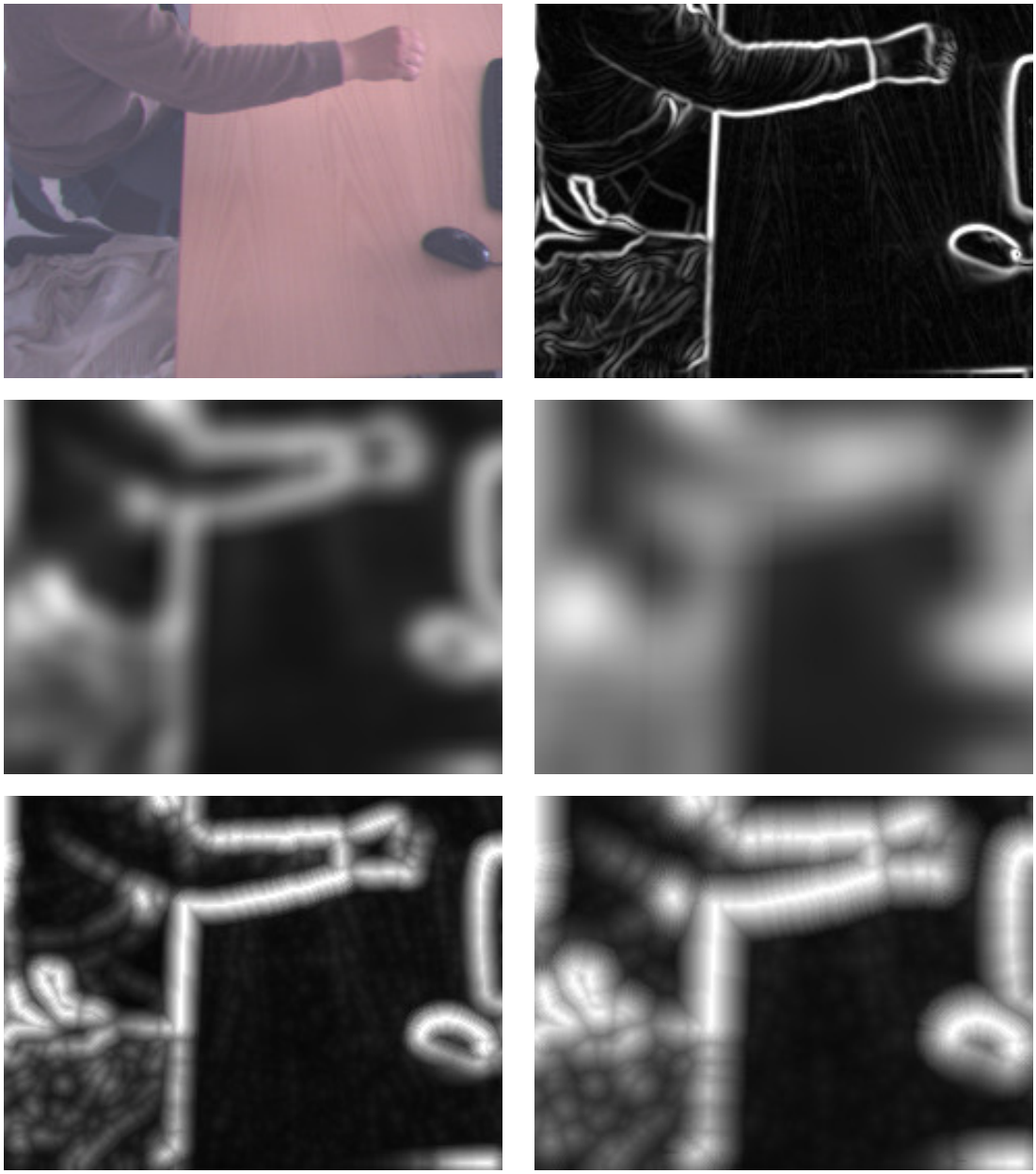


Figure 6.6: Smoothed Images and Distance Images. The first row shows on the left a slightly smoothed and normalized gradient intensity image $g(\boldsymbol{p})$ of the original image shown on the left. The original image is 320×240 pixels, but only 240×160 pixels are shown. The second row shows smoothed (and normalized) versions of $g(\boldsymbol{p})$ which are smoothed with a Gaussian kernel of size $\sigma = 7$ (left) and $\sigma = 14$ (right). The third row shows the distance image $\tilde{G}(\boldsymbol{p})$ (see Equation (6.23)) of $g(\boldsymbol{p})$ for $L = 10$ (left) and $L = 20$ (right).

shown in the second row of Figure 6.6 for the kernel sizes $\sigma = 7$ and $\sigma = 14$. One disadvantage of the smoothing with a large Gaussian kernel is that the value $G(\boldsymbol{p})$ can be greater at a location, which is not on an edge, than for locations which are on an edge. This can be seen at some locations on the forearm in the image on the right in the second row of Figure 6.6. An improvement regarding this concern is

the use of a 'distance' image $\tilde{G}(\boldsymbol{p})$ generated from $g(\cdot)$ by using the definition:

$$\tilde{G}(\boldsymbol{p}) = \max_{\boldsymbol{q} \,:\, \|\boldsymbol{q}-\boldsymbol{p}\|<L} \{g(\boldsymbol{q}) \cdot (1 - \|\boldsymbol{q}-\boldsymbol{p}\|/L)\}\,, \tag{6.23}$$

where L is the maximal considered distance. Examples for $L = 10$ and $L = 20$ are given in the last row of Figure 6.6.

Basically, the definition Equation (6.23) is rather an inverse distance image. More precisely, the truncated distance transform [76] of $g(\cdot)$ is $d(\boldsymbol{p}) \equiv L(1 - \tilde{G}(\boldsymbol{p}))$, where $g(\cdot)$ is assumed to be a binary image. However, an image as $\tilde{G}(\boldsymbol{p})$ is addressed in the following as *distance image*. It is worthwhile to note that it makes sense to replace the edge image $G(\boldsymbol{p})$ in Equation (6.21) through a distance image $\tilde{G}(\boldsymbol{p})$ as defined above. The image $\tilde{G}(\boldsymbol{p})$ is normalized, since $g(\cdot)$ is assumed to be normalized. Obviously, the definition (Equation (6.23)) of a distance image makes also sense when $g(\cdot)$ is not a binary image, see Figure 6.6.

The sum in Equation (6.21) with $\tilde{G}(\boldsymbol{p})$ in place of $G(\boldsymbol{p})$ is basically a Chamfer distance [123] as it is used for Chamfer matching. The similarity becomes obvious if one replaces $1 - \tilde{G}(\boldsymbol{p})$ by $d(\boldsymbol{p}) \equiv 1 - \tilde{G}(\boldsymbol{p})$. A problem of Chamfer matching [76] is that shapes often fit well to background clutter. For example, in Figure 6.6 (second and third row), the contour of the forearm can match quite well against the clutter on the bottom left of each image. This is especially the case for the image smoothed with a large kernel ($\sigma = 14$) and the distance image with $L = 20$. Thus, one would get a high likelihood $P(\boldsymbol{I}|\boldsymbol{x})$ for an obvious wrong hypothesis $\boldsymbol{x}$ with the forearm on the background clutter. Another more extreme example is the following: For an input image $\boldsymbol{I}(\boldsymbol{p})$ that is filled with horizontal lines in a small distance, the value $\tilde{G}(\boldsymbol{p})$ is $\tilde{G}(\boldsymbol{p}) \approx 1$ at each location $\boldsymbol{p}$. As a consequence, also a vertical line will provide a good match. Therefore, the following approach to matching makes also use of the orientations of the image gradients.

6.6.4 Contour Matching for a Robust Likelihood Measure

To handle the problems (as discussed in the section above) in a better way, another matching approach is proposed in the following. The idea is to perform a matching between the hypothesized contour and the observed edges such that the matching preserves the hypothesized shape of the contour (see also Figure 6.8). The approach moves the contour pixels along the normals of the contour into the direction of larger gradients. The matching process takes account of the orientations of the gradients. Since the hypothesized contour is supposed to be matched against a very similar shape in the observed image, the gradients on the edges in the observed image should be parallel/antiparallel to the normals of the contour. The approach to matching minimizes an energy formula (Equation (6.26)). In this sense, the algorithm is similar to the Snake algorithm [124]. However, the following approach preserves (to some degree) the shape of the hypothesized contour and makes use of the orientations of the gradients. Since the optimization process is local, it is required that the gradient image is smooth in order to allow a matching over several pixels. Therefore distance images are generated for different gradient orientations.

Generation of Distance Images for Different Gradient Orientations. The generation of the distance images is described in the following briefly. The border handling is neglected in all descriptions which follow. Of course, in an implementation the access to pixels out of the image domain has to be handled appropriately.

The input image is a small (320×240) color image $I^c(x, y)$ which is a down scaled and slightly smoothed version of a camera image. The image gradients $\boldsymbol{g}(x, y)$ are then computed for each pixel location (x, y) by selecting the largest gradient in the color channels:

$$\boldsymbol{g}^c(x, y) = (I^c(x + 1, y) - I^c(x + 1, y), I^c(x, y + 1) - I^c(x, y))^\top, \text{ (where } c = 1, 2, 3)$$
$$\boldsymbol{g}(x, y) = \boldsymbol{g}^c(x, y), \text{ where } c = \arg\max_i \boldsymbol{g}^i(x, y).$$

In the experiments, this approach resulted sometimes in more significant gradients in the observed contours than by generating first a gray valued image and computing the gradients based on this.

To be insensitive against changes of the image brightness, the gradients $\boldsymbol{g}(x, y)$ are normalized. However, it was often the case in the experiments that some pixels produced very large gradients, e. g., at the edges of the table. As a consequence, the gradients on the arm contour became sometimes very

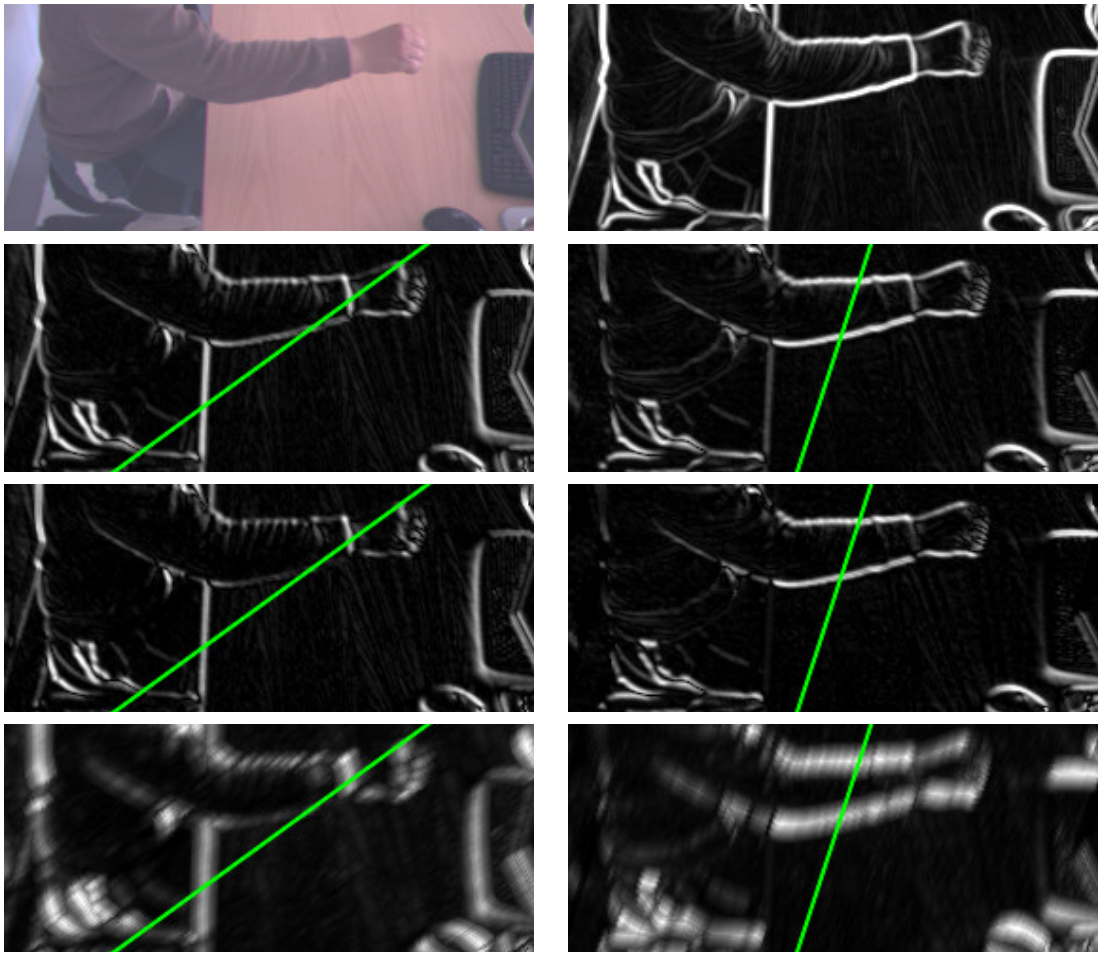


Figure 6.7: Gradients and Distance Images for Different Directions. The figure shows some examples of the images $g_i(\boldsymbol{p})$ (Equation (6.24)) and $\tilde{g}(\boldsymbol{p})$ (Equation (6.25)). The gradient intensity image $\|\hat{\boldsymbol{g}}(\boldsymbol{p})\|$ is shown in the first row. The second row shows $g_i(\boldsymbol{p})$ for $\alpha = 1$, the third row shows $g_i(\boldsymbol{p})$ for $\alpha = 2$, and the last row shows $\tilde{g}_i(\boldsymbol{p})$ for $\alpha = 2$ and $L = 10$. The green lines indicate the directions $\hat{\boldsymbol{n}}_i$ which are used to generate the images, i.e. $\hat{\boldsymbol{n}}_2 = (\sin(2\pi/10), \cos(2\pi/10))$ on the left and $\hat{\boldsymbol{n}}_4 = (\sin(4\pi/10), \cos(4\pi/10))$ on the right.

weak. Therefore, the gradients are scaled by $1/n$ (and truncated):

$$\hat{\boldsymbol{g}}(x,y) = \begin{cases} \boldsymbol{g}(x,y)/n & : \|\boldsymbol{g}(x,y)\| < n \\ \boldsymbol{g}(x,y)/\|\boldsymbol{g}(x,y)\| & : \text{else} \end{cases},$$

where the normalization value n is chosen smaller than the maximal norm of all gradients $\boldsymbol{g}(x,y)$. More precisely, the value n is chosen as large as possible but small enough such that $\|\hat{\boldsymbol{g}}(x,y)\| = 1$ is true for at least N pixel locations. In the implementation the value n is determined on the basis of a histogram of the image $g'(x,y) = \|\boldsymbol{g}(x,y)\|$. (In the experiments a value $N = 1000$ is used.)

The distance images are generated on the basis of $\hat{\boldsymbol{g}}(x,y)$ for different search and gradient directions. Of course, only a small set of distance images can be generated due to real time constraints. The number $D = 10$ of search directions $\hat{\boldsymbol{n}}_i = (\sin(i\pi/D), \cos(i\pi/D)), i = 0,\ldots,D-1$ appeared to be appropriate. For each direction $i = 0,\ldots,D-1$ a gradient intensity image

$$g_i(x,y) = \|\hat{\boldsymbol{g}}(x,y)\| \cdot \left(|\langle \hat{\boldsymbol{g}}(x,y) \,|\, \hat{\boldsymbol{n}}_i \rangle| / \|\hat{\boldsymbol{g}}(x,y)\|\right)^{\alpha} \tag{6.24}$$

is computed first. For a value $\alpha = 1$, the pixel value at each location of the image is the absolute value of the gradient $\hat{\boldsymbol{g}}$ projected on $\hat{\boldsymbol{n}}_i$. However, a gradient with an angle of 60° between gradient and the direction $\hat{\boldsymbol{n}}_i$ still contributes with a factor 0.5. To reduce the contribution of gradients which are rather orthogonal than parallel/antiparallel to $\hat{\boldsymbol{n}}_i$, a value $\alpha = 2$ was chosen in the experiments. Finally, the distance images for the direction $i = 0,\ldots,D-1$ are computed by

$$\tilde{g}_i(\boldsymbol{p}) = \max_{l=-L,\ldots,+L} g_i(\boldsymbol{p} + l\boldsymbol{n}_i) \cdot (1 - |l|/L) \tag{6.25}$$

where a value of $L = 10$ turned out to be reasonable. The distance search is performed in Equation (6.25) only in the directions $\pm\hat{\boldsymbol{n}}_i$. The pixel locations $\boldsymbol{p} + l\boldsymbol{n}_i$ are rounded to integer addresses. It is worth to note, that the value $g_i(\boldsymbol{p} + l\boldsymbol{n}_i) \cdot (1 - l/L)$ combines the distance $(1 - l/L)$ and the gradient intensity $g_i(\cdot)$ (similar to the definition in Equation (6.23)). Examples for the images $g_i(\boldsymbol{p})$ and $\tilde{g}_i(\boldsymbol{p})$ are given in Figure 6.7 for different orientations $\hat{\boldsymbol{n}}$ and values of α.

Objective Function. The objective function that is used for the matching depends on a set of shifts s^i along the normals $\boldsymbol{n}^i$ of some key points. Therefore, first a set of K key points $\boldsymbol{p}^k$ of the arm contour $(\boldsymbol{p}_1,\ldots,\boldsymbol{p}_N)$ are selected. (The contour generation is discussed in Section 6.6.2.) The key points are selected by taking approximately every 7th pixel of the contour. For each key point a normal $\boldsymbol{n}^k$ with $\|\boldsymbol{n}^k\| = 1$ is computed. The first row in Figure 6.8 shows some examples for the key points and their normals. Then, normals $\boldsymbol{n}_i$ are computed for each contour pixel $\boldsymbol{p}_i$ by interpolating the normals of the adjacent key points. In the same way, the shifts s_i of each contour pixel along the normals $\boldsymbol{n}_i$ are defined through linear interpolation between the shifts s^k of the adjacent key points. This means that the shifts s_i are basically a function $s_i(s^1,\ldots,s^K)$. For brevity this is neglected in the following. As mentioned above, a set of distance images, $\tilde{g}_1(\boldsymbol{p}),\ldots,\tilde{g}_D(\boldsymbol{p})$, for different search directions is created. For each normal $\boldsymbol{n}_i$ the index b_i is chosen such that the image $\tilde{g}_{b_i}(\boldsymbol{p})$ is most appropriate for the search along $\boldsymbol{n}_i$ or $-\boldsymbol{n}_i$. The shifted pixel location $\boldsymbol{p}_i(s_i)$ of a contour pixel $\boldsymbol{p}_i$ is defined as $\boldsymbol{p}_i(s_i) = \boldsymbol{p}_i + s_i\boldsymbol{n}_i$. The objective of the matching process is to estimate the shifts $(\hat{s}^1,\ldots,\hat{s}^K)$ which

maximize the objective function:

$$\begin{aligned} E(s^1,\dots,s^n) &= E_{\text{gradients}} \;-\; E_{\text{shift}} \;-\; E_{\text{distortion}} \\ &= \alpha_1\left(\sum_{i=1}^{N}\frac{\tilde{g}_{b_i}(\boldsymbol{p}_i(s_i))}{N}\right) \;-\; \alpha_2\left(\sum_{k=1}^{K}\frac{b(s^k)}{K}\right) \;-\; \alpha_3\left(\sum_{k=1}^{K}\frac{c(s^k,s^{k+1})}{K}\right). \end{aligned} \tag{6.26}$$

The term $E_{\text{gradients}}$ forces the contour to larger gradient intensities. The terms E_{shift} and $E_{\text{distortion}}$ penalize the overall shift of the contour and the distortion of the initial shape of the contour. A straight forward choice for the penalties $b(s^k)$ and $c(s^k, s^{k+1})$ are, for example, $b(s^k) = |s^k|$ and $c(s^k, s^{k+1}) = |s^{k+1} - s^k|$. However, the weighting functions $\alpha_1(x), \dots, \alpha_3(x)$ and penalties $b(s^k)$ and $c(s^k, s^{k+1})$ have been tuned on the basis of some test cases. In the experiments the following definitions were used:

$$\begin{aligned} \alpha_1(x) = 7.5x^2,\ \alpha_2(x) = x,\ \alpha_3(x) = 4x, \\ b(s) = |s|^2, \\ c(s^k, s^{k+1}) = 0.11|q|^2 + |q_\perp|^{2.5}, \end{aligned}$$

where q and $q_\perp$ are the displacement of the key point $\boldsymbol{p}^{i+1}$ to $\boldsymbol{p}^i$ longitudinal and transverse to the contour, i.e.

$$\begin{aligned} q &\equiv \left\langle \boldsymbol{m}^i(s^i, s^i+1) \,\middle|\, \boldsymbol{m}_0^i \right\rangle / \|\boldsymbol{m}_0\| - \|\boldsymbol{m}_0\|, \\ q_\perp &\equiv \langle \boldsymbol{m}^i(s^i, s^i+1) \,|\, \boldsymbol{m}_\perp^i \rangle / \|\boldsymbol{m}_\perp\|, \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{m}_0^i &\equiv \boldsymbol{p}^{i+1} - \boldsymbol{p}^i, \\ \boldsymbol{m}_\perp^i &\equiv \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right) \boldsymbol{m}_0^i, \\ \boldsymbol{m}^i &\equiv \boldsymbol{p}^{i+1}(s^{i+1}) + \boldsymbol{p}^{i+1}(s^i). \end{aligned}$$

Matching. The estimation of the parameters $\hat{s}^1, \dots, \hat{s}^K$ which maximize Equation (6.26) is performed iteratively. The iteration starts with the first guess $(\hat{s}^1, \dots, \hat{s}^K) = (0, \dots, 0)$. In each iteration the values $\hat{s}^1, \dots, \hat{s}^K$ are refined by a value Δs iteratively. This inner iteration is: for $k = 1, \dots, K$ the value $\hat{s}^k$ is replaced by a value $s^k \in \{\hat{s}^k - \Delta s, \hat{s}^k, \hat{s}^k + \Delta s\}$ which increases the value $E(\hat{s}^1, \dots, s^k, \dots, \hat{s}^K)$ (if possible). The outer iteration is performed with different step sizes Δs. In the implementation 20 iterations are used with the step sizes $\Delta s = 1, 0.5, 1, 0.25, 1, 0.5, 1, 0.125, \dots$, where the small step sizes are introduced to prevent the algorithm from getting stuck. Obviously, the matching of a contour requires a large number of evaluations of the objective function $E(s^1, \dots, s^K)$. However, it is worthwhile to note that only very few summands in Equation (6.26) change if one component s^i is altered. This means: if $E(\hat{s}^1, \dots, \hat{s}^K)$ is known, then $E(\hat{s}^1, \dots, s^k, \dots, \hat{s}^K)$ can be computed on the basis of $E(\hat{s}^1, \dots, \hat{s}^K)$ by adding and subtracting the terms that are affected by s^k. The matching together with contour pre-processing (as finding the contour points $\boldsymbol{p}_1, \dots, \boldsymbol{p}_N$ in the rendered image, and selecting the key point etc.) requires about 1.24ms on a single 2.4GHz Core 2 processor. This does not include the generation of the distance images. However, the distance images need to be generated only once per image frame, whereas the matching has to be performed several (e.g., 400) times per frame.

Computing the Likelihood. The likelihood $P(\boldsymbol{I}|\boldsymbol{x})$ is evaluated based on the matching of the contour. One possibility would be to define the likelihood based on the value $E(\hat{s}^1, \ldots, \hat{s}^K)$ after the matching. However, the likelihood is computed in the thesis in a similar way as given by the definition in Equation (6.26). Equation (6.26) is basically

$$P(\boldsymbol{I}|\boldsymbol{x}) = \exp -\frac{1}{2\gamma^2}\frac{1}{N}\sum_{i=1}^{N}(1 - G(\boldsymbol{p}_i))^2, \tag{6.27}$$

where $G(\boldsymbol{p}_i) \in [0, 1]$ was the gradient intensity of a contour pixel. However, in the matching based approach, the contour pixels are matched against the gradients. In the following it is assumed that a matching $(s^1, \ldots, s^K)$ has been estimated as discussed above. This defines then also the shifts $s_1, \ldots, s_N$ for each contour pixel $\boldsymbol{p}_i$. Each contour pixel $\boldsymbol{p}_i$ is matched to a pixel location $\boldsymbol{p}_i(s_i) = \boldsymbol{p}_i + s_i\boldsymbol{n}_i$ with a usually more intensive gradient. The shifts s_i, the normals $\boldsymbol{n}_i$, and the indices b_i are defined as discussed for the objective function. A first approach is to replace $G(\boldsymbol{p}_i)$ in Equation (6.27) by the following definition:

$$G^-(\boldsymbol{p}_i) = g_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i) \cdot 1/(1 + (s_i/\beta_1)^{\beta_2}),$$

where $g_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i)$ is the gradient intensity at the matched pixel location $\boldsymbol{p}_i + \boldsymbol{n}_i s_i$ in the distance image for gradients in the direction $\pm\hat{\boldsymbol{n}}_{b_i}$. The term $1/(1+(s_i/\beta_1)^{\beta_2})$ makes the value $G^-(\boldsymbol{p}_i)$ smaller (and thus the likelihood) when the distance s_i of the match becomes greater. The values β_i were chosen in the experiments as $\beta_1 = 7.5 \pm 0.5$ and $\beta_2 = 2$. For a match $s_i = 0$, the term $1/(1 + (s_i/\beta_1)^{\beta_2})$ becomes 1. The term converges to zero for $s_i \to \infty$. For $s_i > \beta_1$ the term is already $1/2$.

However, in the experiments the gradient intensity $g_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i)$ was often very small even when the point $\boldsymbol{p}_i + \boldsymbol{n}_i s_i$ was on an edge that had the right orientation and visually appeared to be a strong edge. Therefore the gradient intensity $g_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i)$ is boosted in the following way:

$$G^+(\boldsymbol{p}_i) = g^+_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i) \cdot 1/(1 + (s_i/\beta_1)^{\beta_2}),$$

where

$$g^+_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i) = \begin{cases} x & : z/x \leq 1 \\ \min\{1, x \cdot (z/x)^2\} & : z/x > 1 \end{cases}, \text{ with}$$

$$x \equiv g_{b_i}(\boldsymbol{p}_i + \boldsymbol{n}_i s_i),$$

$$z \equiv \max_{k \in \{-10, +10\}} g_{b_i}(\boldsymbol{p}_i + s_i\boldsymbol{n}_i + k\boldsymbol{n}_i).$$

This means that gradient intensity x is increased by a factor $(z/x)^2$ when the gradients of the same orientation have a smaller value z if one goes 10 pixels from matched pixel location $\boldsymbol{p}_i + \boldsymbol{n}_i s_i$ in the direction $+\boldsymbol{n}_i$ or $-\boldsymbol{n}_i$. Again the values -10, $+10$ and the value 2 in the exponent are chosen to achieve the desired effect for some initial contours on some test images. Figure 6.8 shows some matchings and the resulting values $G^-(\boldsymbol{p}_i)$ and $G^+(\boldsymbol{p}_i)$. The right column shows a matching for a totally wrong hypothesis. The contour is distorted more than usual, since there are only weak gradients on the initial contour. However, on the left side of the upper arm the values $G^-(\boldsymbol{p})$ and $G^+(\boldsymbol{p}_i)$ are still significantly smaller than 1 due to the huge shifts of the matching. The extension to multiple views is analog to the extension of Equation (6.21) to Equation (6.22) and requires that the generation and matching of a contour is performed once for each view.

