ALBERT-LUDWIGS-UNIVERSITY FREIBURG
FACULTY OF APPLIED SCIENCES

Department of Computer Sciences
Autonomous Intelligent Systems Lab
Prof. Dr. Wolfram Burgard

Techniques for the Imitiation of Manipulative Actions by Robots

Diploma Thesis

Author: Clemens Eppner

Submitted on: November 21, 2008

Supervisors: Prof. Dr. Wolfram Burgard
Juniorprof. Dr. Maren Bennewitz
Dr. Cyrill Stachniss
M.Sc. Jürgen Sturm

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

(Clemens Eppner)
Freiburg i. Br., den 21. November 2008

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich im Verlauf der Diplomarbeit hilfreich unterstützt haben. Besonders danken möchte ich Prof. Dr. Wolfram Burgard dafür, dass er mir diese Arbeit ermöglicht hat, und für seinen Rat bei offenen Fragen. Weiterhin möchte ich mich bei Dr. Maren Bennewitz, Dr. Cyrill Stachniss und Jürgen Sturm für ihre exzellente Unterstützung bedanken. Zu guter Letzt danke ich allen Laborkumpanen, die mir auch in den dunkelsten Stunden der vergangenen Monate beistanden - mittags in der Mensa.

Abstract

In this thesis, we present an approach that allows a robot to observe, generalize, and reproduce tasks observed from multiple demonstrations. Motion capture data is recorded in which a human instructor manipulates a set of objects. In our approach, we learn relations between body parts of the demonstrator and objects in the scene. These relations result in a generalized task description.

The problem of learning and reproducing human actions is formulated using a dynamic Bayesian network (DBN). The posteriors corresponding to the nodes of the DBN are estimated by observing objects in the scene and body parts of the demonstrator. To reproduce a task, we seek for the maximum-likelihood action sequence according to the DBN. We additionally show how further constraints can be incorporated online, for example, to robustly deal with unforeseen obstacles. Experiments carried out with a real 6-DoF robotic manipulator as well as in simulation show that our approach enables a robot to reproduce a task carried out by a human demonstrator. Our approach yields a high degree of generalization illustrated by performing a pick-and-place, a pouring, and a whiteboard cleaning task.

Zusammenfassung

In dieser Arbeit stellen wir einen Ansatz vor, der es einem Roboter ermöglicht, manipulative Aufgaben anhand der Beobachtung eines menschlichen Vorbilds erfolgreich zu lösen. Dazu werden Bewegungsdaten mehrere Demonstrationen der selben Fähigkeit aufgenommen. Unsere Methode lernt die räumlichen Beziehungen zwischen Objekten in der Szene und der Hand des menschlichen Lehrers sowie dessen Armkonfigurationen. Darauf aufbauend kann eine verallgemeinernde Beschreibung der Aufgabe gewonnen werden, die es erlaubt, auch für Aufgaben in leicht variierenden Kontexten eingesetzt zu werden.

Wir formulieren das Imitationsproblem mit Hilfe eines dynamischen Bayes'schen Netzes (DBN). Die Wahrscheinlichkeiten der einzelnen Knoten werden aufgrund der Demonstrationsobservationen geschätzt. Zur Nachahmung wählen wir die Aktionsreihenfolge, die der höchsten Wahrscheinlichkeit im DBN entspricht. Weiterhin zeigen wir, wie die Einhaltung zusätzlicher Bedingungen die Imitation beeinflußen kann, beispielsweise um Hindernissen auszuweichen. Unsere Experimente wurden sowohl mit einem simulierten als auch echten Roboterarm mit sechs Freiheitsgraden durchgeführt. Sie veranschaulichen die Wirksamkeit und das Generalisierungsvermögen unseres Ansatzes anhand dreier Aufgaben: das Bewegen eines Objekts, das Einschenken aus einer Flasche, sowie das Wischen einer Tafel.

Contents

1 Introduction

Building truly autonomous robot systems that are able to act in unstructured environments is a challenging and active research domain. The reasons to operate a robot without human guidance are manifold. Robotic manipulators for example are hard to handle by teleoperation and possibly require an expert to be in use due to the high degrees of freedom they exhibit. Regarding the ongoing trend of emerging robotic applications in domestic environments and other daily life domains [3], an intuitive programming method is inevitable. Another reason, solutions such as restrictive hard-wiring or cumbersome manual preprogramming of every behavior are avoided, is because of the need for more versatile and adaptable robot platforms.

Hence, the transfer of a skill onto a robot must be realized using a learning approach. The most general approach to learning control is reinforcement learning [59]. For each possible state it finds the optimal action by maximizing an expected scalar reward. Problems arise when the state-action space becomes too huge to be explored efficiently. In case of a movement system the curse of dimensionality strikes as the number of actions that can be taken in every state is exponential in the number of degrees of freedom.

A less tedious learning approach named "Programming by Demonstration" (PbD) or in a more biological context "Learning by Imitation" [7, 5] exploits the observation of an exemplary solution by applying it to a different embodiment and problem setup. This thesis introduces a robot system solving manipulation tasks on the basis of imitation learning.

1.1 Learning by Imitation

To illustrate the learning of a manipulative action by imitation, consider the example shown in Figure 1.1. First, a human teacher demonstrates his skill of cleaning a whiteboard while being observed by a robot. Afterwards, the robot is able to accomplish the same task using his newly acquired knowledge about whiteboard cleaning. With further demonstrations being presented, the robot will generalize its task knowledge. Roughly speaking, the robot is enabled to capture some of the intentions underlying the instructor's performance. In our illustrating example, the robot is able to clean a whiteboard differing in size and position.

Learning by Imitation first appeared as the PbD paradigm at the beginning of the 1980's in software development research [15] where it qualified end-users to configure software without language programming skills. Simultaneously, it attracted the robotics community primarily due to the high costs caused by development and

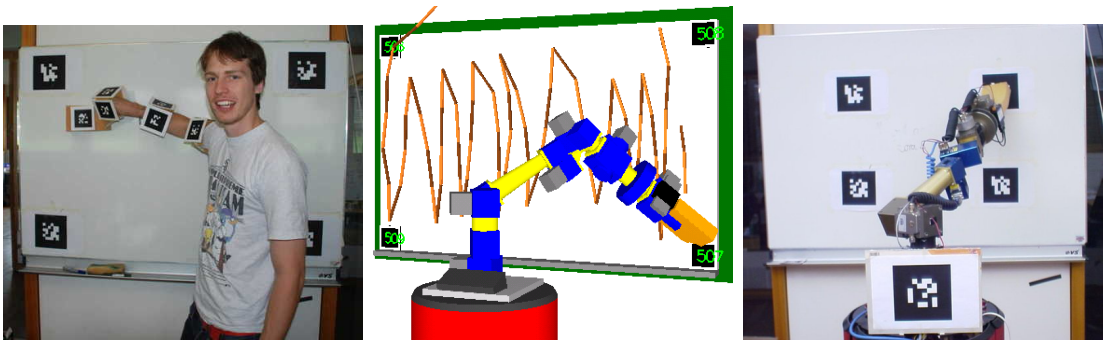

Figure 1.1: Learning the whiteboard cleaning task. **Left**: The human instructor demonstrates how to clean the whiteboard using visual motion capturing. **Middle**: After the first demonstration, the robot can replay the recorded trajectory. **Right**: After several demonstrations, the robot can generalize the task and reproduce it under changed conditions, for example, on a whiteboard with different size and position.

maintenance of manufacturing robots working on assembly lines. Early approaches to PbD involved simple playback methods of recorded manual control sequences or motion plans generated by symbolic reasoning [19]. The development of humanoid robots made imitation learning techniques even more appealing as differences between the embodiment of teacher and student vanished [54]. Nowadays, PbD is a highly interdisciplinary research field ranging from human robot interaction (HRI) through machine learning to computer vision and motor control.

In the following we argue for the benefits of adopting a learning by imitation approach in this thesis:

Imitation makes learning faster. Motion behavior often deals with complex search spaces. Learning by imitation avoids looking for a solution in the whole space. An observed good solution is employed as a starting point for the search, serving as an local optimum. Alternatively, when observing a bad solution, it can be removed from the search space. In contrast to traditional learning algorithms, imitation is a powerful tool to accelerate learning in intelligent systems.