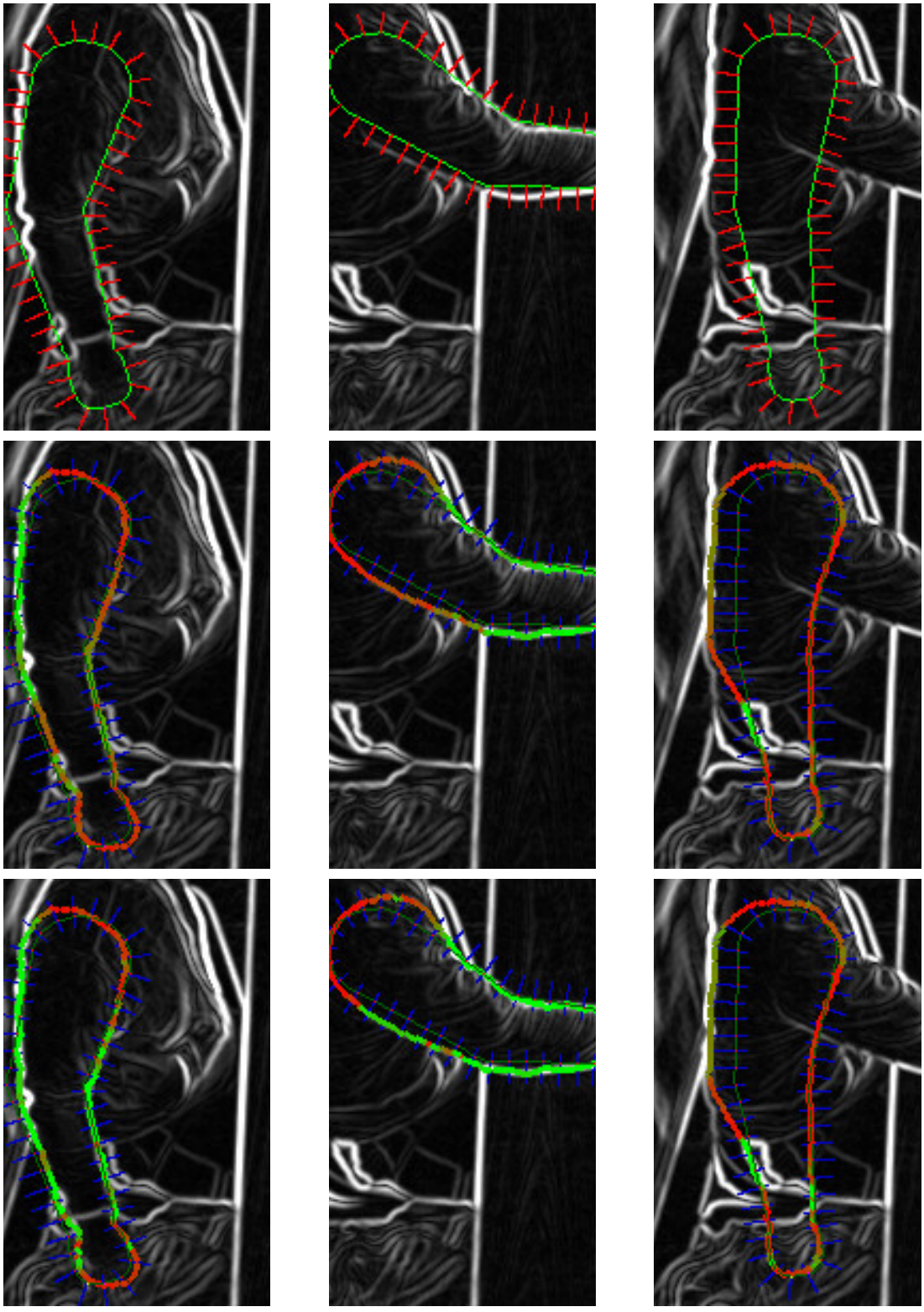


Figure 6.8: Matching. The first row shows the hypothesized contours with the normals (scaled to 10 pixel) at the locations of the selected key points. The second and third row show the matched contours. The colors of the matched contour pixels $\boldsymbol{p}_i$ indicate the value $G^-(\boldsymbol{p}_i) \in [0,1]$ (in the second row) and $G^+(\boldsymbol{p}_i) \in [0,1]$ (in the third row). Light green indicates a value 1 and light red indicates a value 0.

6.7 Experiments

The tracking and recognition experiments are performed in a table-top scenario (Figure 6.1). The parametric actions are manipulative actions or pointing gestures/actions. Manipulative actions are pouring a glass with water into a teapot or taking an object and placing it somewhere else (e.g., on a box). The pointing actions are pointing to table-top locations, which are meant to communicate a specific object or a table-top location. Such an action could be used to specify the meaning of a dialog: "Take this object here, and place it at that location." The parametric actions are represented through PHMMs (Section 6.4.1), which are trained on demonstrations of the actions. The PHMMs have a large number of states (e.g., 80) in order to be an accurate predictive model for tracking. The tracking is performed in the space of possible actions (Section 6.3). The tracking approach makes use of particle filtering (Section 6.4). The number of particles used is 400, which is a tradeoff between speed and tracking quality. The framework runs in real time (8fps) when a single view is used (Section 6.5), but still multiple views can be recorded simultaneously. The tracking is an analysis-by-synthesis approach. The scene model and body model are discussed in Section 6.6. The hypotheses of the body/arm pose are generated from the action models. The analysis is based on the silhouette cues (Section 6.6.4). The actions start and end usually in a base pose, where the arm is resting beside the person. For the tracking in a simple scenario, where only one action is used, the action grammar is a "loop". This allows one to track several consecutive performances of the action. The diffusion of the temporal progress τ of an action (Section 6.4.4) allows the tracking even when the person stays arbitrarily long in the base pose. More complex grammars are used in the cases of recognizing different actions (Section 6.7.2) or tracking complex actions (Section 6.7.3). The complex action of the experiments (Section 6.7.3) is composed from different (basic) parametric actions. The complex action is interesting in two senses. First, different compositions of the basic actions are tracked and recognized. Second, one of the basic actions makes no use of a base pose, it begins and ends at arbitrary table-top locations.

Driving questions of the experiments are: Is it possible to recognize a single action? Is it possible to distinguish different actions? Are the action parameters (i.e. the table-top locations) estimated correctly? Is the body pose estimated accurately? How well is the body pose estimated from a single view? Concerning the last question, usually the model matches the image of the person in the view which is used during the tracking. However, the depth component of the pose in this view can still be wrong (see Section 6.7.1.2). Therefore, the performed actions are recorded for the analysis (Section 6.7.1) from multiple views (Figure 6.1) or with ground truth information.

The main focus in the experiments on single actions (Section 6.7.1) is the accuracy of the estimated body pose and the recognition of action parameters. In the experiments with multiple actions (Sections 6.7.2 and 6.7.3), the main focus is on the recognition of the currently tracked action.

6.7.1 Single Actions

The parametric actions considered in this section are very similar in the following sense: The actions are parameterized by a single table-top location (the action in Section 6.7.1.4 is a slight exception). The table-top locations are represented through 2D vectors $\boldsymbol{\theta} = (u, v)$ which are normalized ($u, v \in [0, 1]$). The parameters u and v correspond to the directions of the x-axis and y-axis in Figure 6.1, respectively. The actions start and end in the same base pose. The parameters of the action models are discussed in Section 6.4.1. The temporal progress of an action is denoted by τ which is normalized ($\tau \in [0, 1]$).

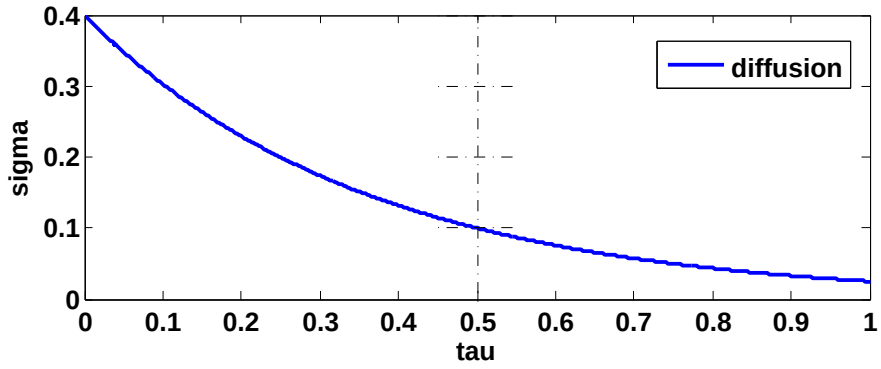


Figure 6.9: The function $\sigma(\tau) = 0.4 \cdot \exp(-2\log_e(1/4) \cdot \tau)$ is used to change the diffusion of the parameters (u, v) during the propagation step.

A parameter $\tau = 1$ corresponds to the end of an action. The grammar of each action in this section is a "loop". This means a particle $\boldsymbol{\omega} = (\boldsymbol{\theta}, \tau)$, where the progress τ of the action is greater than 1, is reset, i.e. τ is replaced by $\tau' = \tau - 1$. This allows one to track multiple repetitions of the action. The propagation and diffusion (Section 6.4.4) of the parameter τ enables the tracking of actions with different progression rates. The propagation is adjusted to the frame rate of the tracking. The diffusion allows the person to stay arbitrarily long in the base pose. The diffusion of the action parameters (Section 6.4.4) u and v depends on the progress τ of the action as shown in Figure 6.9. For $\tau \approx 0$ the diffusion is very large, which reflects the assumption that the action parameters are not known when the first part of an action had been tracked. The diffusion becomes smaller with the progress of the action.

6.7.1.1 Pointing Actions with Ground Truth Data

Figure 6.10 shows the tracking of pointing actions. The settings are slightly different from those which are always used in the following. The image sequence is recorded with a high frame rate (50fps) from a single view, and can be tracked only offline. In the base pose, the arm is hanging down beside the person. The camera view is chosen such that the base pose is visible. The camera distortion is undone before the tracking, and the body pose is captured in addition with the Vicon system for evaluation.

The purpose of this trial is to investigate the accuracy of the estimated body pose when using a high frame rate. The pose estimates are compared in Figure 6.11 with the ground truth data acquired with the Vicon system. The pose estimates are performed by selecting the hypothesis $\boldsymbol{x}^n$ which explains the observation best (Equation (6.16)). As a consequence, the estimates are not very continuous over time. Therefore, the estimates are median filtered. Note, the directions of the x-, y-, and z-axis correspond with the system shown Figure 6.1, but the origin is different. Table 6.1 shows that the pose estimation is very accurate when a high frame rate is used. The root-mean-square error over three consecutively tracked performances is less than 1.8cm for each x-, y-, z- component of the hand locations.

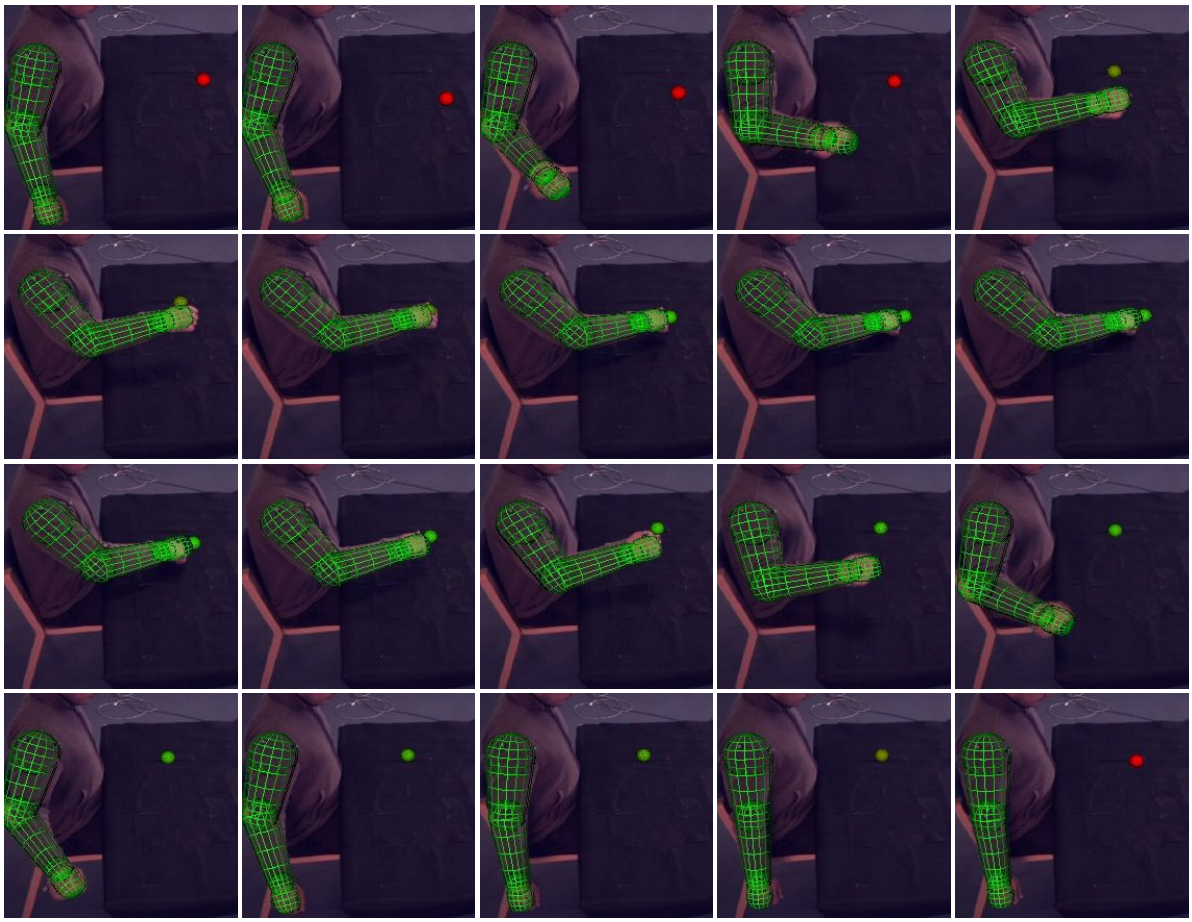


Figure 6.10: Pose Estimation through Tracking in Action Space. In the rows 2–5, a whole pointing action (approaching and withdrawing motion) is shown with the recovered arm pose superimposed. The sequence has about 110 frames. Approximately, every 6th frame is shown. The estimated action parameters are indicated by the tiny ball on the table: the color of the ball reflects the uncertainty of the indicated location (given through the deviation of the action parameters u and v): a red dot indicates a large uncertainty, a green dot a low one.

Table 6.1: Hand Location Error. The table shows the average, root-mean-square, and maximal errors in cm between the recovered hand location and the ground truth location over three consecutively tracked performances (see Figure 6.11). The first three rows show the errors when the estimated hand location is median filtered.

	x [cm]	y [cm]	z [cm]
median: mean	0.47	0.91	0.82
median: RMS	0.89	1.79	1.59
median: max	4.67	9.48	6.35
estimate: mean	2.89	3.98	3.64
estimate: RMSE	3.44	5.45	4.76
estimate: max	6.82	20.11	9.12

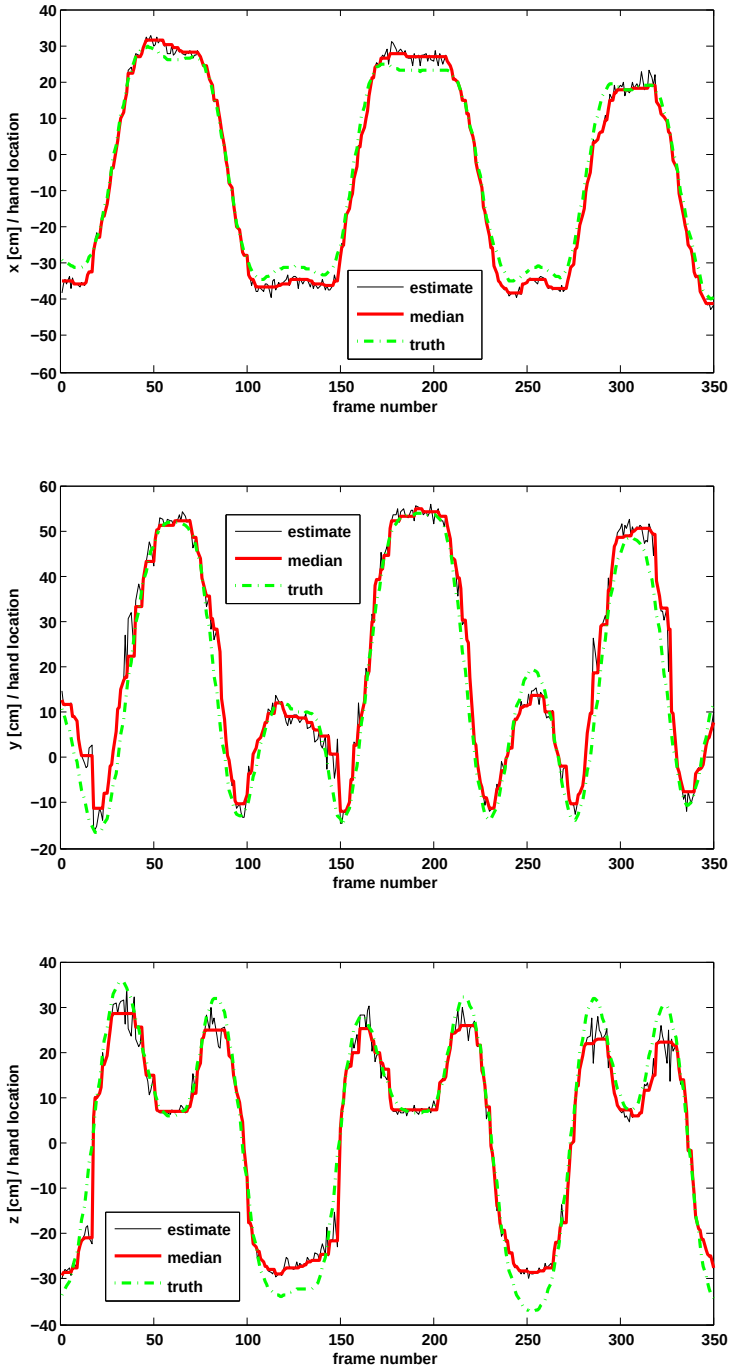


Figure 6.11: Body Pose Estimation. The three plots compare the estimated hand locations (red) to the true locations (green) over three consecutively performed pointing actions. The locations (red) are the median filtered values of the pose estimates through tracking in action space (black). The ground truth locations are given through marker-based tracking.

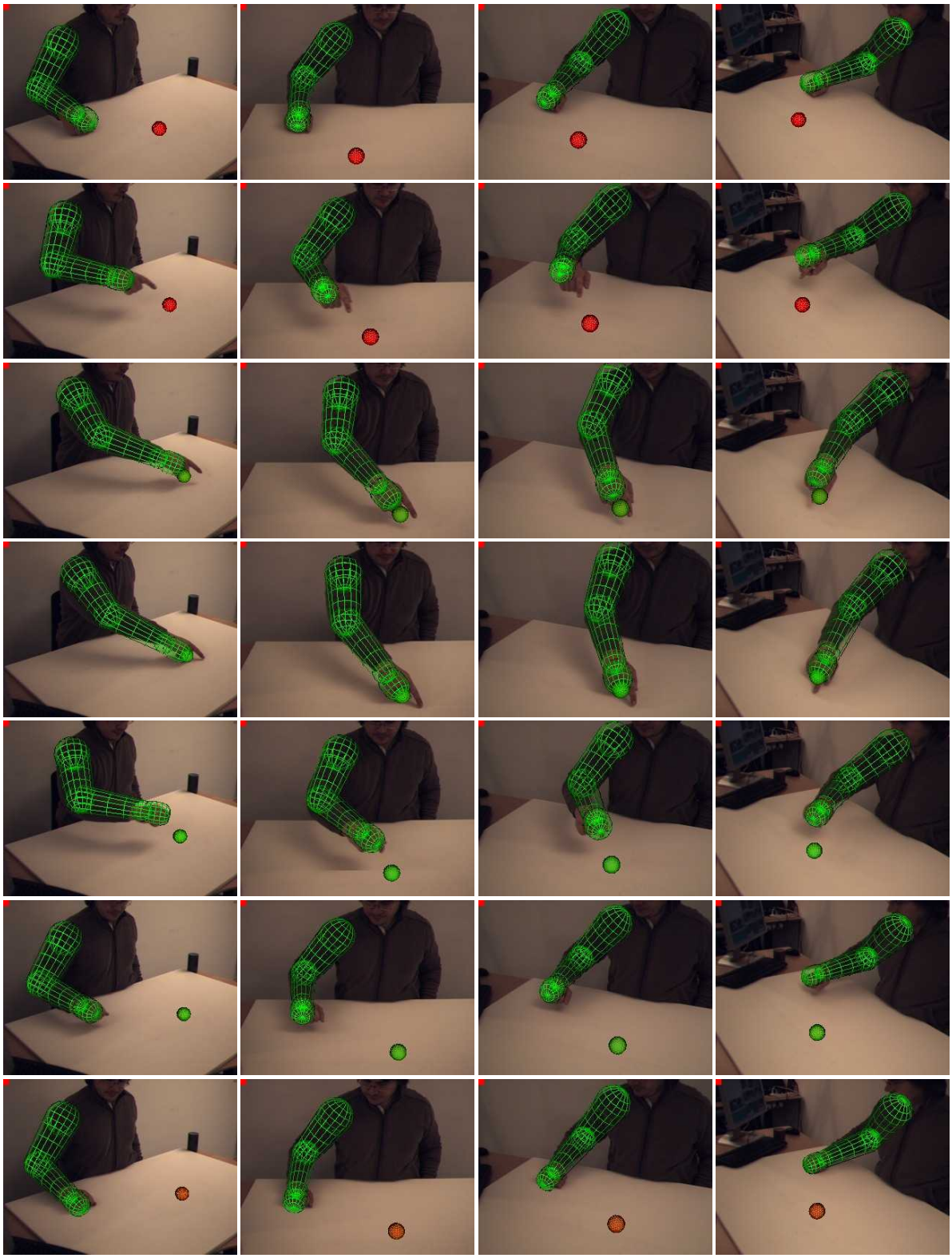


Figure 6.12: Pointing Trial 1: Tracked by 4 Cameras. Each row is showing the same frame of the cameras 1, 2, 3, and 4. The trial has 19 frames. The tracking is performed on the basis of all camera views.

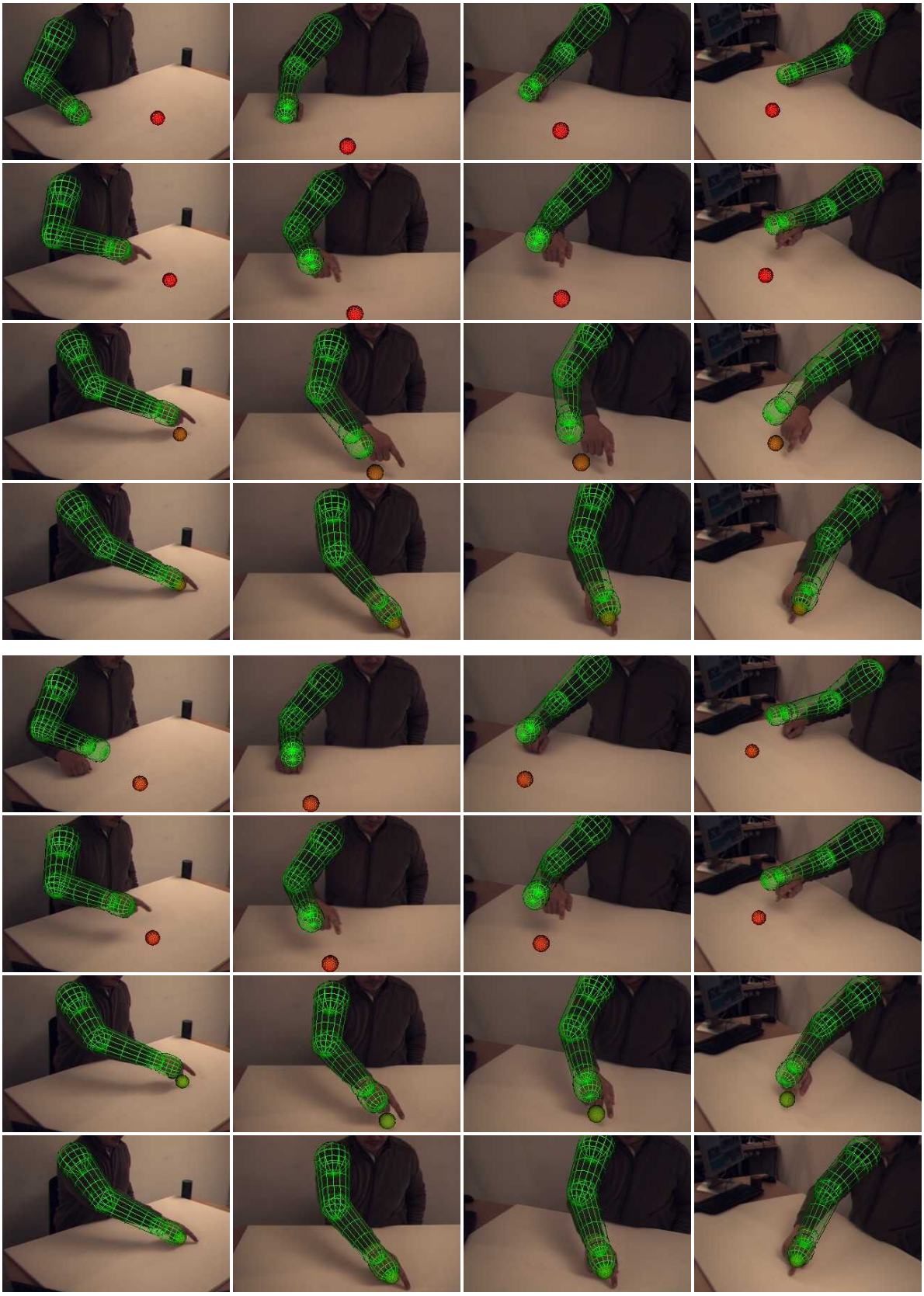


Figure 6.13: Pointing Trial 1: Tracked by Camera 1 or 2. The first part of the pointing trial is shown twice. In the first 4 rows, camera 1 is used for tracking. In the last 4 rows, camera 2 is used.

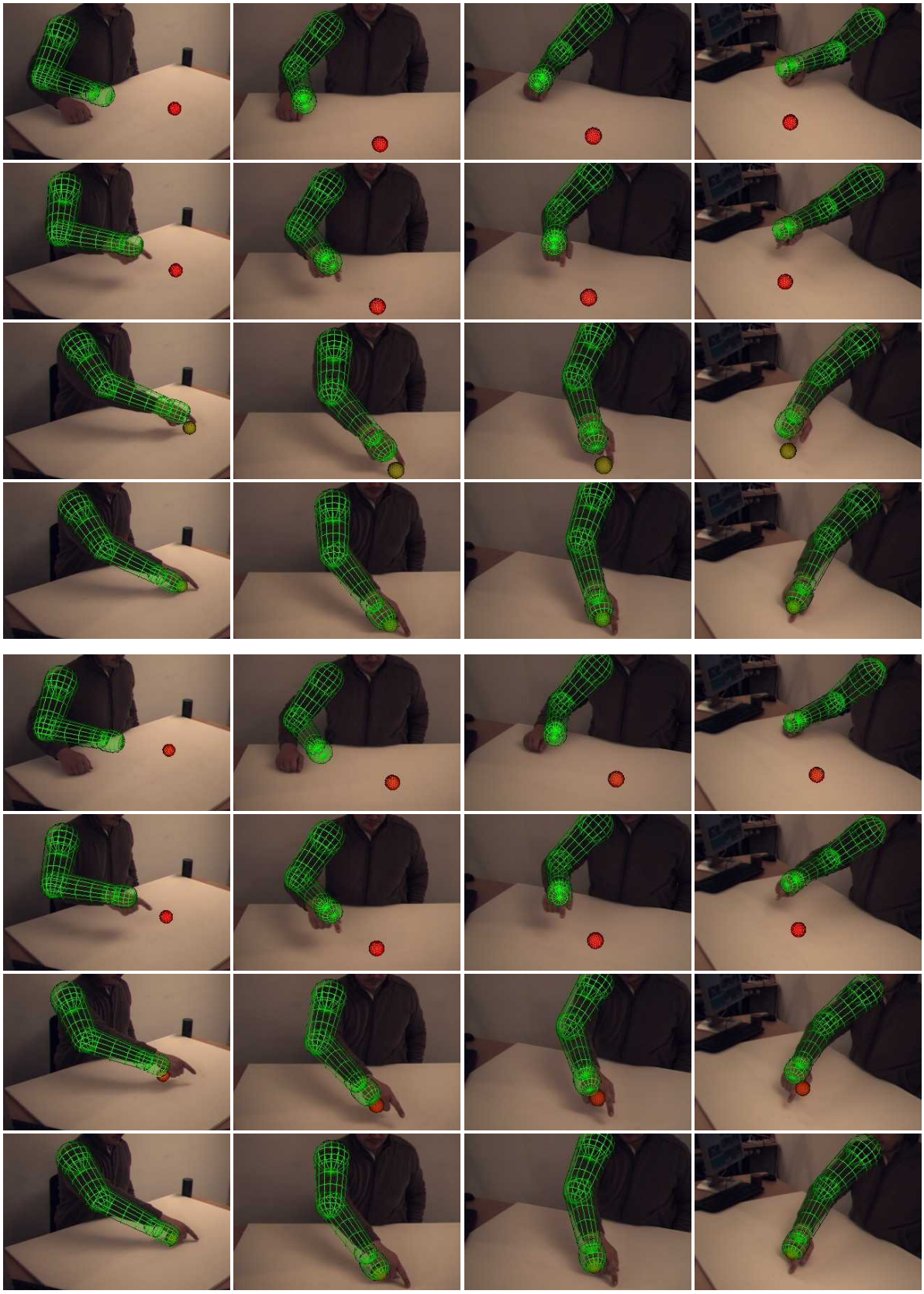


Figure 6.14: Pointing Trial 1: Tracked by Camera 3 or 4. The first part of the pointing trial is shown twice. In the first 4 rows, camera 3 is used for tracking. In the last 4 rows, camera 4 is used.

6.7.1.2 Pointing with Multiple Views

Figures 6.12, 6.13, and 6.14 show the tracking of a single performance of a pointing action. The pointing action is recorded with 8fps. The whole performance of the action has 19 frames. The whole performance is shown in Figure 6.12, where the action is tracked based an all four camera views. The camera locations are shown in Figure 6.1.

The pose estimation is performed on the basis of the method (Equation (6.18)) described in Section 6.4.5. The estimated pose is shown in each image of the views (see Figure 6.12). An estimate of the target location is given by the means $\bar{u}$ and $\bar{v}$ (Equation (6.14)) of the action parameters u and v. The color of the ball correlates with the sum of the deviations $\tilde{u}$ and $\tilde{v}$ (Equation (6.14)) of the action parameters. The color red indicates high deviations and green small deviations. In this way, the color becomes green when the target location has been reached by the hand (this is explained below). The methods of making pose estimates and showing the target location are used in all following experiments.

The Figures 6.13 and 6.14 show the first part of the pointing action (Figure 6.12), where only one of the cameras 1, 2, 3, and 4 are used for tracking. In the view that is used for tracking, the pose estimate seems to be always very accurate. However, in the other views one can see that the estimation of the depth can be inaccurate. This is especially the case in the first frames of the actions. The estimated pose becomes relatively accurate when the target location is reached by the hand. In principle, the problem of estimating the depth is not very surprising, since the estimate is based on the matching of the model silhouette and the silhouette of the person's arm. However, the motion of the arm can provide additional information about the depth.

Figure 6.16 shows the deviations and means of the x-, y-, z-coordinates of the hypothesized hand locations. The standard deviations and means are computed similarly as for the action parameters by Equation (6.14), but with the pose hypotheses $\boldsymbol{x}^n$ in place of $\boldsymbol{\theta}^n$. The deviations are shown in Figure 6.16 for each frame of the whole performance (19 frames), the frame 10 (4th row in Figure 6.12) is a frame where the hand rests at the target location. When the camera 2 or camera 3 are used for tracking, the deviations are in the middle of the performance especially high for the x-coordinate which is basically the depth component in these camera views. For the cameras 1 and 4 the deviations are high for the y-coordinate which corresponds again more or less to the depth. When all cameras are used for the tracking, the deviations are smaller. Note, the deviations of the y-coordinate when using camera 4 are particularly high at the beginning of the performance. As one can see in Figure 6.14, the depth is estimated inaccurately in the first few frames when camera 4 is used for tracking.

It is worthwhile to note that the deviations of the pose hypotheses can be large when the hand is in the base pose but also when the hand is at the target. This is different from the deviations of the action parameters u and v, see Figure 6.17. The deviations of these parameters become significantly smaller when the hand is close to the target (frame 10). An explanation is that only the base pose does not depend on the action parameters. As a consequence, the actions parameters can be arbitrarily chosen in the base pose but not when the action is performed. Note, the diffusion (Figure 6.9) of the action parameters depends on the progress τ of the action but not on the frame number. Therefore, it seems to be reasonable to detect that a target pose has been reached by the means of the deviations of the action parameters. If one considers the "deviation" $\widetilde{uv}$ (calculated by Equation (6.15)) in Figure 6.17, where all cameras are used for tracking, then one can see that $\widetilde{uv}$ is minimal in frame 10. By choosing frame 10, one would select the right pointed to target location, as one can see in Figure 6.15. However,

this detection scheme is insufficient when a single camera is used for tracking. When the tracking is based only on one camera, one would select by this detection scheme the frames 16, 12, 12, and 12 for the cameras 1, 2, 3, and 4, respectively. This would result in lage errors of the detected target locations, especially for camera 1 (the error of the target location in the frame 16 is about 9cm). A better detection scheme is proposed in the following:

Advanced Detection Scheme. After the deviation $\widetilde{uv}$ is below some threshold (e.g., $\widetilde{uv}_{\text{thres}} = 0.2$), the target location is selected when the velocity of the estimated hand location crosses the value 0m/s^2.

The velocity of the hand is shown in Figure 6.16 in black. By this advanced detection scheme, one would select the frames 11, 11, 11, and 10 for the cameras 1, 2, 3, and 4, and the frame 10 when all cameras are used. The errors of the detected target locations are then less than 5cm. Another trial of a tracked pointing action is shown in Figure 6.18. Error plots are given in Figure 6.19. When the advanced detection scheme is used while tracking with camera 2, the error of the detected target location is 2cm.

The Figures 6.15 and 6.19 show the accuracy of the pose estimates and the target locations (given by the action parameters), for both trials. It is worthwhile to note that the target locations are unknown in advance. The estimates of the target locations becomes very accurate when the hand is at the target location. When the hand is withdrawn from the target location, the estimates become again inaccurate. An explanation is that the diffusion of the action parameters (see Figure 6.9) is nonzero for any $\tau \in [0, 1]$. The errors of the estimated arm poses can become quite large (up to 18cm), when a single camera is used and when the arm is in motion. However, when the hand rests at the target location the errors are always less than 6cm and usually about 2cm. An explanation for the large errors are the low frame rate (8fps). In the experiment above (with a high frame rate) the errors are smaller.

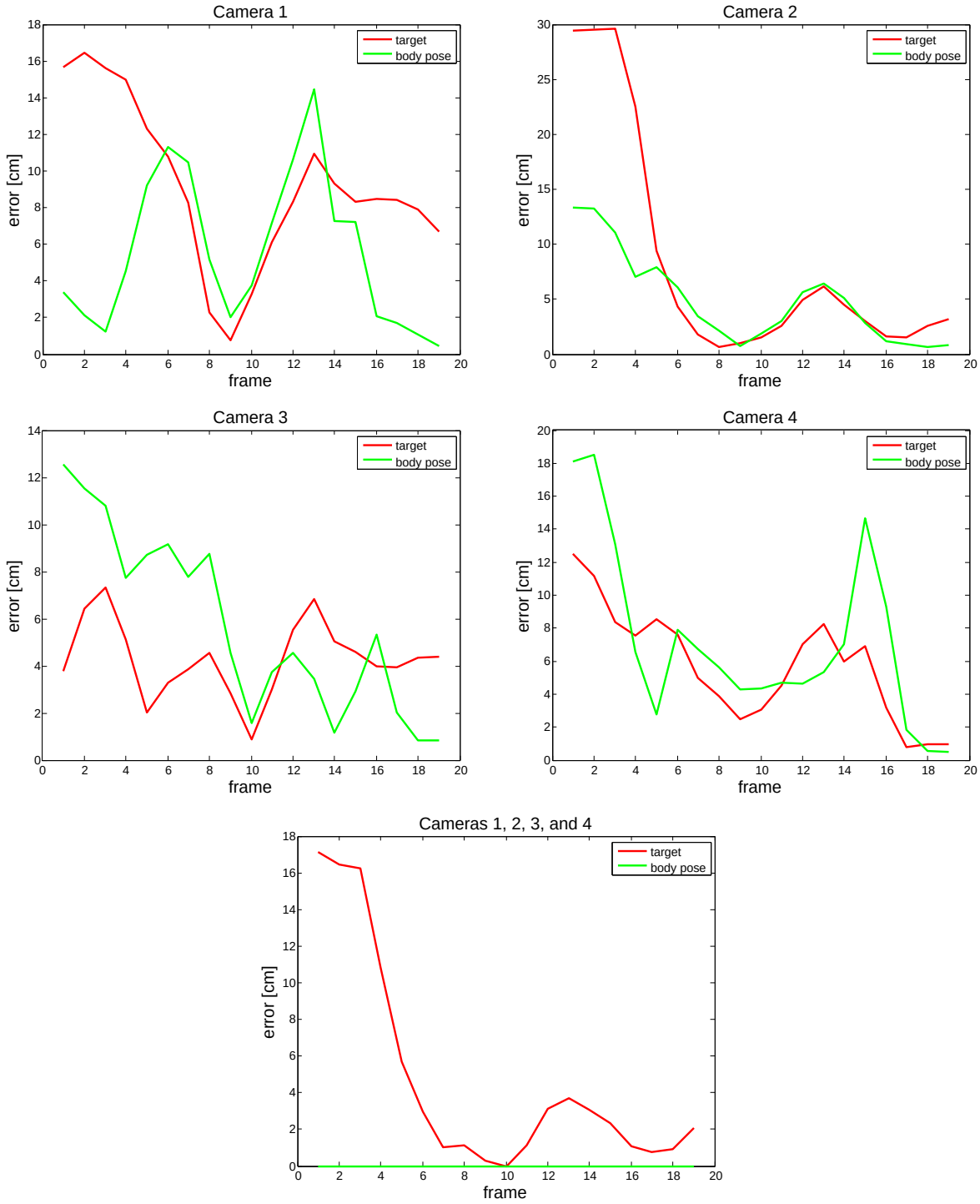


Figure 6.15: Pointing Trial 1: Target and Pose Estimation. The plots show the errors of the estimated target location and the body pose in cm. The error of the body pose is calculated as root-mean-square error (RMSE). As a reference, the estimates of the tracker are used where all cameras are used for the tracking. The target location is selected manually in frame 10 on the basis of the tracking when all cameras are used.

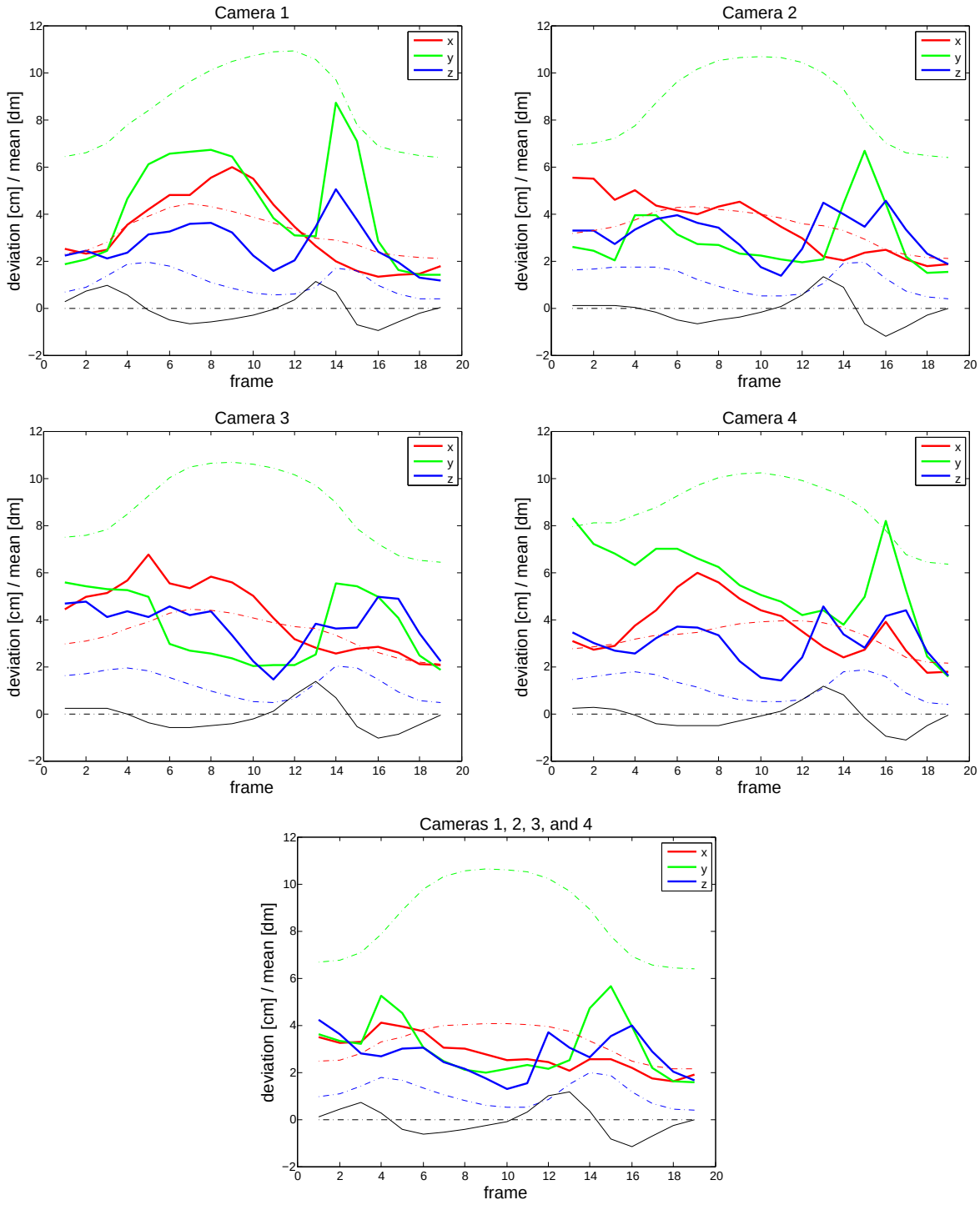


Figure 6.16: Pointing Trial 1: Deviations and Means of Hand Locations. The solid lines in the plots show the deviations of the x-, y-, and z-coordinate of the hypothesized hand locations for each frame in centimeters (cm). The means of the hand locations are shown by the dashed lines in decimeters (dm). The solid black line shows the velocity of the hand location in dm/s^2.

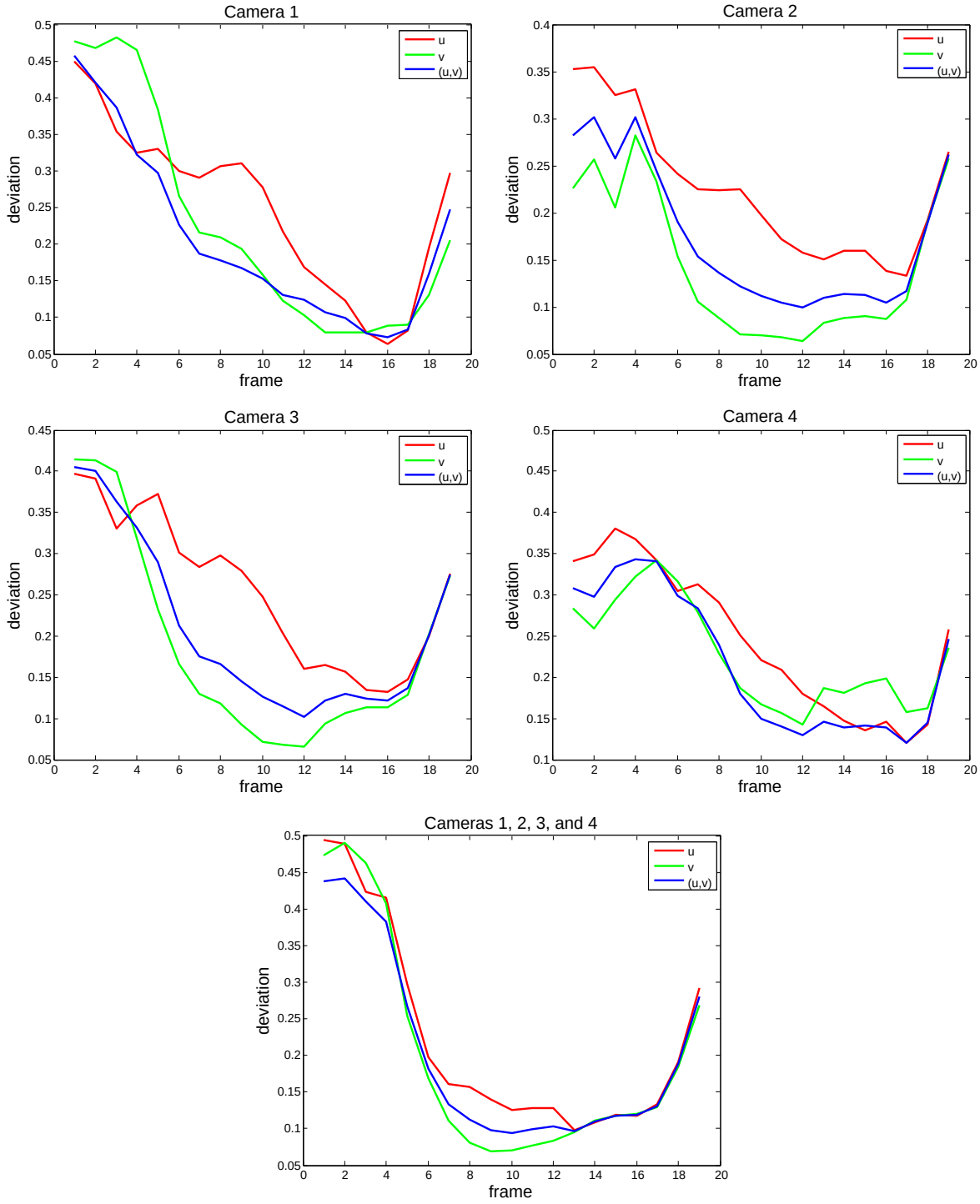


Figure 6.17: Pointing Trial 1: Deviations of Action Parameters. The plots show the deviations $\tilde{u}$, $\tilde{v}$, and $\widetilde{uv}$. The first four plots show the deviations when a single camera is used for the tracking of pointing trial 1. The last plot shows the case when all four cameras views are used for tracking.

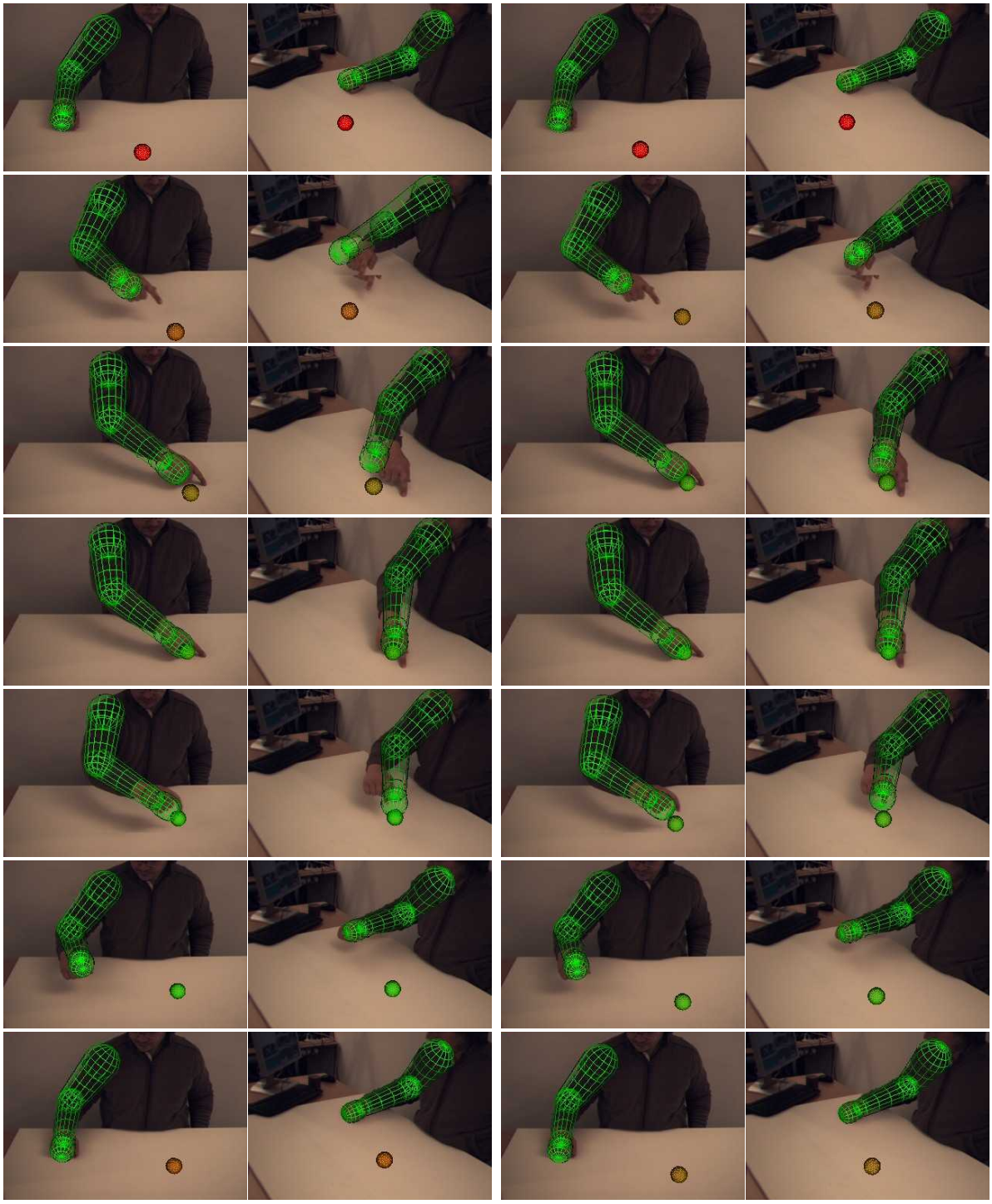


Figure 6.18: Pointing Trial 2. On the left, the camera views of the cameras 2 and 4 are shown, where only camera 2 is used for tracking. On the right, the same camera views are shown, where all cameras (1, 2, 3, and 4) are used for tracking.

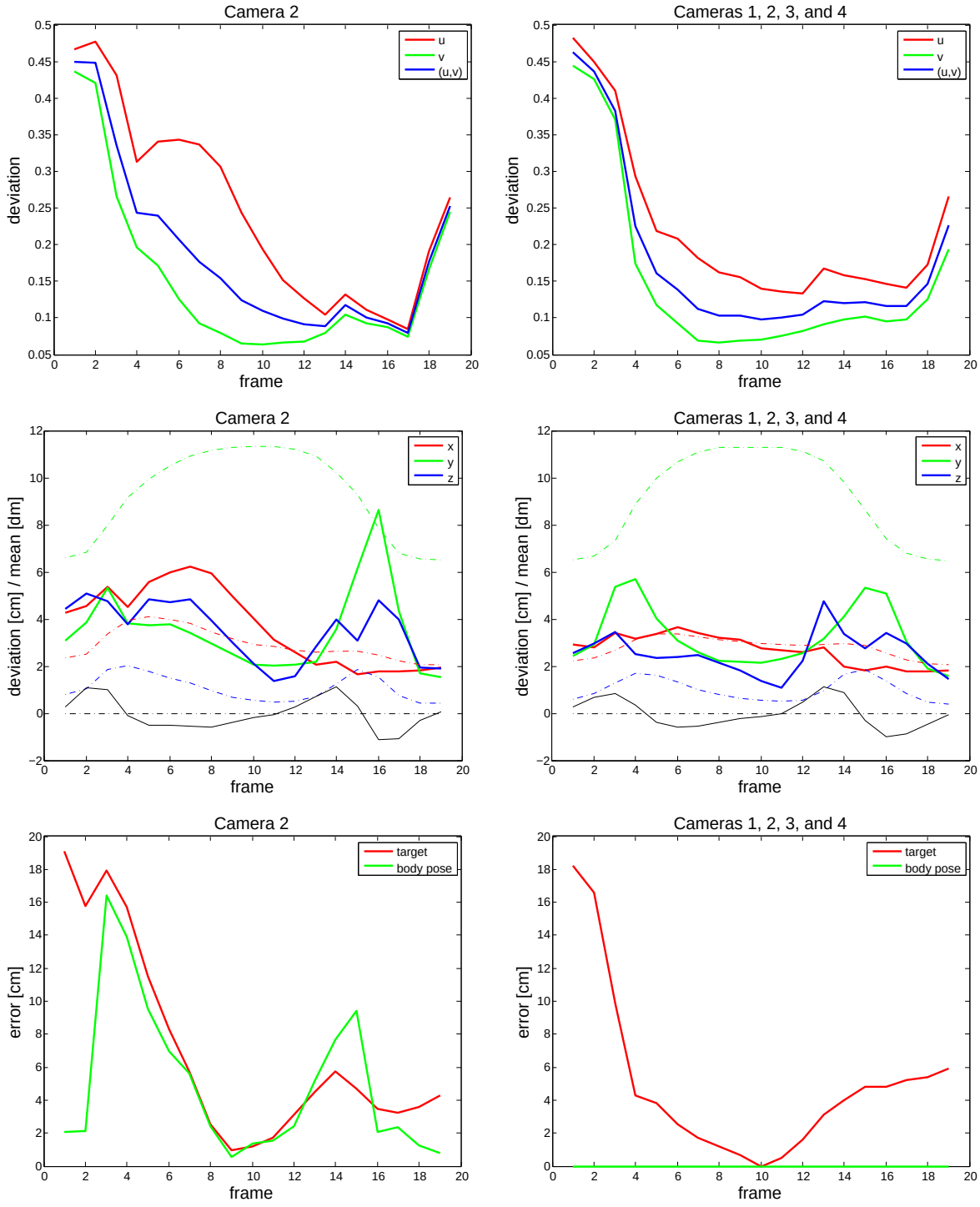


Figure 6.19: Pointing Trial 2.

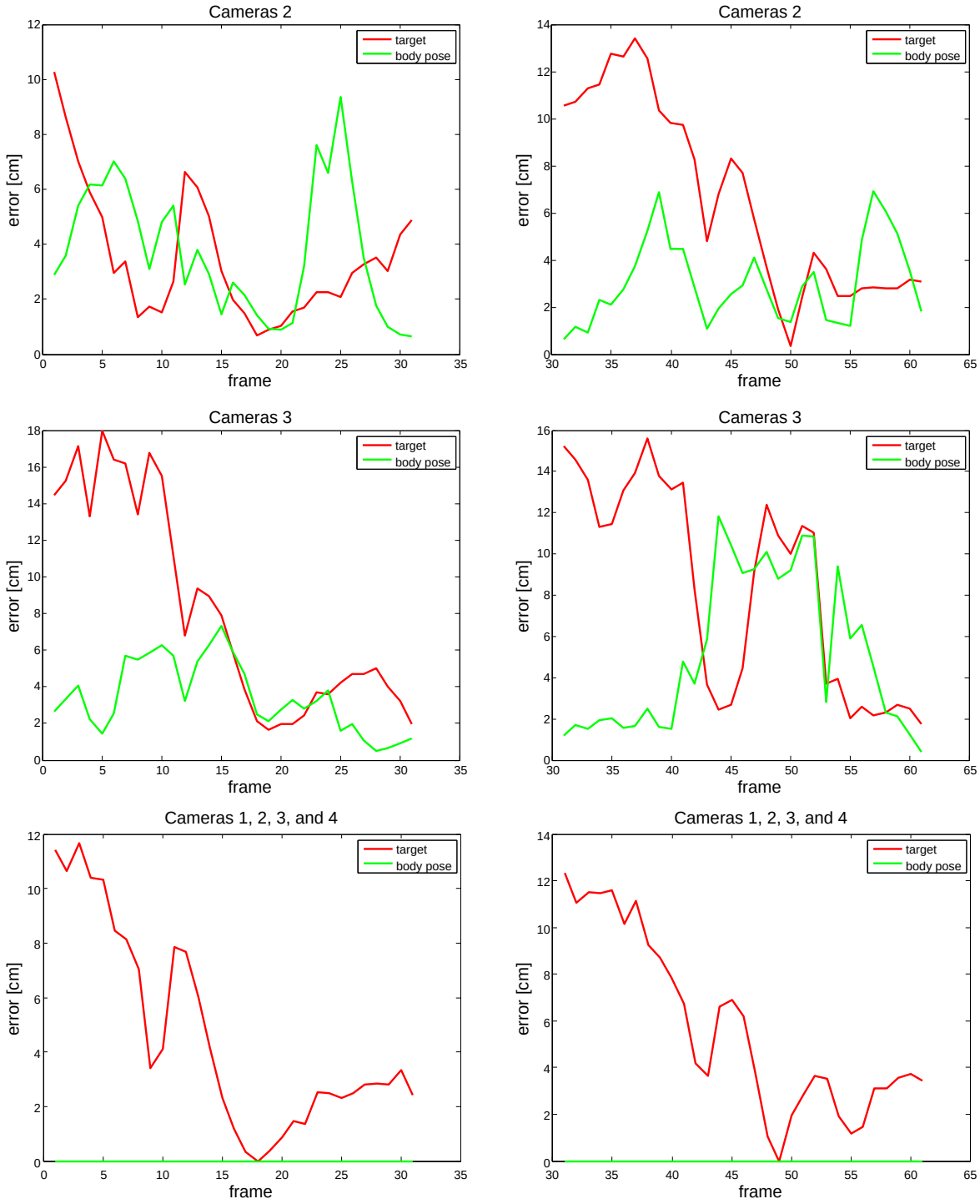


Figure 6.20: Pointing Far: Target and Pose Estimation. The figure shows the plots for the two performances of the pointing action shown in Figure 6.21. The plots on the left and right correspond to the first and second performance of the pointing action. The plots show the errors over both performances. As a consequence, the plots on the right are a continuation of the plots on the left (concerning the arm pose). Note, the target location of the second performance is different from the target of the first performance. Hence, the errors of the target locations produce gaps between the plots on the left and the right.

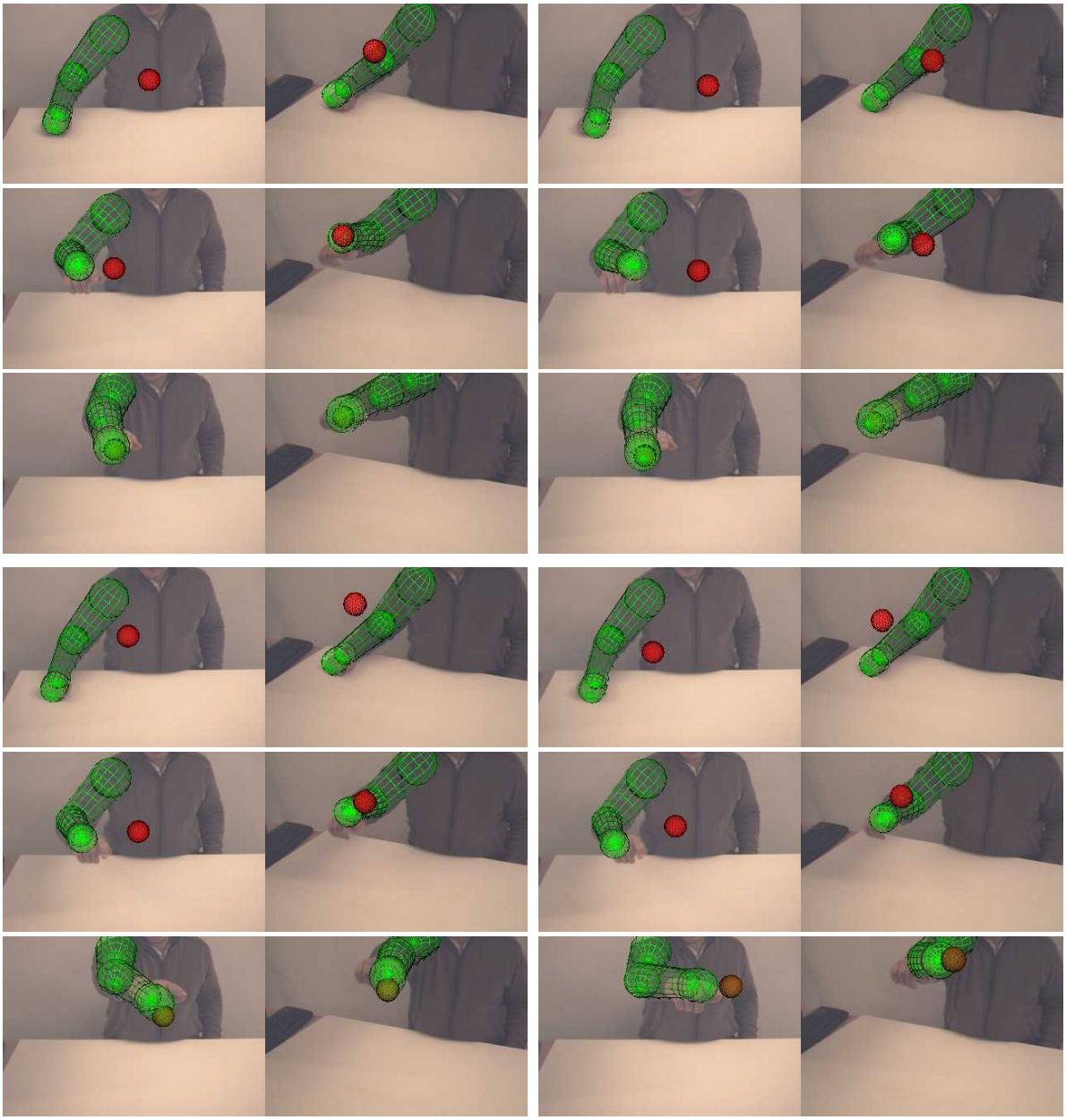


Figure 6.21: Pointing Far. The first three rows show the first part of a pointing action. The following three rows show the first part of another pointing action which is performed in continuation to the first pointing action. On both sides the camera views of the camera 2 and 3 are shown. On the left, the tracking is based on camera 2. On the right, the tracking is based on camera 3.