Imitation learning helps understanding the coupling of perception and action. Imitation is a natural biologically grounded mechanism. Behavioral and cognitive sciences show that the human ability to imitate, e.g., manual gestures [38], starts at an very early age. Indeed, even neonates [39], only a few hours old, are capable of imitating facial gestures. Moreover, the assumption of an innate human imitation system is supported by findings in neuroscience. So-called "mirror neurons" which fire both when executing a specific behavior and observing another individual making a similar action were first found in monkeys [18] and later in humans [51]. The existence of a mirror system corroborates the direct-matching hypothesis which states that an action is

understood when we map its observed visual representation onto our motor representation of the same action. However, the detailed role of mirror neurons as a part of the neurophysiological foundations of true imitation still deserves study.

Imitation learning provides means for intuitive robot programming. Robot applications invade our daily life. Consequently, more and more non-experts confront the problem of configuring these robots to meet their requirements. PbD replaces the explicit programming of a task by training a machine and interacting with it. This user-friendliness allows for more adaptable and sociable robots.

The drawbacks of learning in imitation are caused by the assumption that the demonstrated action is already a good solution. But what is good for the demonstrator may not necessarily be good for the imitator. Thus, misinterpreting the intentions of a presented skill may lead to a completely wrong-learned behavior. It is necessary to judge the capabilities of the demonstrator as well as the social context she performs her actions.

The key questions in learning by imitation are (see [44]):

Who to imitate? It is important to imitate an agent whose tasks and needs are relevant or beneficial. An evaluation of possible candidate models is needed if there are several demonstrators to choose from. Imitating somebody who is an expert at a specific task might be preferred to a rookie executing the same task. As mentioned above, the social context that accompanies the task demonstration is of importance.

When (and where) to imitate? The demonstrated behavior has to be segmented, i.e. beginning and end of an action have to be assigned properly. Apart from the observation, the imitator needs to decide when and where it is reasonable to reproduce the behavior considering the context.

What to imitate? A crucial question depends on the essential features of a task that should be imitated. An observed behavior can be imitated in terms of states, actions, or effects. These kinds of imitations range from copying the surface form of a behavior (trajectory or action level) to copying its organizational structure (symbolic or program level). Our approach learns a task by imitating it at the trajectory level.

How to imitate? The embodiment and affordances of the imitator, i.e. the robot manipulator, in relation to those of the human demonstrator bear a major problem in imitation. A dissimilar morphology of demonstrator and imitator, in our case human and robot, may lead to arbitrary large differences in kinematics and dynamics of motion during the execution of the same action. Consequently, simple copying of a motion may be rendered impossible

or causes different unwanted effects and events. This problem is named "correspondence problem" and is of central importance in the imitation learning process.

Optimal situtations to apply imitation and involved demonstrators (when and who) are less important issues and are explored only scarcely sofar. In this work, the social dimension of imitation is completely ignored. The main issues are what and how to imitate, which refer to the learning and encoding of a skill respectively.

1.2 Problem Definition and Goal of this Thesis

A robot observes a human instructor while carrying out a specific manipulative task. After multiple demonstrations of the same behavior the robot should be able to imitate it even in a slightly modified setup. Problems arise as uncertainty corrupts the observations and executed actions. Moreover, the system needs to imitate at a trajectory level and should extract invariant task features. But of what kind is this constructed task knowledge and how does the robot rely on it during reproduction?

Thus, the goal of this thesis involves finding a generic imitation framework that robustly controls a robot manipulator. Concurrently, we seek a solution which captures the essential features of an imitated task while still allowing for maximal generalization. As an additional challenge, we want to cope with constraints not observed during demonstration, such as avoiding obstacles.

1.3 Contribution

Our approach extends and modifies the recent work of Calinon and Billard [10] in various ways. First, it formalizes the problem of imitation by means of a Dynamic Bayesian Network (DBN). Furthermore, we can incorporate additional constraints that should be considered during reproduction time. In this way, it is easily possible to implement a generic collision avoidance regardless of which specific manipulative motion is imitated. The problem of getting stuck in local minima can be overcome by planning a global optimal path in probability space.

1.4 Outline

The remainder of this thesis is organized as follows: We review the current state of research in learning by imitation in Chapter 2. Next, the fundamental terminology of robot manipulators including forward and inverse kinematics is presented in Chapter 3. Chapter 4 deals with the first processing stage of our imitation framework, where human motion is captured and multiple observations of the same action are aligned. Our probabilistic approach to imitation is handled in Chapter 5

involving skill encoding and reproduction, i.e. the main questions of how and what to imitate. The efficacy of the approach is shown in Chapter 6 by a pick-and-place, a pouring, and a whiteboard cleaning task. Finally, Chapter 7 concludes the thesis by discussing the pros and cons of the imitation framework and showing useful extensions.

2 Related Work

In the past, various techniques have been proposed for transferring task knowledge to a robotic system. Initially, the required motion trajectories were hand-coded by an engineer [14, 55]. However, the more complex the task description becomes, the more difficult it is to create and maintain large controllers. Alternatively, the required joint angle trajectories of the robot can be shown by a single demonstration, for example, by a human teacher using a joystick, a motion capturing system, or kinesthetic training. The resulting sequence is recorded and can then be replayed by the robot. If the observations are noisy or unpredicted disturbances in the task environment occur, simple playback of the recorded motion is in general not sufficient to reliably reproduce a given task. To deal with noise in the observations and to generalize over multiple demonstrations of the same tasks, several authors suggested hidden Markov models (HMM) to encode and reproduce a demonstrated action.

In an early work, Tso *et al.* [61] selected the best trajectory from a set of multiple demonstrations of a single task applying an HMM. To achieve discrete observation symbols for the HMM, the input trajectories are transformed to their frequency spectrum function and turned into discretized codes by vector quantization. Subsequently, an HMM is trained on the basis of those pre-processed trajectories. Finally, the trajectory obtaining the maximum likelihood from the HMM model is considered the best trajectory to reproduce the task.

Asfour *et al.* [4] use HMMs to generalize multiple demonstrations of a dual-arm movement. The hand path, hand orientation path, and joint angle trajectories are recorded and pre-processed extracting characteristic features. Those features are defined separately for position, orientation and joint angles (e.g. detecting sharp corners in the end-effector trajectory) and result in a set of "key points". For each kind of observation a HMM is trained with the key points from all demonstrations. To reproduce the movement the HMM states related to common key points (key points that occur in the majority of demonstrations) are used.

Calinon *et al.* [11] also proposed a HMM-driven approach to learn keypoint sequences of multiple demonstrations. Additionally, they introduce a cost function over a set of variables/constraints (hand path, joint angles, hand-object distance, laterality) which measures their relative importance according to their variability during multiple demonstrations. The optimal imitation w.r.t. the cost function is then determined by gradient-descent. To convert Cartesian coordinates into configuration space a numerical solution to the inverse kinematics problem, the pseudo-inverse, is used.

Recently, Kulić *et al.* [35] developed a system for incremental longterm learning

that enables the recognition and regeneration of whole-body motion patterns. They use a hierarchy of factorial HMMs (FHMM; a generalization of the standard HMM, where a single output is generated by multiple interacting dynamic processes) to encode human motion. FHMMs overcome the trade-off associated with the selection of the number of model states. While a small number of states serves generalization across variable data and is better at recognizing new data, a large state model provides sufficient details required for reproducing observed motion. The system clusters recognized motion and organizes it into a hierarchical tree structure, where leaf nodes represent the most specific motion descriptors.

Reinforcement learning (RL) techniques have been successfully applied in order to learn controllers for an individual skill or for nonlinear dynamic motor primitives (see e.g., [26]). Focusing on humanoid motor control, Peters *et al.* [48] described an improvement to traditional greedy policy-improvement. They introduce an algorithm that benefits from exploiting a natural policy gradient instead of the vanilla policy gradient used in the past. Another complementary employment of RL techniques is presented by Bentivegna *et al.* [6]. Initially, motion primitives solving a marble maze task are learned from observing the teacher. Then, practice takes place, where the motion policy is further refined using Q-Learning while performing the task. However, for larger tasks self-exploring approaches may not be applicable due to the exploding size of the search space.

The question of how to imitate and especially the correspondence problem, i.e. mapping action sequences of the demonstrator to the imitator agent, is tackled by Alissandrakis *et al* [2] introducing the ALICE framework (Action Learning for Imitation via Correspondence between embodiments). ALICE is a generic mechanism that wraps around any particular imitation generating method. Imitative behavior produced by the generating method is stored together with the demonstrated action in a library of correspondences. This library can be used to avoid re-calculating computationally expensive mapping solutions. As a second component ALICE examines the history of the imitator's actions discovering further imitative sequences and adding them as alternative correspondence solutions to the library. Thus, the performance of the imitator is not limited by the choice of the generating method and it overcomes the problem of actions becoming invalid in certain contexts. ALICE's efficiency is demonstrated in the chessworld testbed, where different embodiments/chess pieces reveal different movement rules. When a knight is required to imitate the movements of a queen it performs superior (in terms of displacement distance) including ALICE instead of just applying the generating method.

Another tool that aims at describing body correspondence are interpersonal maps introduced by Hafner and Kaplan [27]. An information distance matrix is built by calculating the entropy of each sensory state (internal proprioceptive sensors and external sensors perceiving other robots) as well as the conditional entropy between any two sensors. The corresponding body map evolves from applying a relaxation algorithm that iteratively positions the sensors in two-dimensional space minimizing the projection error. AIBO robots equipped with 18 sensors are used to collect data while walking. In general, the body schemas of two robots appear

Figure 2.1: Related approaches to Learning by Imitation. **Left:** Calinon and Billard [9] transfer a skill from human to robot combining different modalities. First, the robot passively watches the human executing a task. Then, the teacher revises the robot's imitation attempt through kinesthetic teaching. **Right:** In Ijspeert *et al.* [28] a humanoid learns a forehand tennis swing from human demonstration using dynamical systems.

as two distinct clusters in the body map. However in case of direct imitation, some intercorrelations between the two sets of sensors can be found.

Apart from approaches which describe task knowledge in the form of motion trajectories, current research also covers skill representation on a symbolic level. Though this requires sets of predefined basic controllers it allows for complicated high-level skills and reasoning. Pardowitz and Dillmann [47] presented a system that is able to generalize over household tasks in a hierarchical manner. Actions performed by the human demonstrator are recognized as a sequence of "elementary operators", of which a graph-based task representation is learned. In this approach, the incrementally updated network topology reflects the learned temporal ordering of the individual actions. The efficacy of the method is shown by setting the table after multiple differing demonstrations. To discover corresponding actions between demonstrations, a similarity measurement based on object features (type, functional role) and pre- and post-condition features (including geometrical relations between objects) is implemented. Furthermore, the task similarity helps to cluster task demonstrations by selecting class prototypes.

Symbolic methods fail in domains where a continuous motor control is required. In contrast, learning trajectories rely on massive amounts of captured data. Hence, dimensionality reduction techniques are often applied. Al-Zubi and Sommer [1] present a learning model to mimic human arm movement based on intrinsic and extrinsic features. Intrinsic features describing the mechanics of movement are extracted by applying principal component analysis (PCA) on several samples of recorded joint angle sequences. They are combined with extrinsic features that capture the relation between points in eigenspace and constraint space, where absolute end-effector or obstacle positions are represented. To learn this association a dynamic cell structure (DCS) network is employed that is able to reflect non-linear mappings. As a result, human-like arm trajectories can be synthesized and

generalized to some extent (over start, end, and obstacle position).

Chalodhorn *et al.* [12] used PCA to reduce the high-dimensional motion capture data of a recorded human walk. While a direct playback of the human data on a humanoid robot would make the robot fall, the authors showed that after a few trials the robot was able modify the imitated gait incrementally. Here, a sensory-motor predictor was learned using a radial basis function approximator. An objective function defining torso stability on the basis of gyroscope measurements is optimized by means of the predictive model to find an optimal action sequence in latent subspace which in turn produces dynamically stable motions.

Similarly, Grimes *et al.* [23] also used PCA to reduce the high-dimensional configuration space. Sensory-motor experiences are utilized to learn a non-parametric forward model via Gaussian Process Regression. The imitation process is modelled adopting a dynamic Bayesian network (DBN) and supplementary constraint variables. To infer dynamically stable imitative whole-body actions they leverage existing inference techniques in graphical models such as belief propagation. The same imitation framework was extended by capturing human motion data from a monocular video sequence, detecting most probable body part locations by means of a multi-colored suit [13]. More recently, in [24] they replaced the Gaussian Process forward model by a Gaussian Mixture Model (GMM) allowing for efficient closed form evaluation of the integrals found during inference, without the need to resort to Monte Carlo approximations.

An approach to encode motion trajectories from multiple demonstrations on the base of a set of dynamical systems was presented by Ijspeert *et al.* [28]. These so-called control policies (CPs) do not index trajectories by time. Thus, they are robust to dynamical changes in the environment, The reproduction of a task benefits from the fact, that CPs encode a whole attractor landscape. The desired trajectory results from evolution from the initial state to a unique point attractor, the goal state. In this way, they can generalize reaching movements to a specific target or more complex ones, such as a tennis swing (see Figure 2.1). Apart from point attractors, they also apply limit cycle attractors to encode rhythmic movements, e.g., drumming.

Another important trend in learning by imitation develops biologically-oriented computational models. One of those approaches was presented by Billard and Matarić [8]. Their connectionist model imitates human arm movements. It is composed of a hierarchy of artificial neural networks, each resembling the functionality of a specific brain area involved in visio-motor control. Their experiments show that imitating reaching movements using a biomechanical simulation of a humanoid avatar agrees with real data produced by humans who were engaged in the same imitation task.

Ogawara *et al.* [46] suggested an approach that is able to imitate the invariant features of multiple demonstrations of a grasp-and-pouring task. So-called essential interactions between the grasped object and environmental objects are extracted using a multidimensional dynamic programming matching method. Their idea of essential interactions is similar to the constraints we use in our approach. In

contrast, they build a hybrid task model containing symbolic and trajectory level information reproduce a task. It describes the essential interaction elements as well as their geometrical relations by a mean and variance.

Our approach is related to the recent work of Calinon and Billard [10]. In their approach, observed motion trajectories are first analyzed an transformed into latent space via PCA. After temporally aligning these trajectories using dynamic time warping, they abstract the data into a set of Gaussian mixtures. The number of mixtures is selected via the Bayesian information criterion (BIC). To find the GMM parameters the expectation-maximization (EM) algorithm is used. Instead of generalizing using GMMs our approach relies on a nonparametric kernel regression. During reproduction, the desired position is generated by the GMM. The variances estimated in each individual constraint dimension can be seen as a measure of the relative importance of each particular constraint. Furthermore, they combined their approach with RL techniques to learn an obstacle avoidance [25]. In contrast, we implement collision avoidance by allowing additional constraints online during reproduction. Moreover, we formulize the learning problem as a Dynamic Bayesian Network, can find a global optimal imitative action sequence by planning, use a different inverse kinematic approach, and run the majority of our experiments on manipulators that do not share a human-like morphology.

3 Fundamentals

The following chapter describes different approaches to represent orientations. Building on that, we introduce an open-loop kinematic chain structure with a single end-effector. Finally, the most fundamental techniques when working with robot manipulators are explained in detail, namely forward and inverse kinematics.

3.1 Representing Orientations

In contrast to positions, handling rotations is not as trivial as it may seem, because of their inherent non-Euclidean nature. We define the orientation of a rigid body as its instantaneous rotational configuration, i.e., the rotation w.r.t. a fixed, global frame of reference.

Before presenting widely-used orientation parameterizations numerous conventions have to be stated. We will use right handed coordinate systems (with x-axis to the right, y-axis away from the viewer, and z-axis straight up), column vectors, and a counter-clockwise direction of positive angles (i.e., rotate the object not the reference frame).