6.7.1.3 Another Pointing Action

The pointing action which is considered in this section is one in which the person points in the direction of an object which is supposed to be located far away from the person. This parametric pointing action is parameterized again in the form $\boldsymbol{\theta} = (u, v)$, but here the parameters correspond to a location which is on a plane that is located in front of the person. The settings for the tracking and the particle propagation are the same as for the pointing action of the section above. The grammar is again a simple "loop". Figure 6.21 shows the first parts of two consecutively performed pointing actions. Only the images of the cameras 2 and 3 are shown. The plots of the target and pose estimation (as provided in the sections above) are given in Figure 6.20. The tracking of these pointing actions on the basis of the cameras 2 or 3 are particularly interesting, since the person is pointing almost in the directions of these cameras. However, the target locations and arm poses are estimated accurately when the hand is close to the target plane (in frame 18 and 49). An exception is the second performance of the pointing action (frame 49) when tracked on the basis of camera 4. The person points in this frame (see the last image of the last row of Figure 6.21) directly in the direction of the camera, which complicates the pose estimation dramatically.

6.7.1.4 Pouring Water into a Teapot

The settings for the tracking are the same of the sections above. But the parametric action is tracked only online with 8fps in the view of Camera 1. The grammar is again a "loop". The action is parameterized by a single table-top location $\boldsymbol{\theta} = (u, v)$. The demonstrations of the parametric action which are used for the training of the action model are performed as follows: an action starts from the base pose that is used for the actions above. The person reaches then for a glass at an arbitrary location $\boldsymbol{\theta}$. Then, the person pours the water into a teapot. Finally, the person places the glass at the same location $\boldsymbol{\theta}$ at the table-top.

Figure 6.22 shows the tracking/recognition results of two consecutive performances of this parametric action. The whole trial has 38 frames, 32 frames are shown in a row-wise order. An interesting aspect is here that the person can place the glass after pouring the water into the teapot at a location which is different from the location from which the glass is taken. The tracker accepts this due to the propagation of the particles, since the diffusion of action parameter $\boldsymbol{\theta}$ is still nonzero when the glass is placed again on the table. The color of the ball (which corresponds to the deviation of the action parameters) becomes light green, when the object is taken from a table-top location or when the object is placed on the table. In both cases, the location given by the estimated parameters of the action are corresponding accurately with the location of the glass.

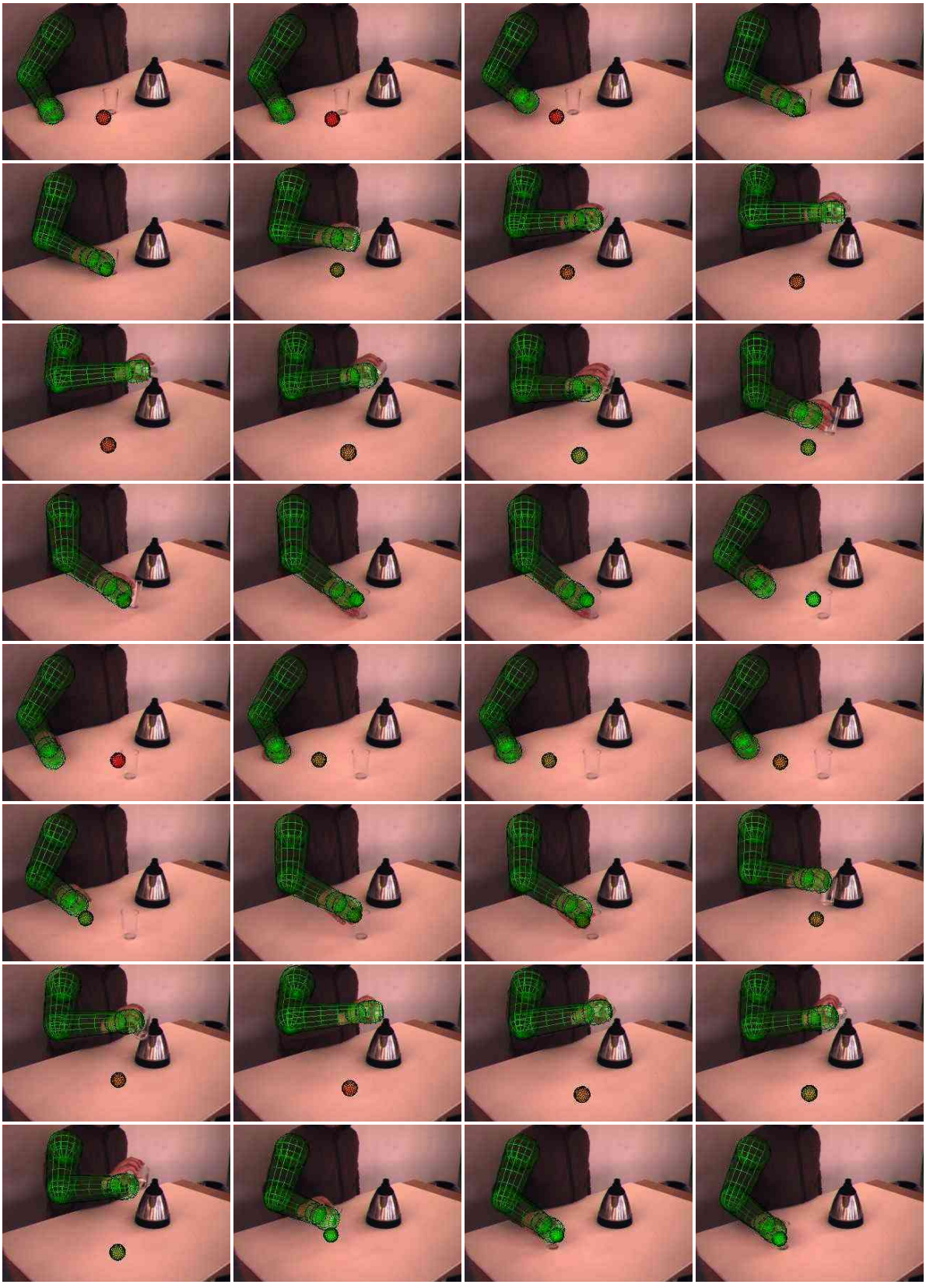


Figure 6.22: Pouring a Glass of Water into a Teapot. Two consecutive performances of the action are tracked on the bases of camera 1. The images of the camera are shown in a row-wise order.

6.7.2 Action Recognition

The settings of the recognition experiment here are similar to those in Section 6.7.1. But here two different actions are considered. The parametric actions are: A, taking an object from the table-top and placing it on a box. B, taking an object from the box and placing it on the table-top. Both actions start and end with the same base pose and are parameterized by the location $\boldsymbol{\theta} = (u, v)$ at the table-top. The location on the box is modeled as noise. The grammar is basically again a "loop", but after a performance of an action any action (A or B) can follow. The grammar is depicted in Figure 6.23 as a state machine. The particle propagation concerning the parameters $\boldsymbol{\theta}$ and τ are basically as discussed in Section 6.7.1. The alterations are: A particle $\boldsymbol{\omega} = (a, \boldsymbol{\theta}, \tau)$, where the progress τ of the action is greater than 1, is reset, i.e. τ is replaced by $\tau - 1$ and the type of the action $a \in \{\mathrm{A}, \mathrm{B}\}$ is chosen randomly. In addition, the action type of a particle $\boldsymbol{\omega} = (a, \boldsymbol{\theta}, \tau)$ is switched with a probability of 20%, when $\tau \in [0.0, 0.1]$. This is necessary to prevent the particles with $\tau \in [0.0, 0.1]$ from corresponding all to a single action (e.g., $a = \mathrm{A}$) before there is any evidence of the action that will be performed.

The tracking is performed in a single camera view (Camera 2) in real time with the same number of particles (400 particles) as in the experiments above. Figure 6.25 shows the tracking of 3 consecutive performances of the actions. The actions are performed in the order: B, A, B. Figure 6.24 shows the posterior probabilities of the actions A and B (Equation (6.8)) for each frame of the 3 consecutive performances. The vertical lines indicate the actual progress of the performed actions (see explanation of Figure 6.24), and allow one to identify the corresponding images in Figure 6.25. The images in Figure 6.25 show in addition the estimates, $\hat{\boldsymbol{\theta}}_{\mathrm{A}} = \mathbb{E}[\boldsymbol{\theta}_t | a = \mathrm{A}]$ and $\hat{\boldsymbol{\theta}}_{\mathrm{B}} = \mathbb{E}[\boldsymbol{\theta}_t | a = \mathrm{B}]$ (Equation (6.10)), of the action parameters of the actions A and B. The corresponding table-top locations are indicated by the colored disc (action A) and the colored ball (action B). The ball is smaller than the disc. The color of the ball/disc indicates again the deviations of the action parameters (as in Section 6.7.1), but here the deviations of the action parameters (Equation (6.11) and Equation (6.14)) are calculated for the particles of each specific action type A and B. The color green indicates small deviations. In addition, the color is set for an action a always to red if $p_t(a) < 0.9$ or $p_t(a, \tau \in [0.25, 0.75]) < 0.5$ (Equation (6.9)).

The mechanism of setting the color provides a simple means to recognize the type of a performed action and its parameters. This is discussed in combination with the 3 consecutive performances shown in Figure 6.25. At first the action B is performed, where the person reaches at first for an object on the box. Over the whole performance of the action B only at one time the ball (corresponding to action B) becomes green, whereas the disc is always red. Hence, the action B is recognized. In addition, the parameters can be recognized by taking the estimate $\hat{\boldsymbol{\theta}}_{\mathrm{B}}$ when the deviations are minimal, i.e. when the ball is very green. In the images 8 and 9 one can see that the estimates correspond with the actual location at which the object is placed on the table. In the following performance of the action A, the disc becomes green when the object is taken. In the images 20 (last image of row 5) and 21, the estimates $\hat{\boldsymbol{\theta}}_{\mathrm{A}}$ correspond accurately with the location, where the object is grasped. As one can see in Figure 6.24, the posterior probability of action A is 1 over the frames following frame 43 (which corresponds to image 20). The last performance of the trial is the action B. In the beginning of the performance the posterior probability indicates rather that action A is tracked. However, when the object on the box is grasped (frame 78 in Figure 6.24) the posterior probability of action B becomes greater than 0.5. The colors of the ball and the disc do not become green before the table-top location (at which the object will be placed) is approached. In the last image of the next-to-last row, the object

is placed, and the ball is green. Hence, the action B is recognized. The estimated location given by the ball corresponds accurately with the actual location at which the object is placed. This is also the case in the previous and the following image. Both of these images restrict the frames when the ball becomes light green. Over the corresponding frames (close to frame 84 given by the vertical green line in Figure 6.24), the posterior probability of action B is 1.

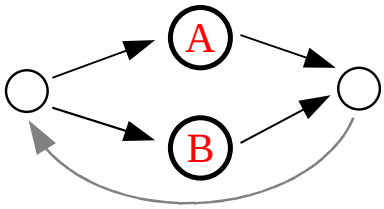


Figure 6.23: Grammar for Recognizing/Tracking two Different Actions.

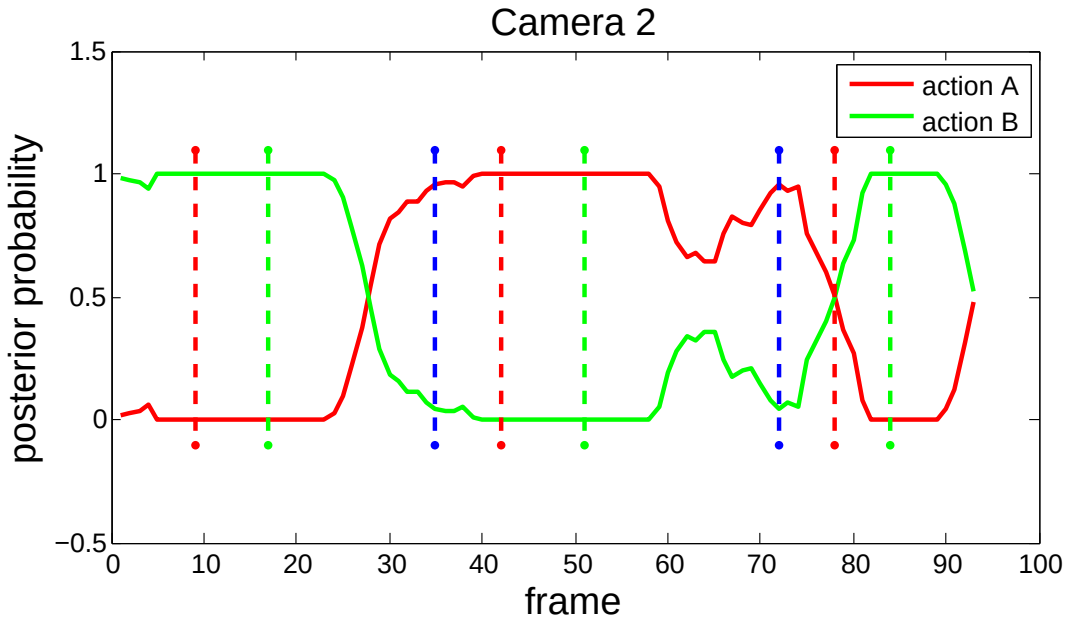


Figure 6.24: Posterior Probabilities of the Actions A and B. The plot shows the posterior probabilities $p_t(a = \mathrm{A})$ and $p_t(a = \mathrm{B})$ of the actions A and B, respectively. The probabilities are calculated for each frame of the tracked sequence shown in Figure 6.24. The blue vertical lines indicate the frame when the person begins with the performance of a new action. The first blue line indicates the beginning of the performance of action A, and the second indicates the beginning of the action B. The red vertical lines indicate the frames when an object is grasped. The green lines indicate that the object is placed.

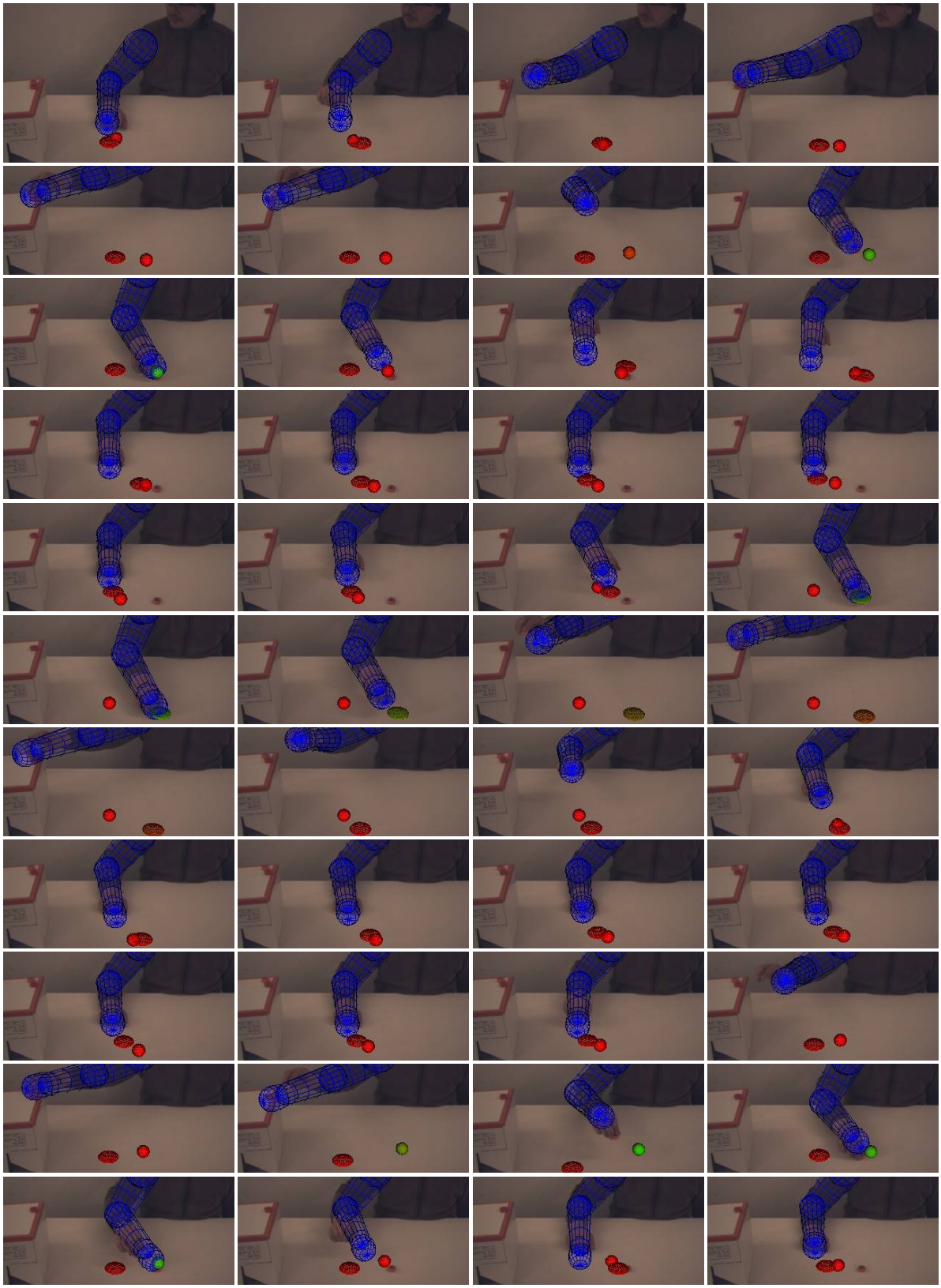


Figure 6.25: Tracking/Recognition of the Actions A and B. Three actions are performed in a sequence, i.e.: B, A, B. The whole sequence has 93 frames. Approximately, every 2nd frame is shown in the order left-to-right. In the last image, one tiny red object is on the table, and one is on the box.

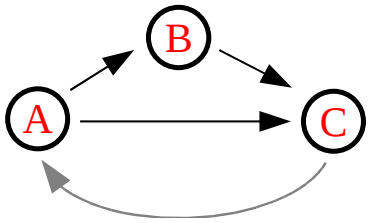


Figure 6.26: Grammar for Simple and Complex Pointing Actions.

6.7.3 Complex Actions

In this experiment the recognition of simple pointing actions and complex dual-pointing actions is considered. Therefore, three different action models trained on demonstrations are used. The actions are: A, the first part of a simple pointing action. This action starts from the base pose (as used in the sections above) and ends when the hand has reached a (object) location $\boldsymbol{\theta}_1$ at the table-top. B, a further pointing action. The action starts with the hand at an arbitrary location $\boldsymbol{\theta}_2$. The action ends when the hand is at an arbitrary second pointed to location $\boldsymbol{\theta}_1$. The training of such a bi-parametric action model is discussed in Section 4.3. C, the withdrawing part of a simple pointing action. This actions starts in a pose, where the hand is at a table-top location $\boldsymbol{\theta}_1$. The actions ends with the base pose. The actions A, B, and C are used as action primitives in order to model complex actions. The actions A, B, and C can be concatenated in two different ways. First, a concatenation of the A and C with matching parameters represent a simple pointing action. Second, a proper concatenation of the action A, B, and C is a complex dual-pointing action. This action could accompany a dialog: "Take this object here, and place it there." Figure 6.26 shows the grammar for the tracking/recognition, where both cases of concatenations are considered simultaneously. The state machine illustrating the grammar has again a loop such that multiple performances can be tracked.

The action space and the particle propagation used in this scenario is discussed in the following more closely. A particle is of the form $\boldsymbol{\omega} = (a, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \tau)$. Usually the parameter $\boldsymbol{\theta}_2$ is not changed when a particle is propagated. The parameter $\boldsymbol{\theta}_2$ has no meaning in the case of a particle with $a \in \{\mathrm{A}, \mathrm{C}\}$. The components $\boldsymbol{\theta}_1$ and τ are propagated as in the case of the simple actions which is discussed in Section 6.7.1. But in the case of a particle with $a = \mathrm{C}$ the parameter $\boldsymbol{\theta}_1$ is kept constant (no diffusion). The handling of a particle, which has a parameter $\tau > 1$, is discussed in the following. The handling of such a particle reflects the grammar and the proper concatenation of the actions. A particle with $\tau > 1$ is handled depending on the action type $a \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ given by the particle:

1. If $a = \mathrm{A}$, a new action type $a' \in \{\mathrm{B}, \mathrm{C}\}$ is chosen randomly. If $a' = \mathrm{C}$, the parameter τ is replaced by $\tau' = \tau - 1$ and the parameter $\boldsymbol{\theta}_1$ is not changed. (This corresponds to a concatenation of the actions A and C to a simple pointing action, where both parts have the same parameters.) If $a' = \mathrm{B}$, the parameters are modified as follows. The parameter τ is replaced by $\tau' = \tau - 1$. The parameter $\boldsymbol{\theta}_2$ is replaced by $\boldsymbol{\theta}_2' = \boldsymbol{\theta}_1$. The parameter $\boldsymbol{\theta}_1$ is replaced by a randomly chosen parameter $\boldsymbol{\theta}_1'$. (This corresponds to a concatenation of the actions A and B, where the action B begins at the location $\boldsymbol{\theta}_2'$ which is the same location that is given by the parameters of action A. The new location $\boldsymbol{\theta}_1'$, to which the person will point, is unknown.)

2. If $a = \mathrm{B}$, then a is replaced by $a' = \mathrm{C}$. The parameter τ is replaced by $\tau' = \tau - 1$. The parameter $\boldsymbol{\theta}_1$ is not changed. (This corresponds to a concatenation of the actions B and C, where the

arm is withdrawn from the location $\boldsymbol{\theta}_1$ which is the finally pointed to location of a dual-pointing action $(\mathrm{A}, \mathrm{B}, \mathrm{C})$.)

3. If $b = \mathrm{C}$, then a is replaced by $a' = \mathrm{A}$. The parameter τ is replaced by $\tau' = \tau - 1$. This case closes the "loop" of the grammar.

The Trial. The Figure 6.27 shows a trial of two consecutive performances of pointing actions. The trial is tracked in real time with 8fps by using camera 2. First, a simple pointing action is performed, which is a concatenation of the action primitives A and B. The corresponding images are in the first two rows of the figure. Below each image, the histograms over τ are given for each basic action $a \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$. The histograms are explained in Figure 6.28. Second, a complex dual pointing action is performed, which is a concatenation of the action primitives A, B, and C. The images are given in the rows 3–5. The color of the ball corresponds to the deviations of the parameters $\boldsymbol{\theta}_1 = (u, v)$. The procedure for calculating the estimate $\hat{\boldsymbol{\theta}} = \mathbb{E}[\boldsymbol{\theta}_1]$ of the parameters (a pointed to location) and the color is exactly the same as discussed for the single actions in Section 6.7.1. It is worthwhile to note that each $\boldsymbol{\theta}_1$ encodes for each action A, B, and C the pointed to location. In the case of the actions A and B this is an unknown location, when a new performance of these basic actions is started.

For the first simple pointing action (rows 1–2), the ball becomes only light green for one time, i.e. when the pointed to location is reached by the hand. For the second action (rows 3–5), i.e. the dual-pointing action, the ball becomes green when the first pointed to location is reached by the hand. The ball becomes then orange. It becomes green again when the second location is reached by the hand. An interesting aspect of both performances is the propagation of the particles of the action A, when $\tau \approx 1$. For the simple pointing action this is the case in the first image of the 2nd row. The particles with $\tau > 1$ are propagated to particles of the actions B and C. However, the posterior probability $p_t(a = \mathrm{B})$ is very small and can be seen only when one looks closely at the histogram of action B. In the following image, the posterior probability $p_t(a = \mathrm{B})$ is still very small, but the probability $p_t(a = \mathrm{C})$ becomes already significant. In the next image the probabilities are $p_t(a = \mathrm{B}) \approx 0$ and $p_t(a = \mathrm{C}) \approx 1$. For the complex dual-pointing action, the situation during the transition from action A to the actions B and C is very similar in the beginning. The transition happens in the second image of the 4th row. In this image, the situation is that $p_t(a = \mathrm{B}) < p_t(a = \mathrm{C})$. In the following image it is $p_t(a = \mathrm{B}) \approx p_t(a = \mathrm{C})$. In the 4th image of the row it becomes obvious that a dual-pointing action is performed, the probabilities are here $p_t(a = \mathrm{B}) \approx 1$ and $p_t(a = \mathrm{C}) \approx 0$. In the last row the particles are propagated after the second pointing motion to the action C. The occurrence of a high posterior probability $p_t(a = \mathrm{B}, \tau \in [0.5, 1.0])$ enables one to distinguish the dual-pointing action from the simple pointing action.

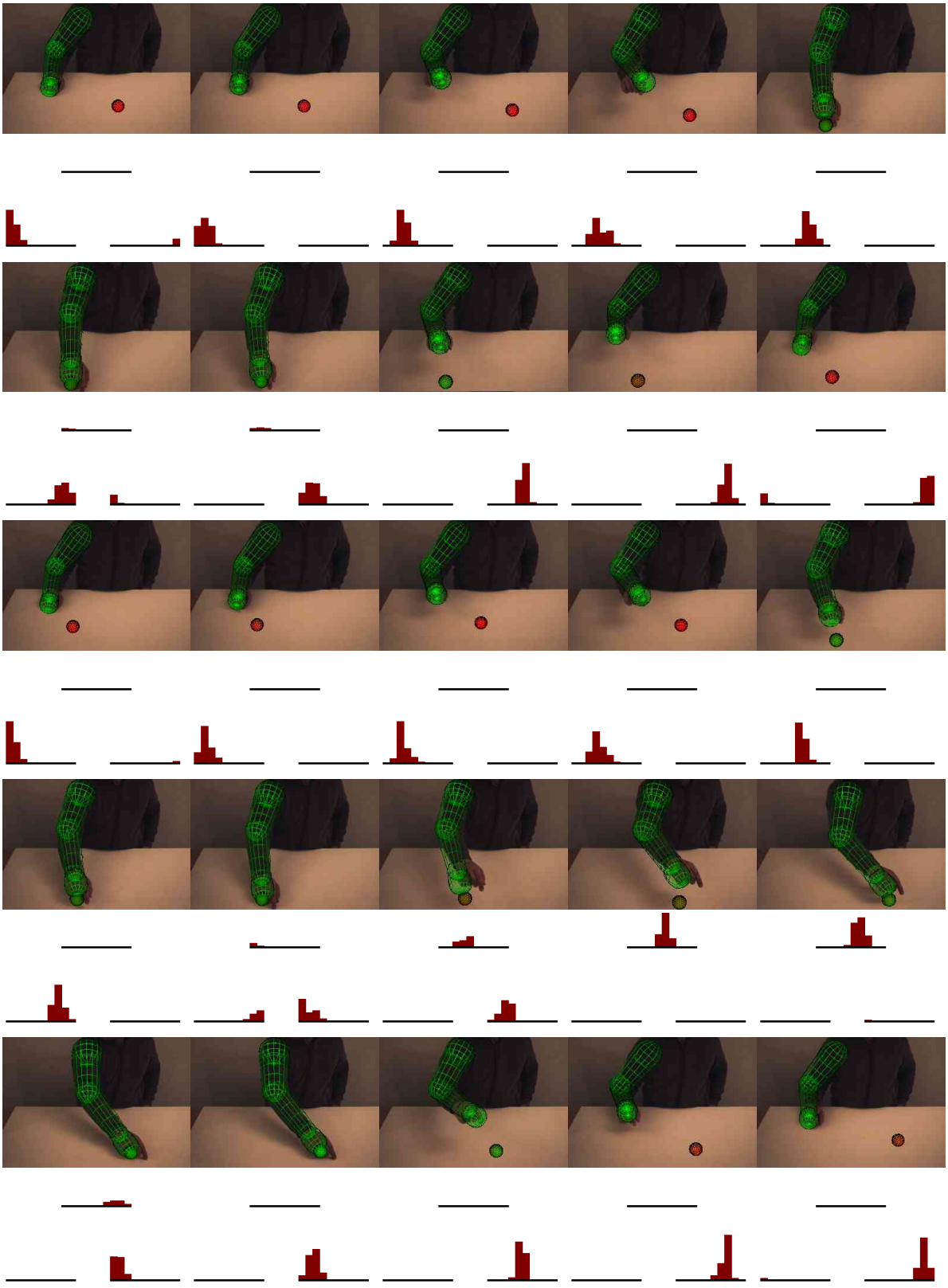


Figure 6.27: Tracking/Recognition of Simple and Complex Pointing Actions. The image sequence shows row-wise approximately every 3rd frame of a trial of 72 frames. Below each image, three histograms of the current particle set are shown. The histograms are explained in Figure 6.28.