The Euler rotation theorem states that an arbitrary rotation in three-dimensional space can be parameterized using three variables. This implies that there exists a sequence of not more than three rotations about distinctive coordinate axes, where no two successive rotations may be about the same axis, to align two independent coordinate frames. Representations that arise directly from this principle are simple and compact but suffer from singularities, are hard to compose, and do not interpolate well. More complex non-Euclidean parameterizations avoiding these problems may be less intuitive and lack existing numerical tools.

The following section compares orientation representations regarding their stability, computational complexity, and practicability. Furthermore, conversions between them are described. Our approach relies on the representation of orientations, as we have to describe joint configurations and object poses.

3.1.1 Euler Angles

Euler angles are a minimal representation to describe an orientation in three-dimensional space using three scalars (ϕ, θ, ψ). Each angle specifies a rotation around one axis of the body-fixed frame of reference.

When it comes to define Euler angles one can think of various conventions, depending on the axes around which the rotations are carried out and their order

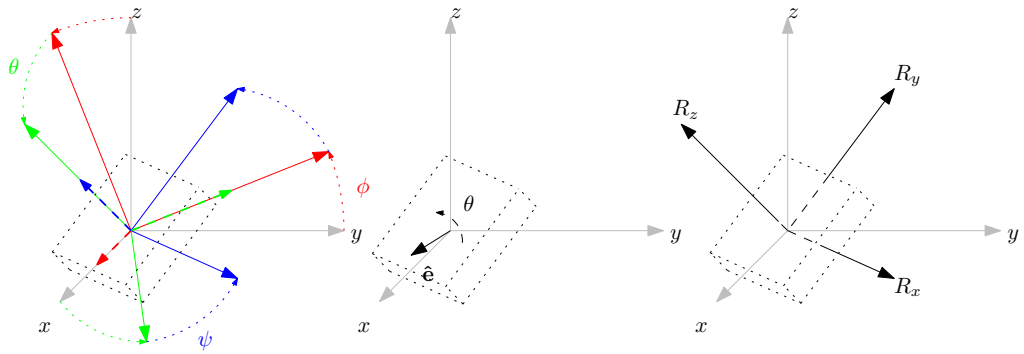

Figure 3.1: Visualization of the different representations of the same orientation of a rigid body. **Left**: Euler Angles, where the three consecutive rotations around the different axes are colored red, green and blue. Temporary resulting reference frames are also sketched. **Middle**: Angle-Axis, where the same orientation is realized via a rotation around $\hat{\boldsymbol{e}}$ of angle θ. **Right**: Rotation Matrix, where each column is illustrated as a vector.

of application. There are $3! = 6$ options to combine the x-, y-, and z-axis (called Cardanian type sequences). Additionally, the last rotated axis can coincide with the first one, as this does not violate the Euler theorem (called Eulerian type sequences). This sums up to a total of 12 possible Euler angle definitions. While all of them are equivalent, some have been more established than others, e.g., the x convention (z-x-z) or the yaw-pitch-roll convention (z-y-x) in the aeronautical domain. Throughout this thesis we will use the z-y-x convention.

Besides the fact that Euler angles are a simple and very compact representation they entail three major drawbacks. First, there exists no operation to compose two Euler angle rotations without changing to another representation. Second, due to the non-linear nature of rotations some orientations - depending on the convention - do not correspond to a unique Euler angle. Such a singularity is a continuous subspace of the parameter space, all of whose elements correspond to the same orientation. Movement within this subspace does not result in a change of rotation. In reverse, small increments of the orientation close to these critical points may cause discontinuous jumps in the Euler angles. During a singularity one degree of freedom is lost as two of the three axis are parallel. Another term referring to this phenomena is “gimbal lock”.

Finally, linear interpolation between Euler angles frequently results in a rather jerky behavior (cf. Figure 3.2).

3.1.2 Angle-Axis

Another representation for orientation that evolves directly from Euler’s rotation theorem is the parameterization via a unit vector $\hat{\boldsymbol{e}}$ and a scalar θ.

The unit vector $\hat{\boldsymbol{e}} = [e_x, e_y, e_z]^T$ indicates the rotation axis and direction exhibiting two degrees of freedom. The third degree of freedom is added by the angle θ,

which describes the magnitude of the rotation about the axis $\hat{\boldsymbol{e}}$.

A very similar but more concise way to express an orientation is the rotation vector $\boldsymbol{r} = \theta \cdot \hat{\boldsymbol{e}}$, where the angle is encoded in the length of a non-normalized vector $\boldsymbol{r}$. If the rotation angle θ is zero, the axis is not uniquely defined.

To rotate a given vector $\boldsymbol{v}$ using angle-axis one has to apply Rodrigues' rotation formula:

$$\boldsymbol{v''} = \boldsymbol{v} \cdot \cos\theta + \hat{\boldsymbol{e}} \times \boldsymbol{v} \cdot \sin\theta + \langle \hat{\boldsymbol{e}}, \boldsymbol{v} \rangle \hat{\boldsymbol{e}} \cdot (1 - \cos\theta).$$

As it is the case with Euler angles, the angle-axis representation exposes singularities. Moreover, it isn't suited to directly combine successive rotations. It is inevitable to switch to a different representation (e.g., rotation matrix), calculate the product, and then convert the result back to angle-axis.

3.1.3 Rotation Matrix

Using three unit vectors $\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$ and $\hat{\mathbf{w}}$ as the basis of a rotated reference frame, one can describe arbitrary orientations in three dimensions. Together they form what is usually called a rotation or direct cosine matrix:

$$R = (\hat{\mathbf{u}}\ \hat{\mathbf{v}}\ \hat{\mathbf{w}}) = \begin{pmatrix} \hat{u}_x & \hat{v}_x & \hat{w}_x \\ \hat{u}_y & \hat{v}_y & \hat{w}_y \\ \hat{u}_z & \hat{v}_z & \hat{w}_z \end{pmatrix}$$

Despite the nine elements a rotation matrix is composed of, it only exhibits three degrees of freedom. This is due to the fact that these matrices are orthonormal, meaning that each vector $\hat{u}$, $\hat{v}$ and $\hat{w}$ has unit length and is orthogonal w.r.t. to the two other ones. These constraints lead to a highly redundant represention.

The main advantage of rotation matrices lies in the ease of combining sequences of rotations by simple matrix multiplication ($R_{total} = R_2 \cdot R_1$) as well as rotating a vector in space ($v' = R \cdot \boldsymbol{v}$). Thus, converting Euler angles to a rotation matrix just corresponds to the product of three elementary rotation matrices according to the chosen convention. For the z-y-x convention and Euler angles (ϕ, θ, ψ), this leads to:

$$\begin{aligned} & R_x(\phi) \cdot R_y(\theta) \cdot R_z(\psi) \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos\theta\cos\psi & -\cos\theta\sin\psi & \sin\theta \\ \sin\phi\sin\theta\cos\psi + \cos\phi\sin\psi & -\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & -\sin\phi\cos\theta \\ -\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi + \sin\phi\cos\psi & \cos\phi\cos\theta \end{pmatrix} \end{aligned}$$

The resulting rotation matrix can be converted back to Euler angles. To achieve this, element $\hat{w}_x$ is used to recover θ:

$$\theta_1 = \arcsin\left(\hat{w_x}\right) = \arcsin\left(\sin\theta\right)$$
$$\theta_2 = \pi - \theta_1.$$

Note, that there are two possible solutions since arcsin only returns values $\in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Angle ϕ is resolved by using elements $\hat{w}_y$ and $\hat{w}_z$:

$$\phi_1 = \arctan\left(\operatorname{sgn}(\cos\theta_1)\frac{-\hat{w_y}}{\hat{w_z}}\right) = \arctan\left(\operatorname{sgn}(\cos\theta_1)\frac{\sin\phi}{\cos\phi}\right)$$
$$\phi_2 = \arctan\left(\operatorname{sgn}(\cos\theta_2)\frac{-\hat{w_y}}{\hat{w_z}}\right) = \arctan\left(\operatorname{sgn}(\cos\theta_2)\frac{\sin\phi}{\cos\phi}\right).$$

Similarly, ψ is calculated:

$$\psi_1 = \arctan\left(\operatorname{sgn}(\cos\theta_1)\frac{-\hat{v_x}}{\hat{u_x}}\right) = \arctan\left(\operatorname{sgn}(\cos\theta_1)\frac{\sin\psi}{\cos\psi}\right)$$
$$\psi_2 = \arctan\left(\operatorname{sgn}(\cos\theta_2)\frac{-\hat{v_x}}{\hat{u_x}}\right) = \arctan\left(\operatorname{sgn}(\cos\theta_1)\frac{\sin\psi}{\cos\psi}\right).$$

This results in two Euler angle triplets $(\phi_1, \theta_1, \psi_1)$ and $(\phi_2, \theta_2, \psi_2)$, both referring to the same rotation matrix $R\left(\phi, \theta, \psi\right)$.

As already mentioned, the Euler angle representation suffers from singularities. Following the z-y-x convention they appear when $\theta = \pm\frac{\pi}{2}$ holds. The corresponding rotation matrix

$$R\left(\phi, \pm\frac{\pi}{2}, \psi\right) = \begin{pmatrix} 0 & 0 & \pm 1 \\ \pm\sin\phi\cos\psi + \cos\phi\sin\psi & \mp\sin\phi\sin\psi + \cos\phi\cos\psi & 0 \\ \mp\cos\phi\cos\psi + \sin\phi\sin\psi & \pm\cos\phi\sin\psi + \sin\phi\cos\psi & 0 \end{pmatrix}$$

reveals that the problem of finding appropriate Euler angles is underconstrained. Now, angles ϕ and ψ have to be calculated from non-zero matrix elements $\hat{u}_y$ and $\hat{v}_y$ using trigonometric identities which in turn causes an infinite number of possible solutions:

$$\psi = \begin{cases} \arctan\left(\frac{\hat{u}_y}{\hat{v}_y}\right) - \phi = \arctan\left(\frac{\sin(\phi+\psi)}{\cos(\phi+\psi)}\right) - \phi & \text{, if } \theta = \frac{\pi}{2} \\ \arctan\left(\frac{\hat{u}_y}{\hat{v}_y}\right) + \phi = \arctan\left(\frac{\sin(\psi-\phi)}{\cos(\psi-\phi)}\right) + \phi & \text{, if } \theta = -\frac{\pi}{2}. \end{cases}$$

An often chosen solution is to set $\phi = 0$ and $\psi = \arctan\left(\frac{\hat{u}_y}{\hat{v}_y}\right)$.

Interpolation between two rotation matrices can be achieved by interpolating two column vectors of each matrix separately and using the resulting vectors to create an orthonormal matrix. But this method may produce little intuitive rotations. A preferred way is to switch to quaternions and benefit from interpolation techniques applicable with this type of representation.

3.1.4 Quaternions

Quaternions are the mathematical extension of the complex numbers. They describe the relationship between two vectors, i.e., they can be defined as their geometrical quotient. Thus, they form an useful tool for representing and calculating spatial orientations. A quaternion $\boldsymbol{q}$ can be written as

$$\boldsymbol{q} = w + xi + yj + zk = [w, x, y, z]^T,$$

where $\{w, x, y, z\} \in \mathbb{R}$ and $\{i, j, k\}$ are imaginary components satisfying

$$i^2 = j^2 = k^2 = ijk = -1$$

and the common algebraic rules except the commutative law of multiplication. It can be easily shown that the product of two quaternions $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ is defined as

$$\begin{aligned}\boldsymbol{q_1} \otimes \boldsymbol{q_2} = &(w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) + (w_1x_2 + x_1w_2 + y_1z_2 + z_1y_2)i \\ &+ (w_1y_2 + x_1z_2 + y_1w_2 + z_1x_2)j + (w_1z_2 + x_1y_2 + y_1x_2 + z_1w_2)k.\end{aligned}$$

In analogy to the angle-axis representation, the complex part of a quaternion behaves like a rotation axis, whereas the real part corresponds to the angle to rotate around it:

$$\boldsymbol{q} = \cos\left(\frac{\theta}{2}\right) + \hat{\boldsymbol{e}}\sin\left(\frac{\theta}{2}\right),$$

where $\hat{\boldsymbol{e}}$ is a unit vector. When used in the rotation domain quaternions are required to have unit length.

The quaternion product $\boldsymbol{q} \otimes \left[0\ \boldsymbol{v}^T\right]^T \otimes \boldsymbol{q^{-1}}$ yields the vector $\boldsymbol{v}$ rotated by the angle θ around the axis $\hat{\boldsymbol{e}}$. To combine two consecutive rotations $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ one simply multiplies them: $\boldsymbol{q_2} \otimes \boldsymbol{q_1} \otimes \left[0\ \boldsymbol{v}^T\right]^T \otimes \boldsymbol{q_1^{-1}} \otimes \boldsymbol{q_2^{-1}}$.

Using quaternions we need a way to convert bidirectionally to other representations. In the *z-y-x* convention, given Euler angles (ϕ, θ, ψ) the corresponding quaternion is defined as

$$\begin{aligned}\boldsymbol{q_{xyz}}(\phi, \theta, \psi) = \boldsymbol{q_z}(\phi) \otimes \boldsymbol{q_y}(\theta) \otimes \boldsymbol{q_x}(\psi) &= \begin{pmatrix}\cos\left(\frac{\phi}{2}\right) \\ \sin(\frac{\phi}{2}) \\ 0 \\ 0\end{pmatrix} \otimes \begin{pmatrix}\cos(\frac{\theta}{2}) \\ 0 \\ \sin(\frac{\theta}{2}) \\ 0\end{pmatrix} \otimes \begin{pmatrix}\cos(\frac{\psi}{2}) \\ 0 \\ 0 \\ \sin(\frac{\psi}{2})\end{pmatrix} \\ &= \begin{pmatrix}\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\phi}{2}\right) - \sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ \cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) + \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right) - \cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) + \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\phi}{2}\right)\end{pmatrix}.\end{aligned} \tag{3.1}$$

First, the rotation matrix Q equivalent to the quaternion $\boldsymbol{q}$ is derived by regrouping the elements of the product $\boldsymbol{qvq^{-1}} = Q\boldsymbol{v}$. This leads to

$$Q = \begin{pmatrix} w^2 + x^2 - y^2 - z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & w^2 - x^2 + y^2 - z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & w^2 - x^2 - y^2 + z^2 \end{pmatrix}$$

$$\overset{\|\boldsymbol{q}\|=1}{=} \begin{pmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{pmatrix}.$$

Now, the Euler angles (ϕ, θ, ψ) can be retrieved from Q like from any other rotation matrix, i.e., using the very same matrix elements:

$$\phi = \arctan\left(\frac{2yz - 2wx}{1 - 2x^2 - 2y^2}\right)$$

$$\theta = \arcsin\left(2xz + 2wy\right)$$

$$\psi = \arctan\left(\frac{2xy - 2wz}{1 - 2y^2 - 2z^2}\right).$$

The backward conversion to Euler angles has to deal with singularities, which occur when $\theta = \pm\frac{\pi}{2}$ applies. In this case, from 3.1 follows

$$\boldsymbol{q}\left(\phi, \pm\frac{\pi}{2}, \psi\right) = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \alpha \begin{pmatrix} \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\phi}{2}\right) \mp \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) \pm \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \pm\cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\psi}{2}\right) - \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\psi}{2}\right) \pm \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\phi}{2}\right) \end{pmatrix},$$

where $\alpha = \cos\left(\frac{\pi}{4}\right) = \sin\left(\frac{\pi}{4}\right) = \cos\left(-\frac{\pi}{4}\right) = -\sin\left(-\frac{\pi}{4}\right)$. Using trigonometric identities one obtains

$$w = \pm y = \alpha\cos\left(\frac{\phi}{2} \pm \frac{\psi}{2}\right) \qquad \text{and}$$

$$\pm z = x = \alpha\sin\left(\frac{\phi}{2} \pm \frac{\psi}{2}\right).$$

This leads directly to the missing Euler angles:

$$\phi \pm \psi = \pm 2\arctan\left(\frac{z}{w}\right).$$

The solution is disambiguated by setting $\psi = 0$ and $\phi = \pm 2\arctan\left(\frac{z}{w}\right)$.