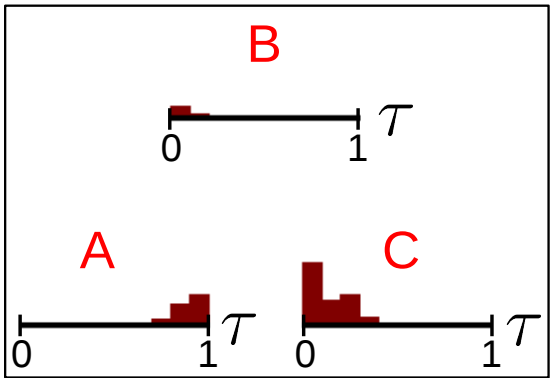


Figure 6.28: Histograms. The figure is an explanation of Figure 6.27. For each action $a \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ a histogram of the function $f_{t,a}(\tau) = p_t(\tau, a)$ over τ is shown for a fixed frame t. The histograms are not normalized. All bins of the three histograms sum up to $100\% = \sum_a \int_\tau p_t(\tau, a)\,\mathrm{d}\tau$.

6.8 Final Remarks

In this chapter a novel concept of tracking within the space of actions, Tracking in Action Space, was presented which combines the aspects of recognition, tracking and prediction based on parametric and time dependent action models. One can argue that this approach is too limited because it is not possible to model all different possible actions with PHMMs. As a response, the starting point for the approach is: 1., an observation is that most actions are object and context dependent. This means that a) object affordances and b) the scenario and the scene state greatly reduce the set of possible actions. 2., according to neuroscientific evidence, actions are composed using action primitives and grammars. Thus, even though the number of possible actions at any time is indeed large, only a small number of actions actually can appear with a certain likelihood. Furthermore, all these possibly appearing actions do not need to be modeled with PHMMs. Instead, it is sufficient to identify the building blocks of actions, i.e. only the action primitives need to be modeled by PHMMs, and complex actions and action sequences are only a concatenation of these primitives.

The experiments were focused on parametric arm actions, which are needed in human-robot communication scenarios. But the belief is here that the approach scales well to more complex actions and more body parts, e.g., by using parallel stochastic grammars which model the synergy of different body parts. Several different aspects were addressed in the experiments: The recovery of the arm pose and the recognition of single and several actions and their parameters, and the tracking/recognition of complex actions which are composed of basis actions. The arm pose is recovered very accurately when a high frame rate is used. The tracking can be performed on-line (8fps) when a single camera view is used. Interestingly, the estimated arm poses and the estimates of the action parameters become in this case very accurate when the hand is close to a target location. The proposed detection scheme allows one to detect the target locations (specified through the parameters of the actions) accurately. The detection scheme allows one to detect the targets/parameters online. However, an estimate of the target location is already given before the target is actually reached. The experiments show that the particular actions and their parameters are also recognized properly when the scenario makes use of different actions. The experiments with complex actions show that the basic actions of a complex action can be identified. An interesting aspect of the concatenation is the use of a bi-parametric basic action, which is parameterized by two locations. This action can start at an arbitrary location and can end at an arbitrary location. This approach allows one to model action sequences which do not make use of a base pose.

Chapter 7

Conclusion

The following conclusion considers problems as well as issues not addressed in the research process. Also suggestions for future study are given.

Motivated by mirror neurons [101, 103] and the similarity between the composition process of speech and human movements [86, 102, 103], PHMMs were investigated in this thesis as a unified representation of (action/movement) primitives that is suitable for recognition and synthesis. Usually models applied in robotics (as DMPs [55, 111], regression models [126, 127], or via-points [78]) aim only at synthesis. A unified representation is particular interesting in the case of humanoid robots, since these need perceptive and generative skills. For the proposed training settings, it was shown that PHMMs can be trained on demonstrations of parametric movements, such that a single model is suitable for recognizing the movements but also for synthesizing accurate prototype movements, as required for robot control. The proposed method for synthesizing from PHMMs generates smooth prototypes movements, which are in this sense very similar to the training movements. The smoothness is established through the proposed training settings, such that the method for synthesizing does not require additional smoothing. Contrary, the approach used in [56] requires smoothing by averaging a large set of generated output sequences. The synthesis method in [132] generates movements which are within the spacial variance of the training data, but the evaluation does not investigate the similarity of training data and synthesized movements in terms of smoothness, etc. An important aspect for the general applicability of PHMMs for representing complex arm movements is that PHMMs were shown to be suitable to represent bi-parametric arm movements which can be used as primitives to compose complex movements. An issue, which has not been explicitly addressed in the thesis, is the amount of demonstrations that are required for the training of PHMMs. The QPHMM action models, used in the approach to tracking, were trained on about 10 to 20 demonstrations. However, in the case of the bi-parametric dual pointing action, the training worked flawlessly for the linear PHMM but not for the quadratic PHMM (QPHMM). The synthesized bi-parametric movements from the QPHMM showed some abnormal characteristics as “wobbling” which are likely a consequence of overfitting and too few training examples. Note, the number of scalar parameters of the parametrization of the QPHMM were 4, which corresponds to an effective number of 10 parameters of the underlying linear PHMM. Similar overfitting effects would likely also occur in the case of the non-linear PHMM due to the large number of network parameters. To mitigate the overfitting problem, one could investigate other training strategies as the use of hyper parameters and Bayesian training. Another aspect which has not been evaluated are movements characterized by style parameters and movements where the

dynamics are important. The general progress of a movement can be decoded from the transition matrix of a (parametric) HMM. However, in the case of certain actions, as throwing a ball, this might be too coarse. In addition the dynamics of a parametric action may depend on the parameters, e. g., when the throwing action is parameterized by the target. The usual PHMMs do not model such a dependency. Here, one could investigated either an extension of the usual PHMMs, where the variation of the transition probabilities is modeled, or the extension of the training data with an additional time variable (Section 2.2.4).

The proposed PHMM-based approach to imitation of parametric movements by humanoid robots requires the mapping of the arm poses and parametrizations. Due to the different sizes and proportions of the robot and the demonstrator, the effect/meaning of a reproduced movement by the robot usually differs from the demonstrated movement. This is especially the case for arm actions as reaching for, pointing to, or relocating (an) object(s), when the imitator has a much longer arm, or, as in the case of the considered robot, a much shorter arm. In [8], the mapping of motion data between different embodiments on the basis of a reference embodiment is addressed. However, the mapping of parameters specifying the effect/meaning of movements is neglected. In the implementation/evaluation of the approach in this thesis, a reaching action was considered for imitation. Here, also the mapping of the parameters is essential to archive that the robot's gripper reaches the right location when reaching for an object. The proposed mapping can be used in the same way for imitating actions as, for example, pointing actions, where the person reaches the meant location with the finger. However, other actions may require a different way to describe their effect/meaning. For example, the meaning of an action as pointing to objects, which are far away, is rather given by the direction and location of the forearm. A consistent mapping of the movement and the parametrization can then be more complicated. For some actions, the effect depends also on the dynamics of a synthesized movement, but the dynamics are usually altered by the mapping of the movement to a different embodiment. Independently of the question of whether PHMMs may be appropriate to represent such actions, the dynamics can be a further complication when establishing a concise mapping of movements and their parameters. If it is too complicated to establish such a mapping, an alternative approach can be to use simulations as discussed in Section 5.1.

In the proposed approach 'tracking in action space', the tracking and recognition is performed in the so called action space of possible actions or action sequences. Parametric actions/primitives are represented through PHMMs trained on demonstrations; possible sequences of primitives are modeled through grammars. The main arguments for using such a primitive-based approach is that the number of complex actions and action sequences are usually very large, whereas the number of primitives is considerably smaller. In addition, the approach allows one to augment the "vocabular" incrementally by additional primitives. Neuroscientific findings [102, 103] support that a primitive approach can be appropriate even in a general scenario. Contrary to the works [25, 89, 129, 130], the goal of the proposed approach was to perform the three tasks, tracking (including 3D pose recovery) and recognition of different actions and their parameters, online in a single framework, where it was argued that the parameters provide an important piece of information about the meaning/effect of many actions. The three aspects have been evaluated in experiments also for single monocular views and different viewpoints. The evaluation considered also complex actions, which were a concatenation of some single- and bi-parametric primitives. However, the settings of each experiment were constrained to a small set of action primitives. From the experiments one can conclude that the approach is suitable in the case of

scenarios as, e.g., human computer interactions with a small action/gesture vocabular. However, in a general framework one would like to have a large set of primitives in combination with a general grammar. Obviously, this complicates the tracking and recognition. An argument, already mentioned in the introduction, is that one could use the context information, as the object types and object locations, to constrain the actions and their parameters. However, still one has to face certain problems even when the context is known: 1. The action parameters can be still very unconstrained: a person can point to a location where no object is placed. 2. Actions can appear very similar, especially when the approach to recognition is online. When a person moves the hand in the direction of an object, it becomes only clear whether the person is pointing to or reaching for the object, when the whole performance of the action has been seen. A problem is when a tracker begins during the performance of an action to focus on the tracking of an action which turns out to be wrong. A solution to this problem could be a proper transition model with transitions between actions even for incompletely performed actions. Besides the augmentation of the vocabular of the implemented framework, following aspects could be addressed in future work: First, utilization of different subjects w.r.t. the training of the movement models and the tracking/recognition. A possible approach is to use a reference model of the human body to establish an intermediate subject-independent representation of the movement data/models (see Section 5.1). Second, the capture process could be extended to the whole upper body and should include then the estimation of the global pose. The extension to the whole upper body raises further questions as, e.g., how to include single and dual arm movements.

Appendix A

Notation

a, b, C, f, γ	some scalar variables, functions, or constants (usually in $\mathbb{R}$ or in $\mathbb{N}$)
$a(t), f(t)$	some functions in time
$\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}; \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$	points and vectors are identified, but points are preferable $\boldsymbol{p}, \boldsymbol{q}, \ldots$
e, π	well known constants are set upright (e.g., $\mathrm{e} = \exp(1)$)
$\ln x = \log x$	the natural logarithm
$\mathcal{A}, \mathcal{B}, \mathcal{D}, \mathcal{X}, \mathcal{Y}, \mathit{\Pi}$	sets (or bag sets in the case of data sets)
$\{0, 1\}$	a set containing the numbers 0 and 1
$\{x \in \mathbb{R} \,\vert\, x^2 > 0\}$	the set $\mathbb{R} \setminus \{0\}$
$\{x_i\}, \{x_i\}_{i=1}^N$	the set with elements x_i: $\{x_1, \ldots, x_N\}$
$\mathbb{N}, \mathbb{N}_0, \mathbb{R}$	natural and real numbers (where, $0 \notin \mathbb{N}$ and $0 \in \mathbb{N}_0$)
$[a, b]$	interval $\{x \vert x \in \mathbb{R}, a \leq x \leq b\}$
$i = 1, \ldots, N$	for each, or, to be done for each $i \in \{1, \ldots, N\}$; in some contexts the order "$i = 1$, $i = 2$, ..." might be of concern
$i = 1, 2, \ldots$	as above, but sequence does not stop, or, the last element is obvious
$\boldsymbol{f}(\boldsymbol{x})$	a function as, e.g., $\boldsymbol{f} : \mathbb{R}^m \supseteq \mathcal{D} \to \mathbb{R}^n, \boldsymbol{x} \mapsto \boldsymbol{f}(\boldsymbol{x})$
$\boldsymbol{x} = (x_i) = (x_i)_{i=1}^N$	a row vector $\boldsymbol{x}$ with elements $x_1, \ldots, x_N$
$(x_1, \ldots, x_N)^\top$	column vector, in *plain text* usually denoted *without* transpose symbol
$(x_1, \ldots, x_N)$	*plain text's column vector* (transpose symbol is usually neglected!)
$\left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right) = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right] = (a, b)^\top$	a column vector of two elements
$\begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix}$	a row vector (in text and formulas)
$(p_\mathrm{x}, p_\mathrm{y}, p_\mathrm{z})$	a 3D point, or, a vector; component subscripts are upright
$\mathbf{e}_i$	i-th vector of "the" standard basis
$[\boldsymbol{x}]_i$	the i-th element of the vector $\boldsymbol{x}$
$\langle \boldsymbol{x} \,\vert\, \boldsymbol{x}' \rangle = \boldsymbol{x}^\top \boldsymbol{x}'$	inner product: $\langle \boldsymbol{x} \,\vert\, \boldsymbol{x}' \rangle = \sum_i x_i x'_i$
$\vert\boldsymbol{x}\vert = \Vert\boldsymbol{x}\Vert_2 = \Vert\boldsymbol{x}\Vert$	Euclidian norm: $\vert\boldsymbol{x}\vert = \sqrt{\langle \boldsymbol{x} \,\vert\, \boldsymbol{x} \rangle}$

$d(\boldsymbol{x}, \boldsymbol{x}')$	some metric
$\|\boldsymbol{x}\|_{\boldsymbol{\Sigma}}$	Mahalanobis norm: $\|\boldsymbol{x}\|_{\boldsymbol{\Sigma}} = \sqrt{\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x}}$
$\mathbf{I}, \mathbf{O};\ \mathbf{I}_{n\times n}, \mathbf{O}_{n\times m}$	identity matrix, and zero matrix; or, with explicit size: $\mathbf{I}_{n\times n} \in \mathbb{R}^{n\times n}$
$\boldsymbol{A} = (a_{ij}) = (a_{ij})_{i,j}$	a matrix $\boldsymbol{A}$ with matrix elements a_{ij}
$\boldsymbol{A} = (a_{ij})_{i,j=1,\ldots,N}$	a square matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$
$\boldsymbol{A} = [\boldsymbol{a}_1 \vert \boldsymbol{a}_2 \vert \cdots \vert \boldsymbol{a}_N]$	a matrix with column vectors $\boldsymbol{a}_i$
$\boldsymbol{A} = \left[\begin{array}{c} \boldsymbol{a}_1^\top \\ \hline \boldsymbol{a}_2^\top \end{array}\right]$	a matrix with two row vectors $\boldsymbol{a}_1^\top$ and $\boldsymbol{a}_2^\top$
$\boldsymbol{A}^\top$	the transpose of a matrix $\boldsymbol{A}$
$[\boldsymbol{A}]_{ij} = [\boldsymbol{A}]_{i,j}$	the i,j-th (i-th row, j-th column) element of the matrix $\boldsymbol{A}$
$\boldsymbol{X} = \boldsymbol{x}_1 \cdots \boldsymbol{x}_T$	a sequence of length T, containing (row) vectors
$\boldsymbol{X} = \boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_t \cdots \boldsymbol{x}_T$	as above, but $\boldsymbol{x}_t$ ($t \in [1 \ldots T]$) is an arbitrary vector of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$
$\boldsymbol{X} = x_1 \cdots x_T$	a sequence of scalars
$\boldsymbol{X} = \boldsymbol{X}_1 \cdots \boldsymbol{X}_T$	a sequence of matrices
$\int x^2 \,\mathrm{d}x$	integral of x^2 over the whole domain of x
$\int \boldsymbol{x}^\top \boldsymbol{x} \,\mathrm{d}\boldsymbol{x}$	integral of $(\boldsymbol{x}^\top \boldsymbol{x})$ over the whole domain of the vector $\boldsymbol{x}$ (e.g., $\mathbb{R}^n$)
$\mathbb{E}[\boldsymbol{x}]$	expectation of $\boldsymbol{x}$, where $\boldsymbol{x}$ is a random vector
$\mathrm{cov}[\boldsymbol{x}]$	the covariance matrix: $\mathrm{cov}[\boldsymbol{x}] = \mathbb{E}[(\boldsymbol{x}^\top - \mathbb{E}[\boldsymbol{x}])^\top (\boldsymbol{x}^\top - \mathbb{E}[\boldsymbol{x}])]$
$\mathcal{N}(\boldsymbol{x} \vert \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution
$\mathcal{N}(\boldsymbol{x} \vert \boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\boldsymbol{x})$	Gaussian density $p(\boldsymbol{x})$; mean: $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}]$, covariance: $\boldsymbol{\Sigma} = \mathrm{cov}[\boldsymbol{x}]$
$\boldsymbol{\lambda} = \boldsymbol{\lambda}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\mu})$	a hidden Markov model (HMM)
$\boldsymbol{\lambda}^{\boldsymbol{\theta}}$	a parametric HMM (PHMM)

Appendix B

Dynamic Time Warping

Dynamic Time Warping (DTW) [62, 97, 108] is a useful tool for comparing and aligning time sequences as they are observed in real life. DTW can be used also to align multivariate sequences [97]. Generally, these sequences vary in their dynamics even though they basically belong to the same process. As an example, a person is considered. The person is pointing at a cup. As a matter of fact, the trajectories of the acting person will vary slightly with each repetition of the same action. The variation will be in space but also in the dynamics of the movement, e. g., depending on the time the person rests with the finger near the cup. In the following it is assumed that two repetitions of the pointing actions have been recorded. The repetitions are called p and q. The recorded trajectories $\boldsymbol{p}(t)$ and $\boldsymbol{q}(t)$ could, for example, describe the whole body configuration over time in Cartesian coordinates. However, in the following plots only one component ($p_{\mathrm{x}}(t)$ and $q_{\mathrm{x}}(t)$) of the trajectories is considered. The component describe the finger tip position along a vector from person to object. The recorded trajectories $p_{\mathrm{x}}(t)$ and $q_{\mathrm{x}}(t)$ are plotted in Figure B.1. The Figures B.2 and B.3 show the results for two different warping approaches.

B.1 Posing the Dynamic Time Warping Problem

In the following, the time warping problem is stated in a more concise continuous form, even though the algorithms work usually on discrete time series. Let $\boldsymbol{p}_i : [0, T_i] \to \mathcal{M}$ be a family of trajectories, where $\mathcal{M}$ is a metric space with a metric $d(\cdot, \cdot)$, e. g., $\mathcal{M} = \mathbb{R}^3$ and $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{y} - \boldsymbol{x}\|_2$. The objective for dynamic time warping is then to find some parameterizations or warping functions $\alpha_i : [0, T_0] \to [0, T_i]$ for the trajectories $\boldsymbol{p}_i(t)$ which minimize a certain distance measure for the re-parameterized trajectories $\boldsymbol{p}'_i(t) = \boldsymbol{p}_i(\alpha_i(t))$. Usually, the warping functions have to comply to some constraints. The interval length T_0 is either a predefined value or is chosen by the algorithm. Typically, dynamic time warping algorithms consider only a pair of trajectories, e.g., $\boldsymbol{p}_1(t)$ and $\boldsymbol{p}_2(t)$.

Distance Measure

A distance measure to define the objective is given through

$$D(\boldsymbol{p}'_1, \boldsymbol{p}'_2) = \int_0^{T_0} d(\boldsymbol{p}'_1(t), \boldsymbol{p}'_2(t))\, \mathrm{d}t, \quad \text{where} \quad \boldsymbol{p}'_i(t) = \boldsymbol{p}_i(\alpha_i(t)). \tag{B.1}$$

A possible extension to the multi trajectory case would be

$$D = \sum_{i<j} \int_0^{T_0} d(\boldsymbol{p}'_i(t), \boldsymbol{p}'_j(t)) \, \mathrm{d}t. \tag{B.2}$$

Warping Constrains

Typical constraints [62] on the warping functions $\alpha_i(t)$ and their derivatives $\alpha'_i(t)$ are:

- Monotonicity:

$$\alpha_i(t') - \alpha_i(t) \geq 0, \quad t' - t \geq 0 \tag{B.3}$$

- Boundary Condition:

$$\alpha_i(t = 0) = 0, \quad \alpha_i(t = T_0) = T_i \tag{B.4}$$

- Continuity:

$$\alpha_i \in \mathcal{C}^0[0, T_0], \tag{B.5}$$

  or:

$$\tau_1 \leq \alpha' \leq \tau_2. \tag{B.6}$$

B.2 Algorithms for Dynamic Time Warping

Typical approaches [62, 97, 108] to dynamic time warping for two time series are dynamic programming based. Since these approaches consider discrete series, the integrals and derivatives have to be interpreted in terms of sums and differences. The dynamic programming approaches differ, for example, in the recurrence equations that are used. Thus, they differ in the constraints of the warping which they comply. The recurrence equations are discussed as a part of the dynamic programming approach, but first the warping problem is restated for two discrete time series.

Restating the DTW Problem for two Discrete Series

Let $\boldsymbol{p} = \boldsymbol{p}_1 \cdots \boldsymbol{p}_{T_1}$ and $\boldsymbol{q} = \boldsymbol{q}_1 \cdots \boldsymbol{q}_{T_2}$ be two time series that have to be aligned through dynamic time warping. The goal is then to find to two index sequences $i_1, \ldots, i_{T_1}$ and $j_1, \ldots, j_{T_2}$ which define the warped series as $\boldsymbol{p}' = \boldsymbol{p}_{i_1} \cdots \boldsymbol{p}_{i_{T_1}}$ and $\boldsymbol{q}' = \boldsymbol{q}_{i_1} \cdots \boldsymbol{q}_{i_{T_2}}$.

Dynamic Programming Approach to DTW

In the dynamic programming framework the two index sequences $i_1, \ldots, i_{T_1}$ and $j_1, \ldots, j_{T_1}$ are determined by finding a path $(i_1, j_1), \ldots, (i_{T_1}, j_{T_2})$ through the trellis of the index sets of the two series that minimizes the overall distance $D = D(\boldsymbol{p}', \boldsymbol{q}') = \sum_t d(\boldsymbol{q}_{i_t}, \boldsymbol{p}_{j_t})$ between the warped series $\boldsymbol{p}'$ and $\boldsymbol{q}'$. Such a minimizing path has to start and end with $(i_1, j_1) = (1, 1)$ and $(i_{T_0}, j_{T_0}) = (T_1, T_2)$. This corresponds to the constraints (B.4). Furthermore, only paths are allowed that follow a certain step

pattern. The step pattern is part of the recurrence equation that is used to find a minimizing path. The dynamic programming approach to DTW is a two step approach. First, the minimal distance $D_{\min}$ of the allowed paths is calculated by filling a distance matrix. Then, a path with distance $D_{\min}$ is decoded from the matrix.

This will be exemplified based on the recurrence equation (A), see Equation (B.7). In this context an allowed path to an index tuple (i, j) through the trellis of the index sets is defined as a path that starts with $(1, 1)$ and has a step pattern, where in each step either one or both components can be incremented by one. The distance matrix $(d_{ij})_{i=1,\ldots,T_1;j=1,\ldots,T_2}$ is supposed to be filled such that each entry d_{ij} is the minimal distance of the allowed paths through the trellis to the index tuple (i, j). A correct initialization of the entries d_{ij} with $i, j \leq 1$ is $d_{11} = d(\boldsymbol{p}_1, \boldsymbol{q}_1)$ and $d_{i1} = d_{1i} = \infty$ for $i > 1$. If one assumes that the entries $d_{(i-1),(j-1)}$, $d_{(i-1),j}$ and $d_{i,(j-1)}$ are the minimal distances of allowed paths to the index tuples $(i-1, j-1)$, $(i-1, j)$ and $(i, j-1)$ then it is obvious that the minimal distance d_{ij} of the allowed paths to the tuple (i, j) can be computed based on the recurrence equation. Based on this observation, one can fill the matrix, for example, in a row-wise order (i.e. $d_{k,2}, d_{k,3}, \ldots, d_{k,T_2}$ for $k = 2, 3, \ldots, T_1$) by applying the recurrence equation. In this way, the recurrence equation is only applied on matrix entries which have been computed previously. It is worthwhile to note that d_{T_0,T_1} gives the minimal distance of the allowed paths to (T_0, T_1), which is the minimal distance $D_{\min} = d_{T_0,T_1}$ of all warping paths which comply to the step pattern. The correctness of this approach can be shown by induction.

Finally, an allowed warping path with minimal distance $D_{\min}$ can be decoded in the reverse order $(i_{T_0}, j_{T_0}), \ldots, (i_1, j_1)$, where one chooses for the current index pair (i, j) the next pair (i', j'), e.g., by choosing the first pair of the pairs $(i', j') = (i-1, j-1), (i-1, j), (i, j-1)$ that satisfies $d_{(i,j)} = d(\boldsymbol{p}_i, \boldsymbol{q}_j) + \min\{d_{(i-1,j-1)}, d_{(i-1,j)}, d_{(i,j-1)}\} = d(\boldsymbol{p}_i, \boldsymbol{q}_j) + d(\boldsymbol{p}_{i'}, \boldsymbol{q}_{j'})$. This procedure stops finally when the pair $(1, 1)$ has been reached.

The dynamic programming approach for the recurrence equation (B), Equation (B.8), is basically the same. But the allowed paths comply to another step pattern. In order to run the procedure in the same way, one could introduce, for example, some dummy entries $d_{i,-1} = d_{-1,i} = \infty$ for $i > 1$. Otherwise, the implementation for equation (B) would access undefined matrix entries. The properties of the different step patterns will be discussed in Section B.3. The computational cost for calculating a warping path is $\mathcal{O}(T_0 T_1) \leq \mathcal{O}(T^2)$ with $T = \max(T_0, T_1)$ due to the number of elements of the distance matrix.

Recurrence Equation (A)

A common recurrence equation [62] is

$$d_{ij} = d(\boldsymbol{p}_i, \boldsymbol{q}_j) + \min\{d_{i-1,j-1}, d_{i-1,j}, d_{i,j-1}\}. \tag{B.7}$$

This equation corresponds to a step pattern that allows either one or both components to be incremented by one.

Recurrence Equation (B)

A recurrence equation [62] with an improved step pattern for some applications is

$$d_{ij} = d(\boldsymbol{p}_i, \boldsymbol{q}_j) + \min\{d_{i-1,j-1}, d_{i-2,j-1}, d_{i-1,j-2}\}. \tag{B.8}$$

This equation corresponds to a step pattern where each component has to be incremented at least by one and allows one component to be incremented by two. This equation results is a more "natural" alignment of the sequences as discussed in Section B.3

B.3 Warping with Different Step Patterns

As mentioned above, the warping through the dynamic programming approach to DTW based on the equations (A) and (B) complies to certain side constraints. The boundary condition (B.4) is met by the algorithm stated in Section B.2. Both step pattern of the equations (A) and (B) comply to the continuity (B.5) and monotonicity (B.3) condition in that sense that the computed index sequences $i_1, i_2, \ldots$ exhibit no decrements ($i_{k-1} - i_k \geq 0$) and have no (big) gaps, since an index is incremented at most by two ($i_{k-1} - i_k \leq 2$). The major difference between the equation (A) and (B) is that (B) imposes (as in (B.6)) a minimal step size $i_{k-1} - i_k \geq \tau_1 = 1$ The Figure B.2 and Figure B.3 show the warping results for the recurrence equations (A) and (B). The original series are given in Figure B.1. In both cases the warping affects the total length T_0 in different ways due to the allowed step sizes. But this effect could be undone be rescaling. Here, the warping or alignment at the local maxima of the series is more interesting. In Figure B.2, the maxima of the green sequence show a one-to-many alignment to the red sequence, where the index sequence addresses over a long period the same sample of the original sequence. This seems to be a rather unnatural warping for motion trajectories, since the trajectory has now a plateau, where no motion exists. In contrast to equation (A), such an unnatural one-to-many alignment is prevented by equation (B) as shown in Figure B.3. This is ensured through the minimal step size of at least one for each index sequence, and seems to result in a more meaningful alignment for motion sequences.

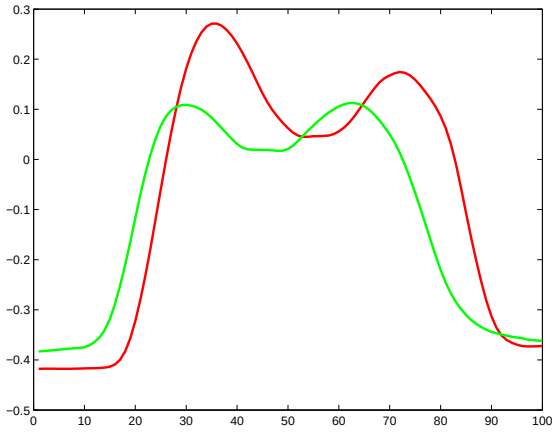


Figure B.1: Exemplar Sequences used for Time Warping.

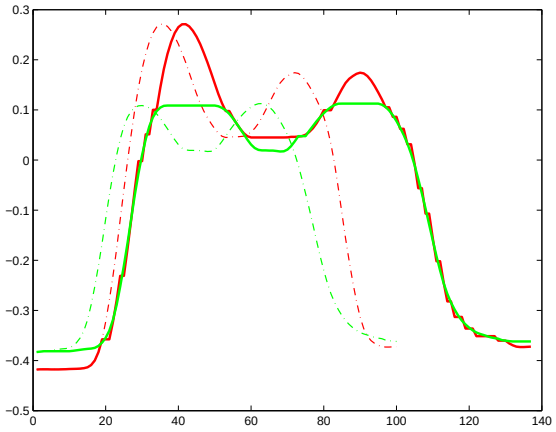


Figure B.2: Exemplar Sequences, Time-Warped by Recurrence Equation (A).