Apart from never becoming singular, quaternions also feature a way to interpolate between orientations. A common technique evolved from the field of computer graphics is to interpolate in quaternion space via the spherical linear interpolation (*slerp*, popularized by Shoemake [57]). Given two unit quaternions $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ representing orientations and an interpolation tendency $t \in [0..1]$ the corresponding interpolated orientation $\boldsymbol{q_t}$ is obtained by:

$$\begin{aligned}\boldsymbol{q_t} = slerp\left(\boldsymbol{q_1}, \boldsymbol{q_2}, t\right) &= \boldsymbol{q_1}\left(\boldsymbol{q_2}\boldsymbol{q_1^{-1}}\right)^t \\ &= \boldsymbol{q_1}\frac{\sin\left(\left(1-t\right)\theta\right)}{\sin\theta} + \boldsymbol{q_2}\frac{\sin\left(t\,\theta\right)}{\sin\theta},\end{aligned}$$

where θ is half the angle between $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$.

There are three basic properties one usually claims when blending between orientations. First, the interpolated path of orientations should be the shortest, also referred to as the torque-minimal path. Second, the interpolation should occur at constant angular velocity. And finally, the interpolation should be commutative, i.e., blending between more than two orientations should be independent of their order. The *slerp* complies with all this, except for the commutativity.

Alternative interpolation techniques are always lacking exactly one of those properties. While *nlerp* (a simple and fast elementwise linear interpolation of quaternions followed by normalization) does not produce constant velocities, log-quaternion *lerp* (a linear interpolation using exponential maps, see [22]) does not result in a torque-minimal path.

For small interpolation distances, which are encountered in the majority of cases, all methods output a nearly identical path. Figure 3.2 shows an exemplary comparison between some of them. We will use the interpolation of rotations later on (see Section 4.1), when merging different observed orientations into a single estimate.

3.2 Homogeneous Transformations

We describe the spatial relationship between two arbitrarily located rigid bodies in three-dimensional Euclidean space by the relative translational and rotational displacement of their body-fixed reference frames w.r.t. a global static coordinate system. Accordingly, the need arises to compactly represent both rotation and translation in a unified manner.

Any linear transformation $t : \mathbb{R}^n \rightarrow \mathbb{R}^m$ can be represented by a matrix $T^{m\times n}$, such that $\forall x \in \mathbb{R}^n : t(x) = Tx$. Though translation is an affine transformation, in contrast to rotation it is not a linear one.

To take advantage of the matrix representation one needs to use homogeneous coordinates. This means adding an extra dimension, the homogeneous component, to our three dimensional vector $[x, y, z, 1]^T$. Thus, every affine transformation, including translation, can be expressed using a matrix.

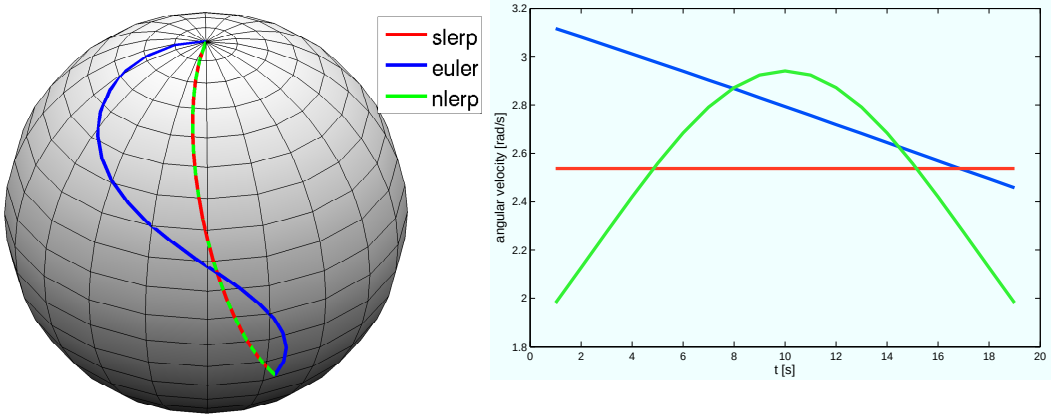

Figure 3.2: A qualitative comparison between different interpolation methods. Only two orientational key frames were given, eighteen intermediate ones were generated using the three different algorithms. The resulting paths show that linear interpolation of Euler angles is little intuitive while interpolation of quaternions using *nlerp* and *slerp* ends up in the shortest possible path on the unit sphere. In addition, only *slerp* preserves the constant speed property, as can be seen in the angular velocity plot.

The homogeneous matrix

$$T^{4\times4} = \begin{pmatrix} & R^{3\times3} & & \boldsymbol{t}^{3\times1} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

combines a rotation R with a following translation $\boldsymbol{t}$.

Homogeneous matrices can be composed sequentially by chain multiplication. A change of bases is accomplished easily. Given a point $\boldsymbol{p}$ and a coordinate system T (whose axes are the column vectors of R and its origin is $\boldsymbol{t}$) w.r.t. the same global reference frame, the point $\boldsymbol{p'} = T\boldsymbol{p}$ matches $\boldsymbol{p}$ w.r.t. T.

To transform $\boldsymbol{p'}$ back to its original frame of reference, i.e., to calculate $\boldsymbol{p} = T^{-1}\boldsymbol{p'}$, we need the inverse of the homogeneous transformation matrix T, which is given by

$$T^{-1} = \begin{pmatrix} & R^{\boldsymbol{T}} & & -R^{\boldsymbol{T}}\boldsymbol{t} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

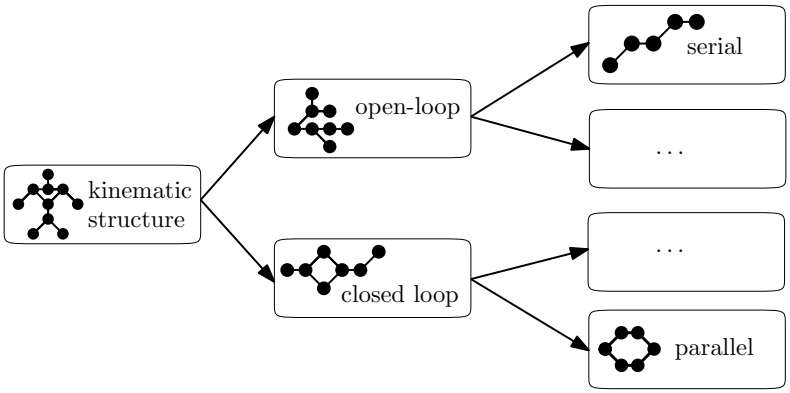

Figure 3.3: A taxonomy of manipulators. By means of graph theory (edges are joints, nodes are links) it is possible to divide manipulators into different types of classes. In this thesis, we focus on serial kinematic chains, listed in the upper right.

where R^T is the transposed rotation matrix which equals the inverse, since R is orthogonal.

Alternatively, dual quaternions are used to represent rigid motion (rotation plus translation) via one single operation. Dual quaternions, i.e., quaternions where each element has a supplemental counterpart, are less common. Thus, we stay with homogeneous matrices.

3.3 Manipulator Structures

A robot manipulator consists of a set of links connected by joints. From a topological point of view, a manipulator can be described as a simple graph (an undirected graph containing no loops or multiple edges). Each vertex represents a link, whereas edges denote the joints between them.

A crucial distinctive feature of kinematic structures is the existence of cycles in their graph models. If the graph is acyclic the manipulator is termed open, otherwise closed. Serial and parallel manipulators are further specializations of open and closed mechanisms, identified by path graphs (trees with two nodes of vertex degree one and the remaining nodes with a vertex degree of two) and Hamiltonian circuits (a graph cycle where each node is visited exactly once) respectively. The notion kinematic chain refers to the serial architecture and is the elementary structure of a robot manipulator. An hierarchical overview of different manipulator structures is given in Figure 3.3.

The open kinematic chain contains only one single sequence of links between its two ends. In tree-like graph structures the root node is named base, whereas a leave is referred to as an end-effector.

The agility of a manipulator is established by means of its joints. Mechanical joints, or generally speaking lower kinematic pairs (kinematic constraints between

rigid bodies with surface-contact, point- or curve-contacts result in higher kinematic pairs), are classified into various types: revolute ("hinge"), prismatic ("slider"), spherical ("ball and socket"), helical, and planar. Joints that depend on an input are called actuators, while others are passive joints. The prismatic and revolute joints are the most simple joints to engineer and thus are used by nearly all manipulators in practice. They perform a relative, one dimensional translation or rotation w.r.t. its adjacent links. All of the other joints mentioned above can be synthesized by means of combinations of revolute and prismatic joints. Hereafter, these two are the only types considered. Optionally, limits can be imposed on joints reducing their set of valid configurations.

An important attribute of a manipulator characterizing its level of mobility is the number of degrees of freedom (DoF). It is derived by the sum of the degrees of freedom of each individual joint. Some literature additionally introduces the notion of degrees of mobility (e.g., [55]), which replaces the DoF term. Hence, the DoF of a manipulator solely depends on the DoF of the end-effector frame, bounded by the upper limit of six possible DoFs in three-dimensional space (three translational plus three rotational DoFs). Following this notation, a manipulator is said to be kinematically redundant, if more degrees of mobility than degrees of freedom are available.

3.3.1 Human-Arm-Like Kinematic Chain

We need a human arm model to capture the joint configurations of the human instructor during the demonstration of a manipulative task. Serial chains that resemble the human arm structure are called Human-Arm-Like (HAL) chains. HAL chains usually assume that there exist centers of rotation and reference locations between segments and joints in the human arm. These simplifications are justified and only introduce insignificant errors as can be seen with recently developed arm models [21]. We further simplify our HAL chain to 6 DoFs to comply with our imitation agent, the 6-DoF robot manipulator Zora (see next section). Figure 3.4 shows a schematic version of the HAL chain used. It consists of a ball and socket joint in the shoulder (3 DoFs) connected with the same type of joint in the elbow (3 DoFs) which is finally linked to the end-effector.

Another decision to be made, refers to the joint coordinate system, especially the order of applied Euler angles. The International Society of Biomechanics recommends certain definitions (cf. [65]), trying to ease the communication of kinematic data among researchers. To avoid gimbal lock some convention orders are preferred, which place the singularities out of the motion range that is affected during demonstration. However, there is no single favored convention performing well with all different types of motion like flexion-extension or abduction-adduction. We choose the cardan sequence z-y-x for both, shoulder and elbow joint.

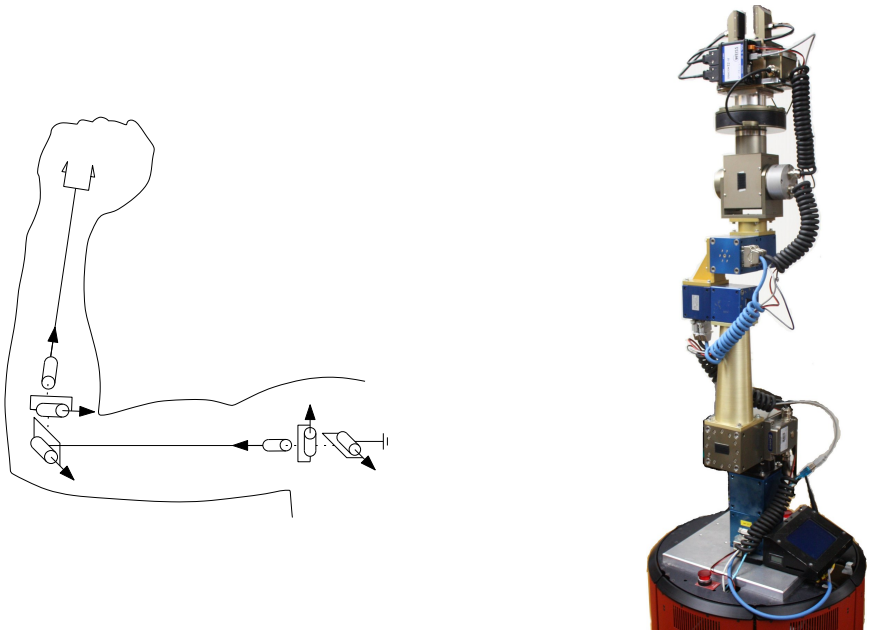

Figure 3.4: Morphology of our human demonstrator and robotic imitator. **Left:** A simplified 6-DoF anthropomorphic mechanical arm structure. We use this model to capture joint configurations of the human instructor during demonstration. Two 3-DoF socket and ball joints form the shoulder and elbow joints. The order of rotation axes is explicitly illustrated (dotted lines imply no physical distance). **Right:** Zora, a 6-DoF robotic manipulator mounted on top of a 3-DoF mobile platform. This serial manipulator is used to imitate human manipulative tasks.

3.3.2 Robotic Manipulator Zora

Zora (see Figure 3.4) is the name of the robot used throughout the experiments to imitate human manipulative actions. It possesses a 6-DoF serial kinematic chain which consists of six revolute joints rotating around the z-, x-, y-, x-, y-, and z-axis from base to end-effector. Due to the independent functional capability of the individual components the kinematic structure is easily reconfigurable.

The manipulator is mounted on top of a mobile platform exhibiting three additional degrees of freedom. However, these are not used in this thesis, leaving the robot arm fixed during activity. A two-finger gripper serves as end-effector. Furthermore, Zora is equipped with a tactile sensor, a webcam and a sensor measuring force and torque. For technical details of Zora and its joint units see B.2 and B.1 respectively.

3.4 Configuration Space and Workspace

The joint configuration $\boldsymbol{q} = [q_1, \ldots, q_n]^T$ with $q_i \in [q_{i_{min}}, q_{i_{max}}]$ completely determines the pose of a manipulator. The configuration of one joint is a continuous function of one or more real scalars. In case of rotational joints, the scalar is the angle of the joint. All possible configurations $\boldsymbol{q}$ span up the configuration space $\mathbb{C}$, also known as C-space. The dimension of $\mathbb{C}$ is n which coincides with the number of joints and the DoFs of the manipulator structure. This leads to an hypersquare in $\mathbb{R}^n$, as all joint variables are bounded:

$$\mathbb{C} = [q_{1_{min}}, q_{1_{max}}] \times \ldots \times [q_{n_{min}}, q_{n_{max}}] .$$

The region of its environment a manipulator's end-effector can access is called workspace, irrespective of how the manipulator must move to reach those positions. It can be defined as

$$\mathbb{W} = \bigcup_{\boldsymbol{q} \in \mathbb{C}} \{ f(\boldsymbol{q}) \},$$

where $f(\boldsymbol{q})$ is the forward kinematic function (as defined in 3.5), which returns the end-effector frame to a given joint configuration. Sometimes $\mathbb{W}$ is also referred to as the reachable workspace. It is constrained by the geometric structure of the manipulator as well as the presence of mechanical joint limits.

The dexterous workspace $\mathbb{W}_{dex}$ is a subset of $\mathbb{W}$ and concerns only workspace volume that is accessible with all orientations:

$$\mathbb{W}_{dex} = \{ \boldsymbol{e} \in \mathbb{W} \mid \forall r \in SO(3) \, \exists \boldsymbol{q} \in \mathbb{C} : f(\boldsymbol{q}) = [\boldsymbol{e}, r]^T \},$$

with $SO(3)$ being the set of all rotations in three-dimensional space (also referred to as the special orthogonal group). At each point in $\mathbb{W}_{dex}$ the end-effector can be arbitrarily oriented. Therefore, the manipulator has complete manipulative capability in the dexterous workspace, while it is limited in the reachable workspace.

The configuration space and workspace of a simple 2-DoF manipulator and a circular obstacle are illustrated in Figure 3.5.

Task space refers to the set of poses needed to solve a given manipulative task. Often task space is a subspace of the work space and ignores dimensions like orientation.