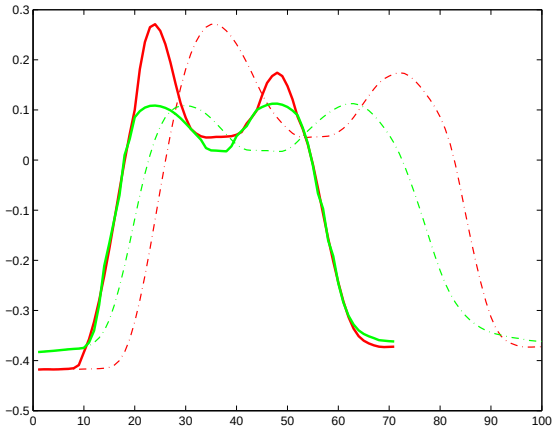


Figure B.3: Exemplar Sequences, Time-Warped by Recurrence Equation (B).

Appendix C

Particle Filters

Particle filtering [5] is a technique to facilitate nonlinear Bayesian tracking. The filtering technique allows one to approximate the state density and moments for a dynamic non-linear continuous state-space model, where the current state $\boldsymbol{\theta}_t$ (generally) can not be observed explicitly, and is, in this sense, latent. The objective is to infer the posterior density $p_t(\boldsymbol{\theta}_t) \equiv p(\boldsymbol{\theta}_t|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$ based on a known observation model $p_t(\boldsymbol{x}_t|\boldsymbol{\theta}_t)$ from the history of observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$ that are measured at discrete time steps t. The basic means within this approach is a stochastic sampling method, where the densities are approximatively represented through a set of weighted samples. This method is applied to infer (approximatively) the posterior state density $p_t(\boldsymbol{\theta}_t)$ from the observation $\boldsymbol{x}_t$ and the previous density $p_{t-1}(\boldsymbol{\theta}_{t-1})$. In an online approach, this allows one to estimate the sequence $p_1(\boldsymbol{\theta}_1), p_2(\boldsymbol{\theta}_2), \ldots$ of state densities (and moments), iteratively, where the current density $p_t(\boldsymbol{\theta}_t)$ is basically an update of the previous density based on the model dynamics and the information derived from the new observation available. The update procedure is formulated in a Bayesian manner as derived in Section C.2. The particle filter framework is described in Section C.3. The assumptions of the underlying model are stated in following Section C.1. The assumptions are there also compared to the assumption of alternative Bayesian tracking techniques, which are, for example, Kalman filtering (KF) [5] and the extended Kalman filter (EKF) [5]. It is worthwhile to note that the assumptions of the particle filter are less restrictive. This regards the system and observation model, but also concerns the shape of densities. Since the densities are approximated in the particle approach through samples, the state density can be multimodal, and thus the approach provides a simple framework for tracking multiple hypothesis [57]. An advantage of the KF is that it does not approximates the state density (provided that the restrictive assumption discussed in Section C.1 are met). The particle filtering technique as such is used in a diverse range of applied science [77], and was developed several times independently. As consequence, it appears in the literature (in slight variations) under various names such as the bootstrap filter (Gordon et al. 1993 [33]), sequential Monte Carlo (SEM) [24], and Condensation (Isard and Blake 1998 [57]); in [24] the bootstrap filter is addressed as the first successful application of (SEM) within the field of nonlinear filtering. A brief history of particle filtering is given in [77]. A short introduction to particle filtering in given in [14]; overview articles and tutorials are for example [24] and [5].

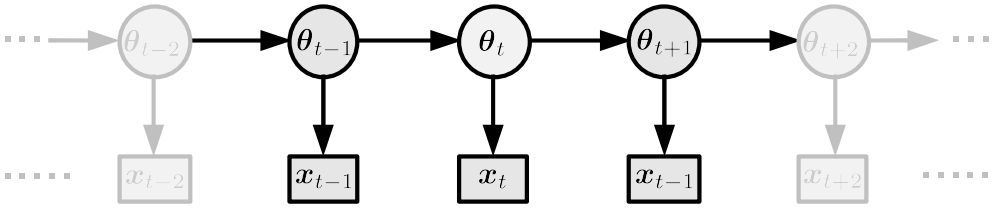


Figure C.1: Markovian Dependencies between States and Observations.

C.1 Generic Model

The stochastic model for the Bayesian tracking through particle filtering consists of two stages, the system model and a measurement model [24, 33, 5]. The dynamic system model $\boldsymbol{f}_t(\cdot,\cdot)$ and the measurement model $\boldsymbol{g}_t(\cdot,\cdot)$ are given in the following equations. The system model is stated as discrete time, continuous state-space model in general form:

$$\boldsymbol{\theta}_t = \boldsymbol{f}_t(\boldsymbol{\theta}_{t-1}, \boldsymbol{u}_t) \quad \sim \quad p_t(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}), \tag{C.1}$$

$$\boldsymbol{x}_t = \boldsymbol{g}_t(\boldsymbol{\theta}_t, \boldsymbol{v}_t) \quad \sim \quad p_t(\boldsymbol{x}_t|\boldsymbol{\theta}_t), \tag{C.2}$$

where the $\boldsymbol{f}_t(\cdot,\cdot)$ and $\boldsymbol{g}_t(\cdot,\cdot)$ are (possibly) non-linear functions which describe the evolution of a state sequences $(\boldsymbol{\theta}_t)_{t\in\mathbb{N}}$ and the process of generating observations $\boldsymbol{x}_t$. The process noise sequence $(\boldsymbol{u}_t)_{t\in\mathbb{N}}$ and the sequence of measurement noise $(\boldsymbol{v}_t)_{t\in\mathbb{N}}$ are assumed to be drawn i.i.d. with known statistics $p_t(\boldsymbol{u}_t)$ and $p_t(\boldsymbol{v}_t)$. The right-hand sides of (C.1) and (C.2) denote the corresponding models in a probabilist manner as conditional state transition density $p_t(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$ and observation density $p_t(\boldsymbol{x}_t|\boldsymbol{\theta}_t)$. A further assumption is that the density $p_0(\boldsymbol{\theta}_0)$ of the initial system state is known. The dependences within the described process model are illustrated as a graphical model in Figure C.1. The stochastic process underlying the state evolution is here a Markov process of order one, i.e.,

$$p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_0,\ldots,\boldsymbol{\theta}_{t-1}) = p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}). \tag{C.3}$$

The generic model stated in (C.1) and (C.2) is used in its general form for particle filtering. In the following, the restrictions to this model are discussed for Kalman filtering in order to point out the generality of the particle approach. The Kalman filter assumes [5] that the posterior density is Gaussian at every time step. This can be ensured through the following assumptions: (a) the functions $\boldsymbol{f}_t$ and $\boldsymbol{g}_t$ are of the form $\boldsymbol{f}_t(\boldsymbol{\theta}_{t-1}, \boldsymbol{u}_t) = \boldsymbol{F}_t\boldsymbol{\theta}_{t-1} + \boldsymbol{u}_t$ and $\boldsymbol{g}_t(\boldsymbol{\theta}_t, \boldsymbol{v}_t) = \boldsymbol{G}_t\boldsymbol{\theta}_t + \boldsymbol{v}_t$ with matrices $\boldsymbol{F}_t$ and $\boldsymbol{G}_t$; (b) the noise densities $p_t(\boldsymbol{u}_t)$ and $p_t(\boldsymbol{v}_t)$ are Gaussian. (One can proof that $p_t(\boldsymbol{\theta}_t)$ is Gaussian if $p_{t-1}(\boldsymbol{\theta}_{t-1})$ is Gaussian, provided that the assumptions (a) and (b) are met [5].) The extended Kalman filter (EKF) overcomes some of the restrictions and can be applied in the nonlinear case of the function $\boldsymbol{f}_t$ and $\boldsymbol{g}_t$. The EKF uses therefore a locally linear approximation of these functions, but still approximates the posterior state density through a Gaussian [5].

C.2 Recursive Bayesian Inference

In this section, the recursive rule to infer the posterior state density $p(\boldsymbol{\theta}_t|\boldsymbol{x}_1,\ldots,\boldsymbol{x}_t)$ on the basis of the posterior state density $p(\boldsymbol{\theta}_{t-1}|\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{t-1})$ and the observation $\boldsymbol{\theta}_t$ is derived. For convenience,

the notations $\boldsymbol{X}_t = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$ and $p_t(\boldsymbol{\theta}_t) = p(\boldsymbol{\theta}_t|\boldsymbol{X}_t)$ are used. Starting point is the Chapman-Kolmogorov equation [5]:

$$p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1}) = \int p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) p_{t-1}(\boldsymbol{\theta}_{t-1}) \, \mathrm{d}\boldsymbol{\theta}_{t-1} \tag{C.4}$$

which infers the knowledge about the state $\boldsymbol{\theta}_t$ given the observations up to the previous time step $t-1$. This is basically the application of the system model $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$. One should note that conditional independency $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \boldsymbol{X}_{t-1}) = p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$ is used in (C.4), which is part of the model assumptions (Section C.1) and can be seen in Figure C.1. The density $p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1})$ is regarded in the following as a prior in order to infer the posterior knowledge about the density $p_t(\boldsymbol{\theta}_t)$ given the new observation $\boldsymbol{x}_t$ at time t (and the previous observations):

$$\begin{aligned}
p_t(\boldsymbol{\theta}_t) &= p(\boldsymbol{\theta}_t|\boldsymbol{X}_t) && \text{(C.5)}\\
&= p(\boldsymbol{\theta}_t|\boldsymbol{x}_t, \boldsymbol{X}_{t-1}) && \text{(C.6)}\\
&= \frac{p(\boldsymbol{x}_t|\boldsymbol{\theta}_t, \boldsymbol{X}_{t-1}) p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1})}{p(\boldsymbol{x}_t|\boldsymbol{X}_{t-1})} && \text{(C.7)}\\
&= \frac{p(\boldsymbol{x}_t|\boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1})}{p(\boldsymbol{x}_t|\boldsymbol{X}_{t-1})} && \text{(C.8)}\\
&= \frac{p(\boldsymbol{x}_t|\boldsymbol{\theta}_t) \int p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) p_{t-1}(\boldsymbol{\theta}_{t-1}) \, \mathrm{d}\boldsymbol{\theta}_{t-1}}{p(\boldsymbol{x}_t|\boldsymbol{X}_{t-1})}, && \text{(C.9)}
\end{aligned}$$

where the Bayes' rule, the conditional independence property of the model, and the Chapman-Kolmogorov equation (C.4) are applied in the Equations (C.7), (C.8), and (C.9). Note that $k_t \equiv p(\boldsymbol{x}_t|\boldsymbol{X}_{t-1}) = \int p(\boldsymbol{x}_t|\boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1}) \, \mathrm{d}\boldsymbol{\theta}_t$ is only a normalization factor which is independent of $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t-1}$ and constant for each time step t. The normalization factor k_t in (C.9) is not important in the particle framework as discussed in the next section. The recursive inference rule (C.9) can be stated in a concise way:

$$p_t(\boldsymbol{\theta}_t) \propto \underbrace{p(\boldsymbol{x}_t|\boldsymbol{\theta}_t)}_{\text{measure}} \underbrace{\int \underbrace{p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})}_{\text{predict}} p_{t-1}(\boldsymbol{\theta}_{t-1}) \, \mathrm{d}\boldsymbol{\theta}_{t-1}}_{p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1})}. \tag{C.10}$$

C.3 Particle Filter Framework

In this section, the building blocks of the particle filtering framework are presented mainly on the basis of [57]. The framework is a means to approximate iteratively the posterior densities $p_1(\boldsymbol{\theta}_1), p_2(\boldsymbol{\theta}_2), \ldots$ of the system state, provided that the system and measurement model are given in a form which complies with the generic definition in Section C.1. The recursive rule (C.10) is the basic means in the particle framework in order to infer the state density $p_t(\boldsymbol{\theta}_t)$ at time step t from the previous density $p_{t-1}(\boldsymbol{\theta}_{t-1})$ and the new observation $\boldsymbol{\theta}_t$ available. In the particle approach, the approximation of the posterior is achieved through evaluating the likelihood function on a sample set of the prior density. This is discussed in Section C.3.1. The algorithm for particle filtering is specified in Section C.3.1.

C.3.1 Approximating the Posterior Density

The crucial point is here the evaluation of the posterior density $p(\boldsymbol{\theta}|\boldsymbol{x})$ as it appears in the Bayes' rule:

$$p(\boldsymbol{\theta}|\boldsymbol{x}) = kp(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{C.11}$$

In the case that (C.11) can not be evaluated in closed form, the posterior density $p(\boldsymbol{\theta}|\boldsymbol{x})$ and its moments can still be approximated through a set $\mathcal{S}$ of weighted samples:

$$\mathcal{S} = \{(\boldsymbol{\theta}^n, \pi^n) \,|\, n = 1, \ldots, N\}. \tag{C.12}$$

The samples $\boldsymbol{\theta}^n$ and weights π^n are generated as follows. The samples $\boldsymbol{\theta}^n$ are drawn from the prior density $p(\boldsymbol{\theta})$. The weights π^n are assigned in proportion to the likelihood function $p(\boldsymbol{x}|\boldsymbol{\theta})$:

$$\pi^n = \frac{p(\boldsymbol{x}|\boldsymbol{\theta}^n)}{\sum_{n=1}^{N} p(\boldsymbol{x}|\boldsymbol{\theta}^n)}. \tag{C.13}$$

The fundamental statement in [57] is that the values $\boldsymbol{\theta} = \boldsymbol{\theta}^n$, where n is chosen with probability π^n, approximate the posterior $p(\boldsymbol{\theta}|\boldsymbol{x})$ with increasing accuracy when the number N of samples increases. The conditional expectation of some function $\boldsymbol{h}$ can be approximated by

$$\mathrm{E}[\boldsymbol{h}(\boldsymbol{\theta})|\boldsymbol{x}] \approx \sum_{n=1}^{N} \boldsymbol{h}(\boldsymbol{\theta}^n)\pi^n.$$

For example, the mean of the posterior density can be approximated by using $\boldsymbol{h}(\boldsymbol{\theta}) = \boldsymbol{\theta}$.

C.3.2 The Particle Filter

The particle filter maintains a set of weighted samples $\mathcal{S}_t = \{(\boldsymbol{\theta}_t^n, \pi_t^n) \,|\, n = 1, \ldots, N\}$ as defined in (C.12). The weighted sample set is an approximation of the current posterior state density $p_t(\boldsymbol{\theta}_t) = p(\boldsymbol{\theta}_t|\boldsymbol{X}_t)$ given the history of measurements $\boldsymbol{X}_t = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$. In order to understand the filter operation, it is now assumed that this is correct at the time step $t-1$. One iteration $(t-1) \mapsto t$ of the particle filter creates then an updated set $\mathcal{S}_t$ of weighted samples based on the previous set $\mathcal{S}_{t-1}$ and the new measurement $\boldsymbol{x}_t$. The filter iteration $(t-1) \mapsto t$ is defined as:

1. **Select.** Select N samples $\boldsymbol{\theta}_t^n$ from $\{\boldsymbol{\theta}_{t-1}^n\}$ w.r.t. the weights π_{t-1}^n.

   (a) compute cumulative weights: $c^n = \sum_{i=1}^{n} \pi_{t-1}^i$
   (use therefore the recurrence $c^n = c^{n-1} + \pi_{t-1}^n$)

   (b) for each $n = 1, \ldots, N$

   - generate random numbers $r \in [0, 1]$, uniformly distributed
   - find the smallest index j for which $c^n \geq r$
     (binary subdivision [57] is here an efficient search method)
   - set: $\boldsymbol{\theta}_t^n = \boldsymbol{\theta}_{t-1}^j$

2. **Predict.** Predict the states $\boldsymbol{\theta}_t^n$ due to the model dynamics.

   - set: $\boldsymbol{\theta}_t^n = \boldsymbol{f}_t(\boldsymbol{\theta}_t^n, \boldsymbol{u}_t^n)$, where $\boldsymbol{u}_t^n$ is drawn from $p_t(\boldsymbol{u}_t)$
     (see Section C.1 for the state evolution by $\boldsymbol{f}_t$ and the process noise $p_t(\boldsymbol{u}_t)$)

3. **Measure.** Compute new weights π_t^n for the samples $\boldsymbol{\theta}_t^n$.

   (a) evaluate the likelihood function: $\pi_t^n = p_t(\boldsymbol{x}_n|\boldsymbol{\theta}_t^n)$
   (see also (C.10))

   (b) normalize the weights: π_t^n
   (such that $\sum_n \pi_t^n = 1$)

The filter iteration $(t-1) \mapsto t$ is illustrated in Figure C.2. In the stage Select, N samples $\boldsymbol{\theta}_t^n$ are selected from the weighted sample set $\mathcal{S}_{t-1} = \{(\boldsymbol{\theta}_{t-1}^n, \pi_{t-1}^n)\}$ w.r.t. the sample weights π_{t-1}^n. This sample set $\{\boldsymbol{\theta}_t^n\}$ still approximates the posterior $p_{t-1}(\boldsymbol{\theta}_{t-1}) = p(\boldsymbol{\theta}_{t-1}|\boldsymbol{X}_{t-1})$ as the weighted sample set $\mathcal{S}_{t-1}$ does but without weights. The unweighted sample set $\{\boldsymbol{\theta}_t^n\}$ is then predicted as given through the model dynamics $p_t(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$, see also (C.10). The sample set $\{\boldsymbol{\theta}_t^n\}$ is now an approximation of the density $p(\boldsymbol{\theta}_t|\boldsymbol{X}_{t-1})$ that is used as a prior in the Bayes' rule. The measuring through the likelihood function $p(\boldsymbol{x}_t|\boldsymbol{\theta}_t)$ results then in a weighted set $\mathcal{S}_t = \{(\boldsymbol{\theta}_t^n, \pi_t^n)\}$ where the weights π_t^n are proportional to the likelihood $p(\boldsymbol{x}_t|\boldsymbol{\theta}_t^n)$. The weighted set $\mathcal{S}_t$ approximates now the new posterior $p_t(\boldsymbol{\theta}_t) = p(\boldsymbol{\theta}_t|\boldsymbol{X}_t)$ in the way as described in Section C.3.1. The mean of the state density can now be approximated as mentioned in Section C.3.1.

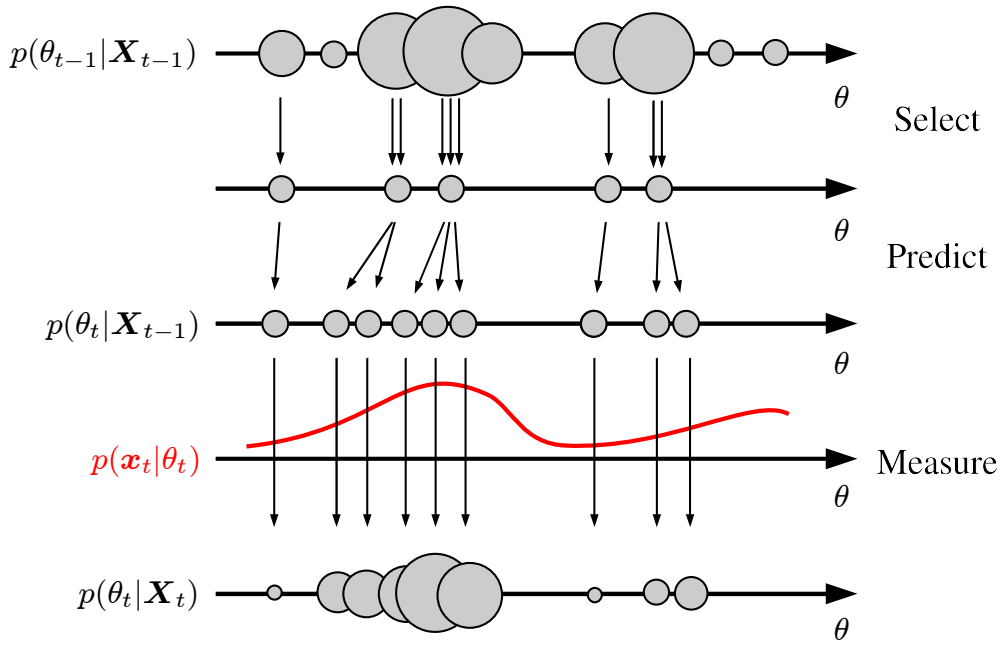


Figure C.2: One Iteration of the Particle Filter. The figure illustrates schematically one iteration $(t-1) \mapsto t$ of the particle filter for a one-dimensional state space. The new posterior state density $p_t(\theta) = p_t(\theta|\boldsymbol{X}_t)$ and the previous density $p_{t-1}(\theta) = p_{t-1}(\theta|\boldsymbol{X}_{t-1})$ are approximated through $N = 9$ weighted samples. The balls sizes indicate the weights π^n_{t-1} and π^n_t. The filter stages Select, Predict, and Measure (noted on the right) are defined in Section C.3.2, see also Equation (C.10).

Appendix D

Scene Modeling

In 3D computer graphics, a 3D scene contains usually a lage set of basic geometries which are composed to represent objects and the scene as such. Typically, a scene graph (Section D.4) is used to represent such a scene in a structured way. The graph representation can be used to describe how objects are composed from basic geometries and how the scene is composed from the objects. An important aspect of such a scene graph are transformations, which can be used to define where/how objects are placed in the scene. Important transformations are the scaling, rotation, and translation of objects in the scene. These transformations are also useful for animation. For example, a proper representation of a human body as a scene graph is useful to articulate the body in a way that is similar to the kinematic structure of the human body, e.g., a shoulder transformation node can be used to animate the lifting of an arm. A useful tool in the context of scene graphs are homogenous coordinates and matrices. The concatenation of several rotations and translations can be represented by a single homogenous 4×4 matrix. To display a scene, the scene is usually rendered in some camera view. In order to mimic the image capturing process of a real scene by a video camera, it is of particular importance to use a camera model for the rendering which has the same perspective properties as the real camera. Such a camera model is introduced in Section D.3. The internal parameters of this camera model can be represented as a homogenous 3×3 matrix, $\boldsymbol{K}$, the calibration matrix [42]. The external parameters describe the pose (location and rotation) of a camera. The internal and external parameters can be combined in a single matrix $\boldsymbol{P} \in \mathbb{R}^{3\times4}$ which describes the mapping of 3D points onto 2D image points.

In the following, homogenous coordinates are introduced. In Section D.2, the representation of basic 3D transformations as homogenous 4×4 matrices is presented. The camera model and scene graphs are discussed in Section D.3 and Section D.4. A more comprehensive but long introduction to homogenous coordinates, transformations and camera models can be found in [42].

D.1 Homogeneous Coordinates

In this section homogeneous coordinates are introduced. Homogenous coordinates allow one to represent affine and projective transformations in a concise matrix notation (homogenous matrices). The homogenous coordinates are introduced in the following based on the projective space. This introduction is rather informal. A more comprehensive introduction to projective spaces is given in [34] or [42].

The projective space $\mathcal{P}(\mathcal{V})$ of a vector space $\mathcal{V}$ is defined as a set $\mathcal{P}(\mathcal{V}) = \{\mathbb{R}\boldsymbol{v} \,|\, \boldsymbol{v} \in \mathcal{V} \setminus \mathbf{0}\}$ containing all straight lines in $\mathcal{V}$ through the origin $\mathbf{0}$. Obviously, the set $\mathbb{R}\boldsymbol{v} \equiv \{\lambda\boldsymbol{v}|\lambda \in \mathbb{R}\}$ defines the same line for two vectors $\boldsymbol{v}$ and $\boldsymbol{v}'$ ($\boldsymbol{v}, \boldsymbol{v}' \in \mathcal{V} \setminus \{\mathbf{0}\}$) if the vectors are identical up to a scale factor, i.e., there exists a scalar $\lambda \in \mathbb{R}$ thus that $\boldsymbol{v} = \lambda\boldsymbol{v}'$.

In the following the case $\mathcal{V} = \mathbb{R}^4$ is assumed. The coordinates of a vector $\boldsymbol{v} = (x, y, z, w)$ that represents a point $\mathbb{R}\boldsymbol{v} \in \mathcal{P}(\mathbb{R}^4)$ are called homogenous coordinates. The use of homogenous coordinates is indicated in the following by brackets. The homogenous coordinates x, y, z, w are identified with a point $\mathbb{R}\boldsymbol{v} \in \mathcal{P}(\mathbb{R}^4)$ in the projective space, i.e.

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \equiv \mathbb{R}\boldsymbol{v} = \mathbb{R} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}.$$

The identity of represented points through homogenous coordinates is then

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \simeq \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \qquad \Longleftrightarrow \qquad \exists\lambda \in \mathbb{R} : \quad \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \lambda \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix},$$

where the use of "$\simeq$" emphasizes that the equality of the represented points does not imply that the homogenous coordinates are the same.

By using the usual matrix multiplication rules on homogenous coordinates

$$\underbrace{\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}}_{\boldsymbol{p}'} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix}}_{\boldsymbol{H}} \underbrace{\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}}_{\boldsymbol{p}},$$

one can define a mapping of a point $\boldsymbol{p}$ to a point $\boldsymbol{p}'$. The mapping defined by $\boldsymbol{H} \neq \mathbf{O}$ is independent of the actually used homogenous coordinate representation, since the representation of both points are defined only up to a scale factor. As a further consequence, two matrices $\boldsymbol{H}$ and $\lambda\boldsymbol{H}$ (for $\lambda \in \mathbb{R} \setminus \{\mathbf{0}\}$) define the same mapping.

The affine space $\mathbb{R}^3$ can be embedded in a canonical way in the projective space $\mathcal{P}(\mathbb{R}^4)$ by using the mapping $\boldsymbol{\iota} : \mathbb{R}^3 \to \mathcal{P}(\mathbb{R}^4)$ for a point $\boldsymbol{p} = (x, y, z) \in \mathbb{R}^3$ as given by

$$\boldsymbol{\iota} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \text{with} \qquad \boldsymbol{\iota}^{-1} \left(\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \right) = \begin{pmatrix} x'/w' \\ y'/w' \\ z'/w' \end{pmatrix}.$$

Even though the homogenous coordinates are defined only up to a scale factor $\lambda \neq 0$, the inverse mapping is well defined, since for each embedded point $\boldsymbol{\iota}(\boldsymbol{p})$ the component w is nonzero. Obviously, $\boldsymbol{\iota}^{-1}(\boldsymbol{\iota}(\boldsymbol{p})) = \boldsymbol{p}$ for each $\boldsymbol{p} \in \mathbb{R}^3$. This allows representation of an affine mapping $\boldsymbol{A}(\boldsymbol{p}) = \boldsymbol{L}\boldsymbol{p} + \boldsymbol{t}$

(where $\boldsymbol{L} = (l_{ij}) \in \mathbb{R}^{3\times3}$ and $\boldsymbol{t} = (t_\mathrm{x}, t_\mathrm{y}, t_\mathrm{z})$) through a single homogenous matrix $\boldsymbol{H}$, as given in the following equation:

$$\underbrace{\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}}_{\boldsymbol{\iota}(\boldsymbol{A}(\boldsymbol{p}))} \simeq \underbrace{\begin{bmatrix} l_{11} & l_{12} & l_{13} & t_\mathrm{x} \\ l_{21} & l_{22} & l_{23} & t_\mathrm{y} \\ l_{31} & l_{32} & l_{33} & l_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{H}} \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{\boldsymbol{\iota}(\boldsymbol{p})} = \boldsymbol{\iota}(\boldsymbol{L}\boldsymbol{p} + \boldsymbol{t}).$$

Again the homogenous matrix $\boldsymbol{H}$ and the choice of the homogenous coordinates are defined only up to scale factors unequal zero.

D.2 Affine Mappings

In this section basic affine mappings are discussed which are used within this thesis. These mappings are usually applied on objects which can be defined as a set of 3D points. As an example, the forearm of a human model needs to be rotated around the joint of the elbow when the elbow of the model is articulated.

D.2.1 Translation

A translation in the affine space $\mathbb{R}^3$ is a mapping of a point $\boldsymbol{p}$ to a translated point $\boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{t}$, where $\boldsymbol{t} = (t_\mathrm{x}, t_\mathrm{y}, t_\mathrm{z})$ is the translation vector. The translation can be represented by the following homogenous 4×4 matrix:

$$\boldsymbol{T}_{\boldsymbol{t}} = \begin{bmatrix} 0 & 0 & 0 & t_\mathrm{x} \\ 0 & 0 & 0 & t_\mathrm{y} \\ 0 & 0 & 0 & t_\mathrm{z} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{O} & \boldsymbol{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}.$$

An example for a translation of an object is given in Figure D.1.

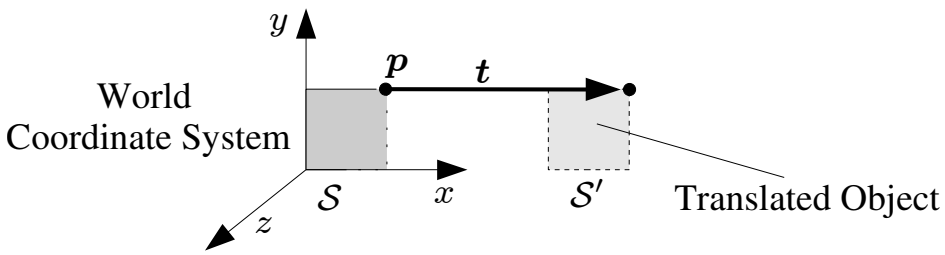


Figure D.1: Translation of a Square. The square $\mathcal{S}$ is defined in the world frame as a set of points. The square is translated by a vector $\boldsymbol{t}$ on a square $\mathcal{S}'$. Therefore, each point $\boldsymbol{p} \in \mathcal{S}$ is translated to a new point $\boldsymbol{p}' \in \mathcal{S}'$, where $\boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{t}$.

D.2.2 Scaling

In order to scale an object one can map each point $\boldsymbol{p} = (p_\mathrm{x}, p_\mathrm{y}, p_\mathrm{z})$ of an object to a point $\boldsymbol{p}' = (s_\mathrm{x} p_\mathrm{x}, s_\mathrm{y} p_\mathrm{y}, s_\mathrm{z} p_\mathrm{z})$, where $s_\mathrm{x}, s_\mathrm{y}, s_\mathrm{z}$, are the scaling factors for each component. The scaling can be written as a homogenous 4×4 matrix

$$\boldsymbol{T_t} = \begin{bmatrix} s_\mathrm{x} & 0 & 0 & 0 \\ 0 & s_\mathrm{y} & 0 & 0 \\ 0 & 0 & s_\mathrm{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The rescaling of an object is performed by this transformation with respect to the origin, i.e., if the object is defined, e.g., in a reference frame with an object center $\boldsymbol{c} \neq \boldsymbol{0}$, then the object center also is translated. If one considers the square $\mathcal{S}$ in Figure D.1 with a object center $\boldsymbol{c} = (0.5, 0.5, 0.5)$ and scaling factors $s_\mathrm{x} = s_\mathrm{y} = s_\mathrm{z} = 0.5$, then the object is scaled isotropically to 50% of its original size, but the center of the rescaled object is now $\boldsymbol{c}' = (0.25, 0.25, 0.25)$.

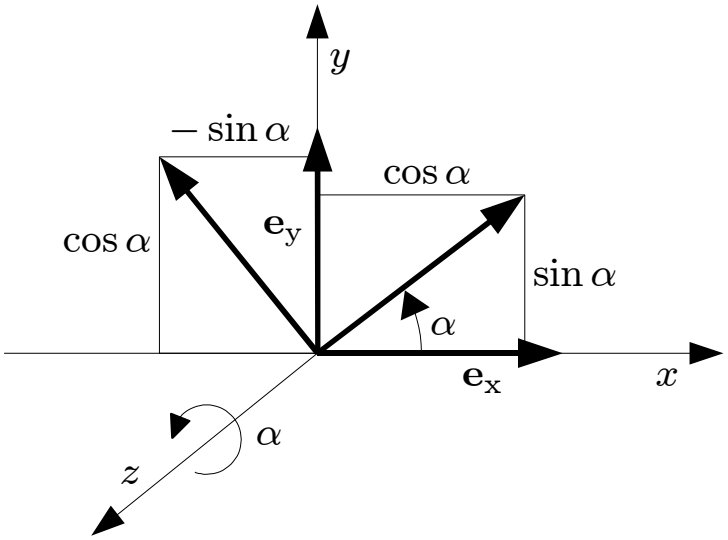


Figure D.2: Rotation around Z-Axis. The figure shows a rotation of the standard basis vectors $\mathbf{e}_\mathrm{x}$ and $\mathbf{e}_\mathrm{y}$ around the z-axis. The rotated basis vectors are $\mathbf{e}'_\mathrm{x} = \cos\alpha \cdot \mathbf{e}_o x + \sin\alpha \cdot \mathbf{e}_o y$ and $\mathbf{e}'_\mathrm{y} = -\sin\alpha \cdot \mathbf{e}_o x + cos\alpha \cdot \mathbf{e}_o y$.

D.2.3 Rotations

A rotation of an object point $\boldsymbol{p} \in \mathbb{R}^3$ to a point $\boldsymbol{p}'$, where the object is rotated around the origin, can be represented as a matrix 3×3 matrix $\boldsymbol{R} = (r_{ij})$, where $\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p}$. A rotation matrix is always orthogonal, and the inverse rotation is therefore simply given by $\boldsymbol{R}^{-1} = \boldsymbol{R}^\top$. The rotation matrices for a rotation around the x-, y-, and z-axis can be constructed by using the sines and cosines functions. The rotation matrix $\boldsymbol{R}$ around the z-axis is given by

$$\boldsymbol{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Under the assumption that the rotation around the z-axis is a linear transformation, the linear mapping $\boldsymbol{R}$ can be easily verified by validating the mapping on a basis set as $\{\mathbf{e}_{\mathrm{x}} = (1,0,0), \mathbf{e}_{\mathrm{y}} = (0,1,0), \mathbf{e}_{\mathrm{z}} = (0,0,1)\}$. For the basis vectors one gets $\boldsymbol{R}\mathbf{e}_{\mathrm{x}} = (\cos\alpha, \sin\alpha, 0) \equiv \mathbf{e}'_{\mathrm{x}}$, $\boldsymbol{R}\mathbf{e}_{\mathrm{y}} = (-\sin\alpha, \cos\alpha, 0) \equiv \mathbf{e}'_{\mathrm{y}}$ as required by Figure D.2. The vector $\mathbf{e}_{\mathrm{z}}$ is not changed by mapping ($\boldsymbol{R}\mathbf{e}_{\mathrm{z}} = (0,0,1)$). The rotation matrices for the rotations around the x-, and y-axis can be verified in a similar way. The homogenous rotation matrices are:

$$\boldsymbol{R}^{\mathrm{z}}_{\alpha} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{R}^{\mathrm{x}}_{\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{R}^{\mathrm{y}}_{\alpha} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

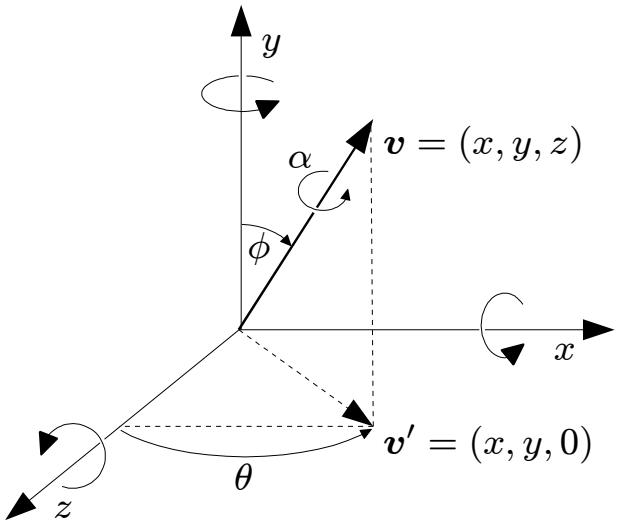


Figure D.3: Rotation around an Arbitrary Rotation Axes. The figure shows an arbitrary rotation axis passing through the origin. The axis is given by the vector $\boldsymbol{v} = (x, y, z)$. The directions of the rotations are indicated for the axis v, and the x-, y-, and z-axis.

A rotation around an arbitrary axis through the origin can be established as a concatenation of several rotation matrices. The axis is given therefore by a vector $\boldsymbol{v} = (x, y, z)$ as shown in Figure D.3. The rotation can be established as a rotation around the y-axis.

The procedure for a point $\boldsymbol{p}$ is the following: 1., the point $\boldsymbol{p}$ is transformed by a rotation which would align the vector $\boldsymbol{v}$ with the y-axis. 2., then the rotation around the y-axis is performed. 3., the transformation of step 1. is undone by using the inverse transformation. The mapping for a 3D point $\boldsymbol{p}$ to the point $\boldsymbol{p}'$ rotated around the axis $\boldsymbol{v}$ is then

$$\boldsymbol{p}' = \underbrace{(\boldsymbol{R}^{\mathrm{x}}_{-\phi}\boldsymbol{R}^{\mathrm{y}}_{-\theta})^{-1}}_{3.}\underbrace{(\boldsymbol{R}^{\mathrm{y}}_{\alpha}}_{2.}\underbrace{(\boldsymbol{R}^{\mathrm{x}}_{-\phi}\boldsymbol{R}^{\mathrm{y}}_{-\theta}\boldsymbol{p}))}_{1.},$$

where the points $\boldsymbol{p}$ and $\boldsymbol{p}'$ are assumed to be given in homogenous coordinates in coherence with the definition of the rotation matrices given above. The angles θ and ϕ (defined in Figure D.3) are given

by

$$\theta = \arctan \frac{x}{y}$$
$$\phi = \arctan \frac{\|\boldsymbol{v}'\|}{z} = \arctan \frac{\sqrt{x^2+y^2}}{z}.$$

Note, if the vector is not always in the first octant, the function $\arctan 2$ has to be used. Finally, the rotation around the axis $\boldsymbol{v}$ can be written as a homogenous matrix:

$$\boldsymbol{R}_{\boldsymbol{v}}^{\alpha} = \boldsymbol{R}_{\theta}^{\mathrm{y}} \boldsymbol{R}_{\phi}^{\mathrm{x}} \boldsymbol{R}_{\alpha}^{\mathrm{y}} \boldsymbol{R}_{-\phi}^{\mathrm{x}} \boldsymbol{R}_{-\theta}^{\mathrm{y}}.$$

D.2.4 Arbitrarily Located Rotation Axes

An arbitrarily located 3D rotation axis $\boldsymbol{j} \equiv (\boldsymbol{p}, \boldsymbol{v})$ which does not (necessarily) intersect the origin can be specified by a point $\boldsymbol{p} \in \mathbb{R}^3$ on the axis and an orientation vector $\boldsymbol{v} \in \mathbb{R}^3$. The rotation of the object point $\boldsymbol{q}$ to a point $\boldsymbol{q}'$ can be established by first translating the point $\boldsymbol{q}$ by $-\boldsymbol{p}$. The rotation is then performed as a rotation around an axis with orientation $\boldsymbol{v}$ which intersects the origin. Finally, the translation is undone. Assuming that $\boldsymbol{q}$ and $\boldsymbol{q}'$ are given in homogenous coordinates, the rotated point $\boldsymbol{q}'$ is then given by

$$\boldsymbol{q}' = \boldsymbol{T}_{\boldsymbol{p}}(\boldsymbol{R}_{\boldsymbol{v}}^{\alpha}(\boldsymbol{T}_{-\boldsymbol{p}}\,\boldsymbol{q})),$$

where α is the rotation angle. The homogenous matrix for this transformation is

$$\boldsymbol{R}_{\boldsymbol{j}}^{\alpha} = \boldsymbol{T}_{\boldsymbol{p}} \boldsymbol{R}_{\boldsymbol{v}}^{\alpha} \boldsymbol{T}_{-\boldsymbol{p}},$$

the transformation $\mathcal{R}_{\boldsymbol{j}}^{\alpha} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ without using homogenous coordinates for $\boldsymbol{q}$ and $\boldsymbol{q}'$, is simply given by $\mathcal{R}_{\boldsymbol{j}}^{\alpha}(\boldsymbol{q}) = \boldsymbol{\iota}^{-1}(\boldsymbol{R}_{\boldsymbol{v}}^{\alpha}\boldsymbol{\iota}(\boldsymbol{q}))$, which is the notation that is used in Section 5.2.1.

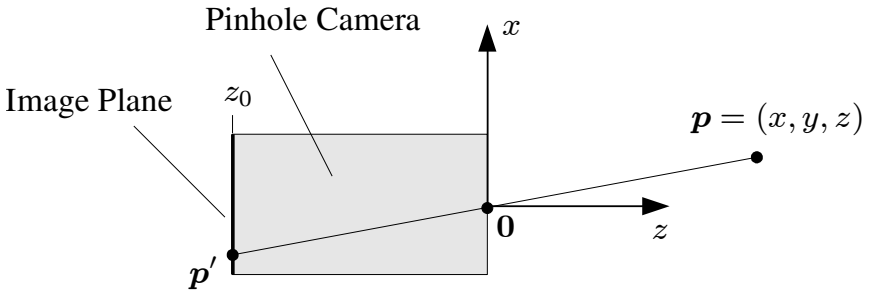


Figure D.4: Pinhole Camera.

D.3 Camera Models

In Figure D.4, a pinhole camera is shown, where the projection center, the pinhole, is identical with the origin of the used reference frame. The image plane lies at $z = z_0$. The y-axis is not shown in the figure. A point $\boldsymbol{p} = (x, y, z)$ is mapped to a point $\boldsymbol{p}' = (x', y', z') = (xz_0/z, yz_0/z, z_0)$ in the reference frame. The z-component of the point $\boldsymbol{p}'$ is neglected in the following, since it is always

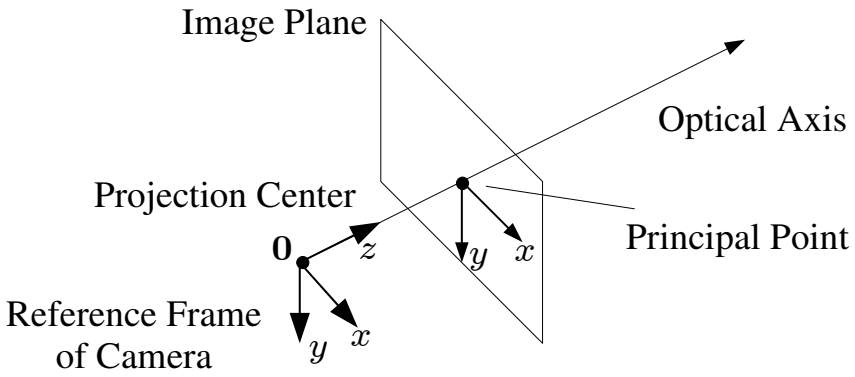


Figure D.5: Virtual Camera Model.

$z' = z_0$. The mapping can then be written as a homogenous matrix $\boldsymbol{P}_0$, as shown in the following equations:

$$\underbrace{\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}}_{\boldsymbol{p}'} = \underbrace{\begin{bmatrix} xz_0/z \\ yz_0/z \\ 1 \end{bmatrix}}_{\boldsymbol{p}'} = \frac{z}{z_0} \underbrace{\begin{bmatrix} x \\ y \\ z_0/z \end{bmatrix}}_{\boldsymbol{p}'} \simeq \underbrace{\begin{bmatrix} x \\ y \\ z_0/z \end{bmatrix}}_{\boldsymbol{p}'} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/z_0 & 0 \end{bmatrix}}_{\equiv \boldsymbol{P}_0} \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{\boldsymbol{p}}.$$

The mapping defined by the matrix $\boldsymbol{P}_0$ is a (non-linear) perspective mapping. However, the image on the back of the box is flipped vertically and horizontally, since $z_0 < 0$. Therefore, it is common to introduce a virtual camera model as shown in Figure D.5, where the image plane lies before the projection center. The camera coordinate system in Figure D.5 is assumed to be identical with the reference frame in which a point $\boldsymbol{p}$ is specified (the world frame). The distance z_0 of the image plane to the projection center is also called the focal length f. A usual digital camera has a CCD chip (charge coupled device), where each cell corresponds finally to a pixel (picture element) of an image. The mapping from the image coordinates x', y' to pixel coordinates x'', y'' is usually described by a 3×3 homogenous matrix, called the K-matrix or calibration matrix.

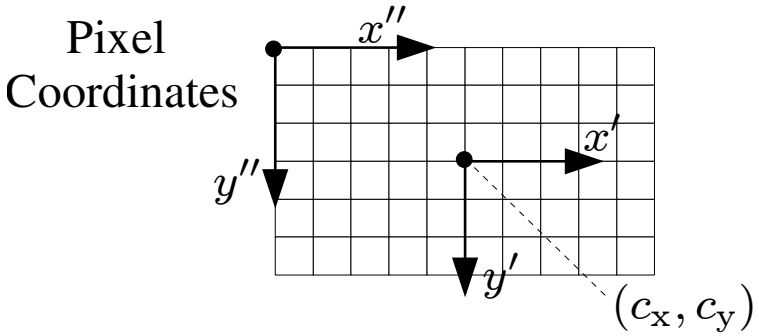


Figure D.6: Pixel Coordinates on a CCD-Chip.

K-Matrix. In Figure D.6, the pixel coordinate system of a CCD chip is shown. The pixel size is assumed to be dx (width) and dy (height) in the image frame. The principal point $(c_\mathrm{x}, c_\mathrm{y})$, given by

the point where the optical axis intersects the image, is here assumed to be given in pixel coordinates. In the settings of Figure D.6, the point

$$\boldsymbol{p}' = (x', y') = (xz_0/z, yz_0/z) \tag{D.1}$$

in the image plain is then mapped to pixel coordinates

$$\boldsymbol{p}' \mapsto \boldsymbol{p}'' = (x'', y'') = (x'\tilde{f}_\mathrm{x}, y'\tilde{f}_\mathrm{y}) + (c_\mathrm{x}, c_\mathrm{y}), \tag{D.2}$$

where $\tilde{f}_\mathrm{x} = 1/dx$ and $\tilde{f}_\mathrm{y} = 1/dy$. In order to include the focal length f into the last mapping (D.2), the value z_0 is replaced by a value $z_0' = 1$ in Equation (D.1), and $\tilde{f}_\mathrm{x}$ and $\tilde{f}_\mathrm{y}$ are replaced by the values $f_x = f \cdot \tilde{f}_x = f/dx$ and $f_y = f \cdot \tilde{f}_y = f/dy$. Hence, the pixel coordinates (x'', y'') are unaltered, since $f = z_0$. The mapping of a point $\boldsymbol{p}' \mapsto \boldsymbol{p}''$ is described by the so called K-matrix [34]:

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_\mathrm{x} & 0 & c_\mathrm{x} \\ 0 & f_\mathrm{y} & c_\mathrm{y} \\ 0 & 0 & 1 \end{bmatrix}}_{\equiv K} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

where the values f_x and f_y can be interpreted as the focal length in units of the pixel width and height. The parameters f_x, f_y, c_x and c_y are the internal camera parameters.

P-Matrix. The whole mapping $\boldsymbol{p} \mapsto \boldsymbol{p}' \mapsto \boldsymbol{p}''$ of a point $\boldsymbol{p}$ given in the reference frame of the camera to a point $\boldsymbol{p}''$ on the image plain (given in pixel coordinates) can be performed by the use of a single homogenous matrix. However, generally one assumes that a reference frame of a camera can be arbitrarily located in a given world frame. The reference frame of the camera is in the following assumed to be rotated by a rotation matrix $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ and translated by a vector $\boldsymbol{t} \in \mathbb{R}^3$. A point of the scene $\tilde{\boldsymbol{p}}$ is then assumed to be given in the world frame. The transformation $\boldsymbol{T}$ of the camera's reference frame can be written as a homogenous matrix:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}. \tag{D.3}$$

Instead of reestablishing the equations for the projection of the point $\tilde{\boldsymbol{p}}$, one can map the point $\tilde{\boldsymbol{p}} = (\tilde{x}, \tilde{y}, \tilde{z})$ first to the point $\boldsymbol{p} = (x, y, z)$ in the camera frame, which is simply a coordinate transformation. The point $\boldsymbol{p}$ can be computed by using inverse mapping of $\boldsymbol{T}$:

$$\underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{\boldsymbol{p}} = \underbrace{\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}}_{\boldsymbol{T}}{}^{-1} \underbrace{\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{p}}} = \underbrace{\begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top\boldsymbol{t} \\ \boldsymbol{0} & -1 \end{bmatrix}}_{\boldsymbol{T}^{-1}} \underbrace{\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{p}}}.$$

The mapping of the point $\tilde{\boldsymbol{p}}$ to the point $\boldsymbol{p}''$ on the image plain (represented in pixel coordinates) is then given by

$$\underbrace{\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix}}_{\boldsymbol{p}''} = \overbrace{\boldsymbol{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\boldsymbol{P}_0 \text{ (for: } z_0'=1)} \underbrace{\begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top\boldsymbol{t} \\ \boldsymbol{0} & -1 \end{bmatrix}}_{\boldsymbol{T}^{-1}}}^{\equiv P} \underbrace{\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{p}}}.$$

The matrix $\boldsymbol{P} \in \mathbb{R}^{3\times4}$ is called projection matrix [34]. The projection matrix can be written as

$$\boldsymbol{P} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top \boldsymbol{t} \end{bmatrix}.$$

If one assumes that the transformation $\boldsymbol{T}$ (Equation (D.3)) instead of $\boldsymbol{T}^{-1}$ describes the mapping of a scene point into the camera reference frame, then the projection matrix can be simply written as $\boldsymbol{P} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}$ as in [34].

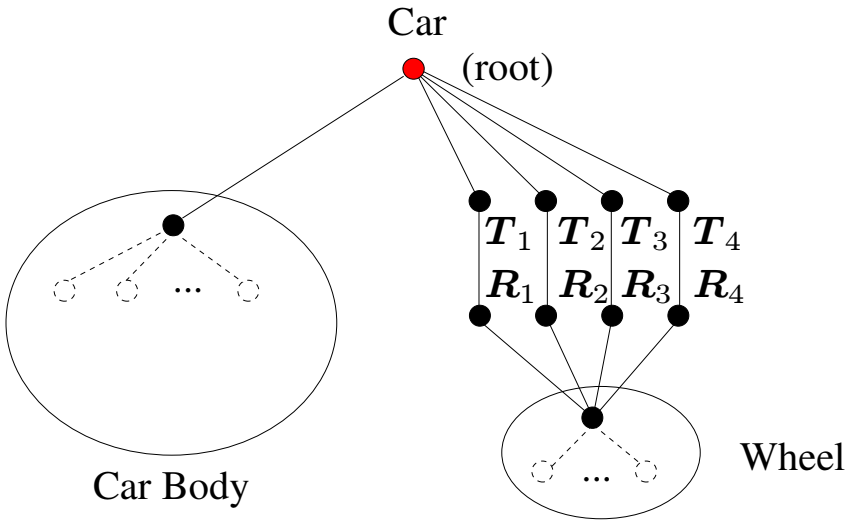


Figure D.7: Example of a Scene Graph. The figure is illustrating a scene graph that describes a car. The idea of the drawing stems from [3].

D.4 Scene Graphs

A scene in computer graphics is generally represented as a graph, called scene graph. An example for a scene graph is given in Figure D.7, which illustrates the representation of a car. The nodes of the graph are transformation nodes and/or geometries. The graph has a designated node, i.e. the root node. The graph can be investigated starting from the root. The graph in the example contains two subgraphs. One is representing the car body, which has again a designated transformation node, which is the root of this geometry. The other subgraph represents a wheel, where the root of this subgraph represents a point on the axis of the wheel. In the scene graph of the car, the wheel is used four times. Usually, each component (as the wheel) is assumed to be defined in a local reference frame. If the car is displayed (rendered), the geometry of the wheel is transformed by the transformation nodes which lie on the path from the root node of the wheel to the root node of the car. This means that a point $\boldsymbol{p}$ of the wheel's geometry is transformed for displaying the i-th wheel ($i \in \{1, \ldots, 4\}$) in the order

$$\boldsymbol{p}' = \boldsymbol{T}_i \boldsymbol{R}_i \boldsymbol{p}.$$

In this equation it is assumed that the transformations and 3D points are defined through homogenous matrices/coordinates. The point $\boldsymbol{p}'$ defines the coordinates of the point $\boldsymbol{p}$ in the reference frame of the car, where the transformation $\boldsymbol{T}_i$ is used to place the wheel at the designated location of the car. The

rotation $\boldsymbol{R}_i$ can be used for the purpose of animation. The modification of rotation angle over time would generate the impression that the wheels are turning. Some scene graphs implementations, as, for example, OpenSG [3], require that the graph is a tree. A tree is a graph structure, which has no cycles. Such a graph has the property that there exist only one path from each node to the root node. Each node has then exactly one parent node, which is the next node on the path to the root node. This simplifies the interpretation of the graph. If one starts at a node, one can easily find the path to the root in a step-by-step approach, where one goes from the current node to the parent node until one reaches the root. The disadvantage of the tree structure is that geometries (defined by a subgraph) cannot be used several times. As an example, for the graph in Figure D.7, one has to duplicate the subgraph of the wheel four times in order to generate an equivalent tree. In order to overcome the replication of geometries, the OpenSG scene graph implementation uses the concept of cores. Each node has a core. But a single core can be used by several nodes. Such a core has, for example, the ability to define even very complex geometries, which are built from a large number of primitives such as triangles, or quadrangles.

Obviously, a scene graph with a tree structure is well suited to describe models of humans or robots for the purpose of animation. In order to articulate the models, one can define transformation nodes between the segments of the modeled embodiment to model the joints of the human/robot. The tree structure defines then also the kinematic chains of the body. Kinematic chains are discussed for a humanoid robot in Section 5.2.1.

Appendix E

Multivariate Gaussian Distributions

The multivariate Gaussian distribution is essential in this work, since it is used in the hidden Markov model (HMM) for representing human movements. The definition of the Gaussian density is given in Section E.1. The essentials to evaluate the density in an efficient way are provided in Section E.2. The displaying of a Gaussian density is discussed in Section E.2.3.

E.1 Gaussian Distribution

The definitions of the Gaussian distribution and the Mahalanobis distance are given in the following. A comprehensive discussion is given in [14]. The multivariate Gaussian density is

$$\mathcal{N}(\hat{\boldsymbol{x}}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{\det(\boldsymbol{\Sigma})^{1/2}}\mathrm{e}^{-\frac{1}{2}d_{\boldsymbol{\Sigma}}(\boldsymbol{\mu},\hat{\boldsymbol{x}})^2}, \tag{E.1}$$

where $\hat{\boldsymbol{x}} \in \mathbb{R}^D$ is a vector. The vector $\boldsymbol{\mu} \in \mathbb{R}^D$ and the matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ are the mean ($\boldsymbol{\mu} = \mathrm{E}[\boldsymbol{x}]$) and the covariance matrix ($\boldsymbol{\Sigma} = \mathrm{cov}[\boldsymbol{x}] = \mathrm{E}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^\top(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])]$, where $\boldsymbol{x}$ is a random vector that is assumed to be distributed according to $\mathcal{N}(\hat{\boldsymbol{x}}|\boldsymbol{\mu},\boldsymbol{\Sigma})$. The Mahalanobis distance $d_{\boldsymbol{\Sigma}}(\cdot,\cdot)$ in Equation (E.1) is

$$d_{\boldsymbol{\Sigma}}(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{y}-\boldsymbol{x}\|_{\boldsymbol{\Sigma}} \qquad (\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^D), \tag{E.2}$$

where

$$\|\boldsymbol{x}\|_{\boldsymbol{\Sigma}} = \sqrt{\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x}} \qquad (\boldsymbol{x} \in \mathbb{R}^D). \tag{E.3}$$

E.2 Cholesky Decomposition and Matrix Inversion

The invserse $\boldsymbol{A}^{-1}$ of a symmetric positive definite matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ can be computed by the means of the Choleky decomposition $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^\top$, where $\boldsymbol{L} = (l_{ij})$ is a lower triangular matrix. The inverse is simply a composition $\boldsymbol{A}^{-1} = \boldsymbol{S}^\top\boldsymbol{S}$ of the inverse $\boldsymbol{S} \equiv \boldsymbol{L}^{-1}$, where $\boldsymbol{S}$ is a lower rectangular matrix. This method of obtaining the inverse is attractive for its numerical stability and speed [113]. Moreover, this method provides the means for calculating the determinant of $\boldsymbol{A}$, since

$$\det(\boldsymbol{A}) = \det(\boldsymbol{L}\boldsymbol{L}^\top) = \det(\boldsymbol{L})\det(\boldsymbol{L}^\top) = \det(\boldsymbol{L})\det(\boldsymbol{L}) = \det(\boldsymbol{L})^2 = \left(\prod\nolimits_i l_{ii}\right)^2. \tag{E.4}$$

In the following two sections, the Cholesky decomposition of $\boldsymbol{A}$ and the composition of the inverse $\boldsymbol{A}^{-1}$ is discussed, both require $\mathcal{O}(D^3)$ arithmetic operations. Hence, the whole procedure of determining $\boldsymbol{S}$ (or $\boldsymbol{A}^{-1}$) and the determinant of $\boldsymbol{A}$ requires $\mathcal{O}(D^3)$ operations, since matrix multiplication and the computation of $\boldsymbol{L}$ and $\boldsymbol{S}$ are in $\mathcal{O}(D^3)$.

E.2.1 Cholesky Decomposition

The Cholesky decomposition of a positive definite symmetric square matrix $\boldsymbol{A} = (a_{ij})_{i,j=1}^{D}$ is a decomposition of the form

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\top}, \tag{E.5}$$

where $\boldsymbol{L} = (l_{ij})$ is a unique lower triangular square matrix of the same size. The elements of $\boldsymbol{L}$ are given by the formulas

$$l_{ii} = \left(a_{ii} - \sum_{j=1}^{i-1} l_{ij}^2 \right)^{\frac{1}{2}} \qquad (i = 1, \ldots, D), \tag{E.6}$$

$$l_{ki} = \frac{1}{l_{ii}} \left(a_{ki} - \sum_{j=1}^{i-1} l_{ij} l_{kj} \right) \qquad (k = i+1, \ldots, D;\ i = 1, \ldots, D), \tag{E.7}$$

which can be derived from the element wise formulation of the matrix product $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\top}$ [113]. The choice of the positive square roots in (E.6) makes the decomposition unique [113].

E.2.2 Composition of the Inverse Matrix

The inverse $\boldsymbol{S} = (s_{ij})_{i,j=1}^{D}$ of a nonsingular lower triangular matrix $\boldsymbol{L} = (l_{ij})_{i,j=1}^{D}$ is given by the formulas

$$s_{ii} = 1/l_{ii} \qquad (i = 1, \ldots, D), \tag{E.8}$$

$$s_{ij} = -\frac{1}{l_{ii}} \left(\sum_{k=1}^{i-1} l_{ik} s_{kj} \right) \qquad (j = 1, \ldots, D;\ i = j+1, \ldots, D), \tag{E.9}$$

which again can be derived from the element wise formulation of the matrix product $\mathbf{I} = \boldsymbol{L}\boldsymbol{S}$ [113]. The inverse $\boldsymbol{S}$ is a lower triangular matrix. The inverse $\boldsymbol{A}^{-1}$ is given by

$$\boldsymbol{A}^{-1} = (\boldsymbol{L}\boldsymbol{L}^{\top})^{-1} = (\boldsymbol{L}^{\top})^{-1}\boldsymbol{L}^{-1} = (\boldsymbol{L}^{-1})^{\top}\boldsymbol{L}^{-1} = \boldsymbol{S}^{\top}\boldsymbol{S}, \tag{E.10}$$

where the matrix rules $(\boldsymbol{C}^{\top})^{-1} = (\boldsymbol{C}^{-1})^{\top}$ and $(\boldsymbol{C}\boldsymbol{D})^{-1} = (\boldsymbol{D}^{-1}\boldsymbol{C}^{-1})$ for nonsingular square matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ have been applied.

E.2.3 Displaying Gaussians by Singular Value Decomposition

For a symmetric matrix $\boldsymbol{A}$, the Schur decomposition exists [138]:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^{T}, \tag{E.11}$$

where $\boldsymbol{S} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ are a diagonal matrix and a orthogonal matrix. If it is assumed that $\boldsymbol{A}$ is a positive definite symmetric matrix (as in the case of a usual covariance matrix), then all eigenvalues of $\boldsymbol{A}$ are positive. In such a case all diagonal elements of the matrix $\boldsymbol{S} = (s_{ij})$ are positive ($s_{ii} > 0$), and one can compute easily the matrices $\boldsymbol{S}^{\frac{1}{2}}$ and $\boldsymbol{S}^{-\frac{1}{2}}$.

Displaying Gaussians. In the following it is shown how the sphere $\mathcal{B}_{\boldsymbol{\Sigma}} = \{\boldsymbol{x}|\ \|\boldsymbol{x}\|_{\boldsymbol{\Sigma}} = 1\}$ can be displayed. For each point $\boldsymbol{x} \in \mathcal{S}$ on the sphere, the Mahalanobis distance of $\boldsymbol{x}$ to the origin $\boldsymbol{0}$ equals 1. It is assumed that $\boldsymbol{\Sigma}$ is a usual (symmetric) covariance matrix. The covariance matrix can be decomposed by using the Schur decomposition: $\boldsymbol{\Sigma} = \boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T$. It is $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^\top$, since $\boldsymbol{Q}$ is a orthogonal square matrix. The following equations are obviously true for a vector $\boldsymbol{x}$:

$$\begin{aligned}\|\boldsymbol{x}\|^2_{\boldsymbol{\Sigma}} = \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} = \boldsymbol{x}^\top(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top)^{-1}\boldsymbol{x} = \boldsymbol{x}^\top(\boldsymbol{Q}^\top)^{-1}\boldsymbol{S}^{-1}\boldsymbol{Q}^{-1}\boldsymbol{x} &= \boldsymbol{x}^\top\boldsymbol{Q}\boldsymbol{S}^{-1}\boldsymbol{Q}^\top\boldsymbol{x} \\ &= \boldsymbol{x}^\top\boldsymbol{Q}(\boldsymbol{S}^{-\frac{1}{2}})^\top\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{Q}^\top\boldsymbol{x} \\ &= \|\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{Q}^\top\boldsymbol{x}\|_2^2\end{aligned}$$

As a consequence, the sphere is

$$\mathcal{B}_{\boldsymbol{\Sigma}} = \{\boldsymbol{x}|\ \|\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{Q}^\top\boldsymbol{x}\|_2 = 1\}.$$

This can be written as

$$\mathcal{B}_{\boldsymbol{\Sigma}} = \{\boldsymbol{x}|\ \|\boldsymbol{D}\boldsymbol{R}\boldsymbol{x}\|_2 = 1\}, \tag{E.12}$$

where $\boldsymbol{D} \equiv \boldsymbol{S}^{-\frac{1}{2}}$ and $\boldsymbol{R} \equiv \boldsymbol{Q}^\top$. If $\det(\boldsymbol{R}) = 1$, then $\boldsymbol{R}$ is a usual rotation matrix. Otherwise, if $\det(\boldsymbol{R}) = -1$, one can multiply one (arbitrary) row of the matrix $\boldsymbol{R}$ with -1. Obviously, this does not change the sphere $\mathcal{B}$, since the multiplication of the row of the matrix $\boldsymbol{R}$ only changes the sign of one component of the vector $\boldsymbol{y} = \boldsymbol{D}\boldsymbol{R}\boldsymbol{x}$ and does not change the value of $\|\boldsymbol{D}\boldsymbol{R}\boldsymbol{x}\|_2$. However, this procedure ensures that $\det(\boldsymbol{R}) = 1$ such that $\boldsymbol{R}$ is a usual rotation matrix. The sphere in Equation (E.12) can be written as

$$\begin{aligned}\mathcal{B}_{\boldsymbol{\Sigma}} &= \{\boldsymbol{x}|\ \exists\boldsymbol{y} : \|\boldsymbol{y}\|_2 = 1, \boldsymbol{y} = \boldsymbol{D}\boldsymbol{R}\boldsymbol{x}\}, &\text{(E.13)}\\ &= \{\boldsymbol{x}|\ \exists\boldsymbol{y} : \|\boldsymbol{y}\|_2 = 1, \boldsymbol{x} = \boldsymbol{R}^\top\boldsymbol{D}^{-1}\boldsymbol{y}\}, &\text{(E.14)}\end{aligned}$$

Hence, the surface of the ellipsoid $\mathcal{B}_{\boldsymbol{\Sigma}}$ is only the scaled and rotated surface of the ball $\mathcal{B} = \{\boldsymbol{x}; \|\boldsymbol{x}\|_2 = 1\}$. To display the ellipsoid $\mathcal{B}_{\boldsymbol{\Sigma}}$, one can display the ball $\mathcal{B} = \{\boldsymbol{x}; \|\boldsymbol{x}\|_2 = 1\}$ scaled by the diagonal matrix $\boldsymbol{D}^{-1}$ and rotated by the matrix $\boldsymbol{R}^\top$. In the 2D case, the ball $\mathcal{B}$ and the sphere $\mathcal{B}_{\boldsymbol{\Sigma}}$ are a circle and an ellipse.

References

[1] Camera Calibration Toolbox for Matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/, 2011. 128

[2] Nvidia's CUDA Programming Framework. http://www.nvidia.com/object/cuda_home_new.html, 2011. 130

[3] OpenSG. http://www.opensg.org, 2011. 128, 191, 192

[4] C. Achard, X. Qu, A. Mokhber, and M. Milgram. Action recognition with semi-global characteristics and hidden markov models. In *ACIVS'07: Proceedings of the 9th international conference on Advanced concepts for intelligent vision systems*, pages 274–284, Berlin, Heidelberg, 2007. Springer-Verlag. 20

[5] M. S. Arulampalam, S. Maskell, N. J. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50 (2):174–188, Feb. 2002. 177, 178, 179

[6] T. Asfour, F. Gyarfas, P. Azad, and R. Dillmann. Imitation learning of dual-arm manipulation tasks in humanoid robots. In *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, pages 40–47, Genoa, Italy, Dec. 2006. 15

[7] T. Asfour, K. Welke, A. Ude, P. Azad, J. Hoeft, and R. Dillmann. Perceiving objects and movemetns to generate actions on a humanoid robot. In *Proc. of International Conference on Robotics and Automation (ICRA), Workshop: From features to actions – Unifying perspectives in compnational and robot vision*, Rome, Italy, April 2007. 119

[8] P. Azad, T. Asfour, and R. Dillmann. Toward an unified representation for imitation of human motion on humanoids. In *IEEE International Conference on Robotics and Automation*, pages 2558 –2563, Apr. 2007. 15, 93, 166

[9] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970. 33

[10] A. Billard, S. Calinon, and F. Guenter. Discriminative and adaptive imitation in uni-manual and bi-manual tasks. *Robotics and Autonomous Systems*, 54(5):370–384, 2006. 15

[11] J. A. Bilmes. What hmms can do. Technical Report UWEETR-2002-0003, University of Washington, Department of Electrical Engineering, 2002. https://www.ee.washington.edu/techsite/papers/refer/UWEETR-2002-0003.html. 29, 33, 34

[12] J. A. Bilmes. What hmms can do. *IEICE - Transactions on Information and Systems*, E89-D (3):869–891, March 2006. 33

[13] J. A. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. www.seanborman.com/publications/EM_algorithm.pdf, 1998. 34, 36

[14] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, Aug. 2006. 29, 33, 34, 35, 36, 40, 177, 193

[15] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1395 –1402, Oct. 2005. 20

[16] A. Bobick. Movement, activity, and action: The role of knowledge in the perception of motion. *Royal Society Workshop on Knowledge-based Vision in Man and Machine*, B-352:1257–1265, 1997. 11

[17] A. F. Bobick, J. W. Davis, I. C. Society, and I. C. Society. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:257–267, 2001. 20

[18] S. Borman. *The Expectation Maximization Algorithm – A short tutorial*, July 2004. 29, 33, 34

[19] J. Bray. Markerless based human motion capture: A survey. Technical report, Vision and VR Group Dept Systems Engineering, Brunel University, Uxbridge UB8 3PH, 2003. 16, 17

[20] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Computer Vision and Pattern Recognition*, pages 8–15, 1998. 17, 121

[21] M. A. Brubaker, L. Sigal, and D. J. Fleet. Video-based people tracking. In *Handbook on Ambient Intelligence and Smart Environments*. Springer Verlag, 2009. 16

[22] D. N. Bub and M. E. J. Masson. Gestural knowledge evoked by objects as part of conceptual representations. *Aphasiology*, 20(9):1112 – 1124, Sept. 2006. 119

[23] S. Calinon, F. Guenter, and A. Billard. On learning, representing and generalizing a task in a humaniod robot. *IEEE Trans. on Systems, Man and Cybernetics*, 37(2):286–298, 2007. 15

[24] O. Cappe, S. J. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5):899–924, May 2007. 177, 178

[25] J. Chen, M. Kim, Y. Wang, and Q. Ji. Switching gaussian process dynamic models for simultaneous composite motion tracking and recognition. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2655 –2662, 20-25 2009. 16, 18, 166

[26] J. Deutscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 126–133, 13–15 June 2000. 17, 18, 119, 121, 128, 131

[27] T. Duong, H. Bui, D. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 838 – 845 vol. 1, 20-25 2005. 21

[28] A. Elgammal and C.-S. Lee. Inferring 3d body pose from silhouettes using activity manifold learning. *Computer Vision and Pattern Recognition*, 2:681–688, 2004. 19, 121

[29] Q. Fan, R. Bobbitt, Y. Zhai, A. Yanagawa, S. Pankanti, and A. Hampapur. Recognition of repetitive sequential human activity. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 943 –950, 20-25 2009. 16

[30] M. Gales. Maximum likelihood linear transformations for hmm-based speech recognition. Technical report, Cambridge University, Cambridge, UK, 1997. 59

[31] J. Gall, J. Potthoff, C. Schnörr, B. Rosenhahn, and H.-P. Seidel. Interacting and annealing particle filters: Mathematics and a recipe for applications. *J. Math. Imaging Vis.*, 28(1):1–18, 2007. 18, 121

[32] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1746 –1753, 20-25 2009. 17, 18

[33] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F Radar and Signal Processing*, 140(2):107–113, April 1993. 177, 178

[34] D. Grest. *Marker-Free Human Motion Capture in Dynamic Cluttered Environments from a Single View-Point*. PhD thesis, Multimedia Information Processing Group, Reinhard Koch, Christian-Albrechts-Universität zu Kiel (CAU), Kiel, Germany, 2007. 16, 17, 183, 190, 191

[35] D. Grest, D. Herzog, and R. Koch. Human model fitting from monocular posture images. In *Proceedings of Vision, Modeling, and Visualization 2005*, pages 107–114, Erlangen, Germany, Nov. 2005. 59, 63

[36] D. Grest, J. Woetzel, and R. Koch. Nonlinear body pose estimation from depth images. In *Proc. of 27th Annual Symposium of the German Association for Pattern Recognition (DAGM) 2005*, pages 285–292, Vienna, Austria, Sept. 2005. 17

[37] G. Guerra-Filho and Y. Aloimonos. A sensory-motor language for human activity understanding. In *Proc. 6th IEEE-RAS International Conference on Humanoid Robots*, pages 69–75, 4–6 Dec. 2006. 120, 122

[38] G. Guerra-Filho and Y. Aloimonos. A language for human action. *Computer*, 40(5):42–51, May 2007. 11, 120, 122

[39] S. Günter and H. Bunke. Optimizing the number of states, training iterations and gaussians in an HMM-based handwritten word recognizer. In *Seventh International Conference on Document Analysis and Recognition*, volume 01, pages 472–476, Los Alamitos, CA, USA, 2003. IEEE Computer Society. 31

[40] A. Gupta and L. S. Davis. Objects in action: An approach for combining action understanding and object perception. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR '07*, pages 1–8, 17–22 June 2007. 119

[41] A. Gupta, A. Mittal, and L. S. Davis. Constraint integration for efficient multiview pose estimation with self-occlusions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (3):493–506, 2008. 18, 121

[42] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 183

[43] N. Hasler, B. Rosenhahn, T. Thormahlen, M. Wand, J. Gall, and H.-P. Seidel. Markerless motion capture with unsynchronized moving cameras. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 224 –231, 20-25 2009. 17

[44] H. B. Helbig, M. Graf, and M. Kiefer. The role of action representations in visual object recognition. *Experimental Brain Research*, 174(2):221–228, 04 2006. 119

[45] D. Herzog and V. Krüger. Parametric HMMs for movement recognition and synthesis. In *Proceedings of IEEE's NTAV/SPA*, pages 9–15, 2008. 23

[46] D. Herzog and V. Krüger. Recognition and synthesis of human movements by parametric HMMs. In *Book-chapter, in Springers' Lecture Notes in Computer Science (LNCS), 2009, ISSN: 0302-9743*, pages 148–168, 2009. 23

[47] D. Herzog, V. Krüger, and D. Grest. Exemplar-based parametric hidden Markov models for recognition and synthesis of movements. In *Proceedings of Vision, Modeling, and Visualization 2007*, pages 253–261, Saarbrücken, Germany, Nov. 2007. 23, 24

[48] D. Herzog, V. Krüger, and D. Grest. Parametric hidden Markov models for recognition and synthesis of movements. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 1, pages 163–172, Leeds, UK, sep 2008. 23, 25, 59

[49] D. Herzog, V. Krüger, and D. Grest. Parametric hidden markov models for recognition and synthesis of movements. Technical Report 1, CVMI, Aalborg University, 2008. 23

[50] D. Herzog, A. Ude, and V. Krüger. Motion imitation and recognition using parametric hidden Markov models. In *Proc. 8th IEEE-RAS International Conference on Humanoid Robots 2008*, pages 339–346, Daejeon, Korea, South, Dec. 2008. 23, 25, 94

[51] D. L. Herzog and V. Krüger. Tracking in action space. In *ECCV 2010 (Workshop on Human Motion)*. (downloadable from the workshop homepage: `http://humanmotion.rutgers.edu/program_final.htm`), 2010. 23, 25, 123

[52] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of honda humanoid robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1321 –1326 vol.2, may. 1998. 15

[53] B. Hommel and B. Elsner. Acquisition, representation, and control of action. In E. Morsella, J. A. Bargh, and P. M. Gollwitzer, editors, *The Psychology of Action*, volume 2. Oxford University Press, 2006. 12

[54] X. Huang, Y. Ariki, and M. Jack. *Hidden Markov Models for Speech Recognition*. Columbia University Press, New York, NY, USA, 1990. 29

[55] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 1398–1402, 2002. 16, 165

[56] T. Inamura, I. Toshima, H. Tanie, and Y. Nakamura. Embodied symbol emergence based on mimesis theory. *Int. J. Robotics Research*, 23(4-5):363–377, 2004. 12, 13, 16, 43, 165

[57] M. Isard and A. Blake. CONDENSATION: Conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998. 121, 123, 131, 177, 179, 180

[58] Y. A. Ivanov and A. F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):852–872, 2000. 10, 21, 120

[59] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991. 13

[60] O. C. Jenkins and M. J. Mataric. Deriving action and behavior primitives from human motion. In *International Conference on Intelligent Robots and Systems*, pages 2551–2556, 2002. 120

[61] S. B. Kang and K. Ikeuchi. Toward automatic robot instruction from perception-mapping human grasps to manipulator grasps. *Robotics and Automation, IEEE Transactions on*, 13(1):81 –95, feb. 1997. 15

[62] E. Keogh and M. Pazzani. Derivative dynamic time warping. In *Proceedings of the 1st SIAM International Conference on Data Mining (SDM'2001)*, 2001. 66, 171, 172, 173

[63] T.-K. Kim and R. Cipolla. Canonical correlation analysis of video volume tensors for action categorization and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:1415–1428, 2009. 20

[64] H. Kjellström, J. Romero, D. M. Mercado, and D. Kragic. Simultaneous visual recognition of manipulation actions and manipulated objects. In *Proc. of 10th European Conference on Computer Vision (ECCV)*, volume 2, pages 336–349, 2008. 119

[65] V. Krüger, D. Kragic, A. Ude, and C. Geib. The meaning of action: A review on action recognition and mapping. *Advanced Robotics*, 21(13):1473–1501, 2007. 11, 15, 19, 119

[66] V. Krüger, D. L. Herzog, S. Baby, A. Ude, and D. Kragic. Learning actions from observations. *Robotics and Automation Magazine, IEEE*, 17(2):30–43, June 2010. 13, 23

[67] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *Robotics and Automation, IEEE Transactions on*, 10(6):799 –822, dec. 1994. 15

[68] I. Laptev. On space-time interest points. *International Journal of Computer Vision*, 64(2-3): 107–123, 2005. 20

[69] N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *NIPS*, page 2004, 2004. 19

[70] M. W. Lee and R. Nevatia. Human pose tracking in monocular sequence using multilevel structured models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 31(1):27–38, 2009. 18, 119, 121

[71] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech & Language*, 9(2):171–185, April 1995. 59

[72] C.-W. Lin and Z.-H. Ling. Automatic fall incident detection in compressed video for intelligent homecare. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 1172 –1177, aug. 2007. 16

[73] Z. Lu, M. A. Carreira-Perpinan, and C. Sminchisescu. People tracking with the laplacian eigenmaps latent variable model. In *NIPS*, 2007. 19

[74] J. luc Gauvain and C. hui Lee. Map estimation of continuous density hmm: Theory and applications. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 185–190. Morgan Kaufmann, 1992. 31

[75] F. Lv and R. Nevatia. Recognition and segmentation of 3-d human action using hmm andmulticlass adaboost. In *European Conference on Computer Vision*, volume 4, pages 359–372, 2006. 119

[76] T. Ma, X. Yang, and L. J. Latecki. Boosting chamfer matching by learning chamfer distance normalization. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *Computer Vision – ECCV 2010*, volume 6315 of *Lecture Notes in Computer Science*, pages 450–463. Springer, 2010. 133

[77] J. MacCormick. *Probabilistic modelling and stochastic algorithms for visual localisation and tracking*. PhD thesis, University of Oxford, 2000. 18, 177

[78] H. Miyamoto and M. Kawato. A tennis serve and upswing learning robot based on bi-directional theory. *Neural Networks*, 11(7-8):1331 – 1344, 1998. 15, 74, 165

[79] T. B. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding (CVIU)*, 81(3):231–268, 2001. 16, 17

[80] T. B. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding (CVIU)*, 104(2-3):90–126, November 2006. 11, 16, 17, 19, 121

[81] H. Moon, R. Chellappa, and A. Rosenfeld. 3D object tracking using shape-encoded particle propagation. In *Proc. Eighth IEEE International Conference on Computer Vision (ICCV) 2001*, volume 2, pages 307–314, July 2001. 18, 119, 121

[82] K. Murphy. Hidden semi-Markov models (HSMMs). http://www.cs.ubc.ca/~murphyk/Papers/segment.pdf, 2002. unpublished notes. 38

[83] K. Murphy. Fitting a conditional gaussian distribution. Technical report. www.cs.ubc.ca/~murphyk/Papers/learncg.pdf, 1998. 41

[84] K. Murphy. Documentation to the Matlab HMM Toolbox. http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm_usage.html, 2011. 41

[85] H.-H. Nagel. From image sequences towards conceptual descriptions. *Image Vision Comput.*, 6:59–74, May 1988. 11

[86] D. Newton, D. Engquist, and J. Bois. The objective basis of behavior unit. *Journal of Personality and Social Psychology*, 35(12):847–862, 1977. 11, 165

[87] J. C. Niebles, H. Wang, and L. Fei-Fei. Unsupervised learning of human action categories using spatial-temporal words. In *BMVC*, 2006. 20

[88] T. Oates, L. Firoiu, and P. R. Cohen. Using dynamic time warping to bootstrap HMM-based clustering of time series. In *Sequence Learning: Paradigms, Algorithms, and Applications*, pages 35–52, 2001. 45

[89] V. Pavlovic, J. M. Rehg, T.-J. Cham, and K. P. Murphy. A dynamic bayesian network approach to figure tracking using learned dynamic models. *Computer Vision, IEEE International Conference on*, 1:94, 1999. 17, 18, 166

[90] R. Plankers and P.Fua. Model-based silhouette extraction for accurate people tracking. In *European Conference on Computer Vision, Copenhagen, Denmark*, May 2002. 17

[91] R. Polana and A. Nelson. Detection and recognition of periodic, nonrigid motion. *International Journal of Computer Vision*, 23:261–282, 1997. 20

[92] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2):257–285, Feb. 1989. 29, 30, 31, 32, 33, 37, 38, 39, 40, 41, 43

[93] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, Jan. 1986. 29, 40, 45

[94] P. R. Ramana, D. Grest, and V. Krüger. Human action recognition in table-top scenarios : An hmm-based analysis to optimize the performace. In *Proceedings of Computer Analysis of Images and Patterns*, pages 101–108, Vienna, Austria, 2007. 90

[95] C. Rao, A. Yilmaz, and M. Shah. View-invariant representation and recognition of actions. *Int. J. Comput. Vision*, 50(2):203–226, 2002. 19

[96] L. Raskin, E. Rivlin, and M. Rudzsky. Using gaussian process annealing particle filter for 3d human tracking. *EURASIP Journal on Advances in Signal Processing*, 2007. 18

[97] T. M. Rath and R. Manmatha. Lower-Bounding of Dynamic Time Warping Distances for Multivariate Time Series. Technical Report MM-40, CIIR, 2003. 171, 172

[98] H. Ren, G. Xu, and S. Kee. Subject-independent natural action recognition. In *Proc. Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 523–528, May 2004. 119

[99] J. Rett and J. Dias. Gesture recognition using a marionette model and dynamic bayesian networks. In *ICIAR 2006. LNCS*, pages 69–80. Springer, 2006. 21

[100] M. Riley and C. G. Atkeson. Methods for motion generation and interaction with a humanoid robot: Case studies of dancing and catching. In *Proc. 2000 Workshop on Interactive Robotics and Entertainment, Robotics Inst., Carnegie Mellon Univ*, pages 35–42, 2000. 15

[101] G. Rizzolatti and L. Craighero. The mirror-neuron system. *Annual Reviews of Neuroscience*, 27:169–192, 2004. 11, 165

[102] G. Rizzolatti, L. Fogassi, and V. Gallese. Parietal cortex: from sight to action. *Current Opinion in Neurobiology*, 7(4):562–567, Aug. 1997. 11, 120, 165, 166

[103] G. Rizzolatti, L. Fogassi, and V. Gallese. Neurophysical mechanisms underlying the understanding and imitation of action. *Nature Reviews: Neuroscience*, 2(9):661–670, Sept. 2001. 11, 120, 165, 166

[104] J. Romero, H. Kjellstro andm, and D. Kragic. Hands in action: real-time 3d reconstruction of hands in interaction with objects. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 458 –463, may. 2010. 16

[105] R. Rosales, V. Athitsos, L. Sigal, and S. Sclaroff. 3d hand pose reconstruction using specialized mappings. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 378 –385 vol.1, 2001. 16

[106] M. J. Russel and A. E. Cook. Experimental evaluation of duration modeling techniques for automatic speech recognition. In *Proceedings of IEEE ICASSP. ICASSP 87, Dallas, USA, 1997*, pages 2376–2379, 1997. 42

[107] J. Saboune and F. Charpillet. Using interval particle filtering for marker less 3d human motion capture. In *Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on*, pages 7 pp. –627, 16-16 2005. 18

[108] S. Salvador and P. Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *KDD Workshop on Mining Temporal and Sequential Data*, pages 70–80, 2004. 171, 172

[109] Sanmohan and V. Krueger. *Lecture Notes in Computer Science*, volume LNCS 5575 of *Image Analysis*. Springer Berlin / Heidelberg, 2009. 13

[110] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Science*, 3 (6):233–242, June 1999. 10, 11

[111] S. Schaal, J. Peters, J. Nakanishi, and A. J. Ijspeert. Learning movement primitives. In P. Dario and R. Chatila, editors, *Robotics Research: The Eleventh International Symposium*, pages 561–572, Berlin, Heidelberg, 2005. Springer. 15, 16, 165

[112] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32 – 36 Vol.3, aug. 2004. 20

[113] T. Seaks. Syminv: An algorithm for the inversion of a positive definite matrix by the cholesky decomposition. *Econometrica*, 40(5):961–962, 1972. 193, 194

[114] H. Sidenbladh, M. J. Black, and D. J. Fleet. Stochastic tracking of 3d human figures using 2d image motion. In *In European Conference on Computer Vision*, pages 702–718, 2000. 17, 18

[115] H. Sidenbladh, F. De la Torre, and M. Black. A framework for modeling the appearance of 3d articulated figures. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 368 –375, 2000. 17

[116] H. Sidenbladh, M. J. Black, and L. Sigal. Implicit probabilistic models of human motion for synthesis and tracking. In *European Conference on Computer Vision*, pages 784–800, 2002. 18, 119, 121

[117] C. Sminchisescu and B. Triggs. Covariance scaled sampling for monocular 3d body tracking. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2001*, volume 1, pages 447–454, 2001. 18, 119, 121

[118] C. Smincisescu and B. Tiggs. Kinematic jump processes for monocular 3d human tracking. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 18-20 2003. 18

[119] R. Souvenir and J. Babbs. Learning the viewpoint manifold for action recognition. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–7, 2008. 20

[120] T. E. Starner and A. Pentland. Visual recognition of american sign language using hidden markov models. In *IEEE International Workshop on Automatic Face and Gesture Recognition*, pages 189–194, 1995. 20

[121] A. Sundaresan and R. Chellappa. Markerless motion capture using multiple cameras. In *Computer Vision for Interactive and Intelligent Environment, 2005*, pages 15 – 26, 17-18 2005. 17, 129

[122] G. W. Taylor, L. Sigal, D. J. Fleet, and G. E. Hinton. Dynamic binary latent variable models for 3d human pose tracking. In *Computer Vision and Pattern Recognition, 2010. CVPR 2010. IEEE Conference on*, 2010. 19

[123] A. Thayananthan, B. Stenger, P. H. S. Torr, and R. Cipolla. Shape context and chamfer matching in cluttered scenes. In *Proc. CVPR*, volume I, pages 127–133, Madison, WI, June 2003. 133

[124] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998. 133

[125] P. Turaga, R. Chellappa, V. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1473 –1488, nov. 2008. 11, 19

[126] A. Ude, M. Riley, B. Nemec, A. Kos, T. Asfour, and G. Cheng. Synthesizing goal-directed actions from a library of example movements. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 115 –121, nov. 2007. 15, 117, 165

[127] A. Ude, M. Riley, B. Nemec, A. Kos, T. Asfour, and G. Cheng. Goal-directed action synthesis from a library of example movements. In *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Pittsburgh, Pennsylvania, Dec. 2007. 12, 15, 16, 165

[128] R. Urtasun and P. Fua. 3d human body tracking using deterministic temporal motion models. In *ECCV (3)*, pages 92–106, 2004. 18, 121

[129] R. Urtasun, D. J. Fleet, A. Hertzmann, and P. Fua. Priors for people tracking from small training sets. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 403–410, Washington, DC, USA, 2005. IEEE Computer Society. 18, 166

[130] R. Urtasun, D. Fleet, and P. Fua. 3d people tracking with gaussian process dynamical models. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 238 – 245, 17-22 2006. 18, 166

[131] J. Wang, D. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):283 –298, feb. 2008. 18, 19, 121

[132] Y. Wang, L. Xie, Z.-Q. Liu, and L.-Z. Zhou. The SOMN-HMM model and its application to automatic synthesis of 3D character animations. *IEEE International Conference on Systems, Man and Cybernetics*, 6:4948–4952, 2006. 44, 59, 74, 89, 165

[133] D. Weinland, R. Ronfard, and E. Boyer. Free viewpoint action recognition using motion history volumes. *Computer Vision and Image Understanding*, 2006. 20

[134] A. D. Wilson and A. F. Bobick. Parametric hidden Markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 21(9):884–900, 1999. 10, 13, 14, 16, 24, 57, 58, 59, 60, 61, 62, 63, 71, 119, 120

[135] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Porr. Cognitive agents – a procedural perspective relying on the predictability of object-action-complexes (oacs). *Robotics and Autonomous Systems*, 57(4):420 – 432, 2009. 12

[136] T. Xiang and S. Gong. Beyond Tracking: Modelling Action and Understanding Behavior. *International Journal of Computer Vision*, 67(1):21–51, 2006. 119

[137] A. Yilmaz and M. Shah. Actions sketch: a novel action representation. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 984 – 989 vol. 1, June 2005. 20

[138] Q. Zhang, Y. Niu, Y. Yang, and W. Niu. Neurodynamic analysis for symmetric schur decomposition problems. In *International Conference on Computational Intelligence and Security*, pages 485 –489, Dec. 2007. 194