3.5 Forward Kinematics

Forward or direct kinematics is the problem of finding the end-effector position and orientation $\boldsymbol{e}$ of a manipulator as a function f of its n-dimensional joint variable $\boldsymbol{q}$, i.e., $f : \mathbb{R}^n \to \mathbb{R}^3 \times SO(3)$.

3.5.1 General Approach

Let an open kinematic chain be constituted by $n+1$ links and n joints. Link $i \in \{0, \ldots, n-1\}$ is connected to link $i+1$ by joint i. While link 0 corresponds to the base of the kinematic chain, link n matches the end-effector.

Each link has a reference frame attached to itself describing its position and rotation w.r.t a global static coordinate system via a homogeneous matrix (3.2). The transformation between two neighboring link frames i and $i+1$ is then given by another homogeneous matrix $T_{i+1}^{i}(q_i)$, which depends exclusively on joint i. To calculate the forward kinematics, one simply applies the composition formula of homogeneous matrices from the base to the end-effector frame:

$$f(\boldsymbol{q}) = T_e^b(\boldsymbol{q}) = T_0^b \left(\prod_{i=0}^{n} T_{i+1}^{i}(q_i) \right) T_e^n, \tag{3.2}$$

where T_0^b and T_e^n are constant transformations, explaining optional offsets between the base and link 0 and the end-effector and link n respectively.

This approach works for any serial manipulator, with any number of joints. If at least one of the joints is a revolute one the resulting mapping $f(\boldsymbol{q}) = \boldsymbol{e}$ is nonlinear in $\boldsymbol{q}$.

3.5.2 Denavit-Hartenberg

In robotics applications the Denavit-Hartenberg (D-H) convention [16] is a widely used notation. It specifies the transformation of coordinate systems in linkages of kinematic chains in an efficient way. Instead of the six variables needed to form an arbitrary homogeneous transformation, the minimal representation of the D-H notation uses only four parameters: two angles α and θ and two distances d and a.

Figure 3.6 is a schematic illustration of these four D-H parameters. We denote the rotation axis of the revolute joint i as z_i. The parameter a_i describes the shortest distance between two consecutive joint axes z_{i-1} and z_i, i.e., the common

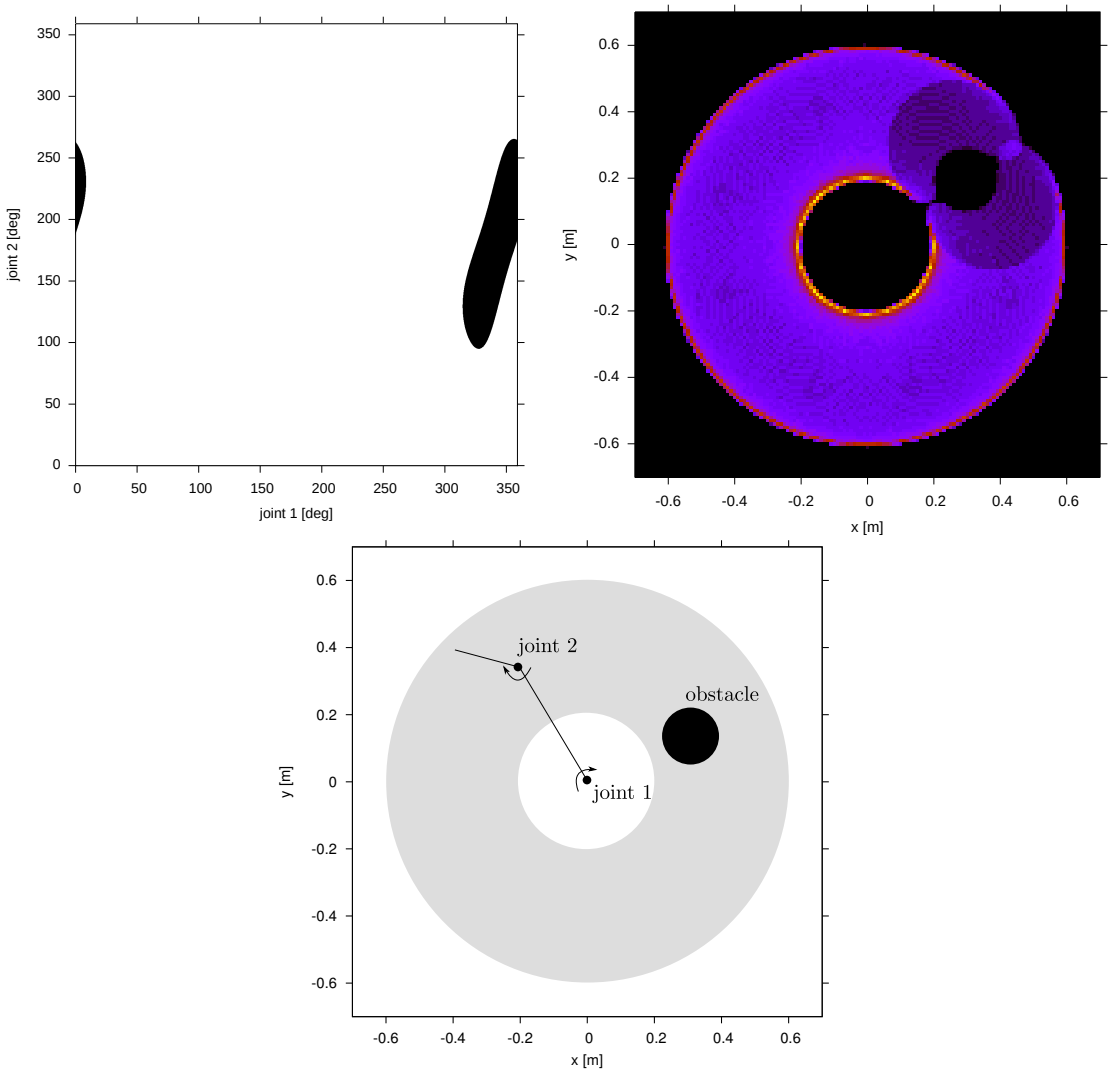

Figure 3.5: Configuration space and workspace. **Bottom**: The dimensions of a 2-DoF chain and the position of the obstacle are shown. **Left:** Configuration space of the 2-DoF manipulator. White area denotes valid joint configurations, whereas the black region represents the circular obstacle not accessible by the robot arm. **Right:** Corresponding workspace. The color identifies the percentage of accessible orientations in every point, ranging from black (not reachable) to yellow (part of dexterous workspace).

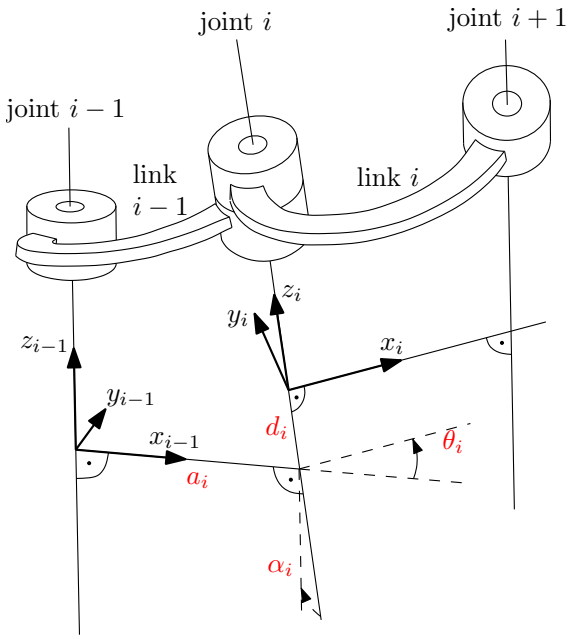

Figure 3.6: The four Denavit-Hartenberg parameters a, α, d, and θ. Using these parameters, a minimal representation for describing coordinate system transformations can be established. This in turn forms the forward kinematics. See text for details.

perpendicular. The included angle of axes z_{i-1} and z_i is represented by α_{i-1}. The distance d_i is measured between the origin of the local coordinate frame x_i-y_i-z_i and the intersection of the common perpendicular x_i with axis z_i. Parameter θ_i is the included angle of the two common perpendiculars x_{i-1} and x_i. The lack of two degrees of freedom in contrast to arbitrary homogeneous transformations imposes two constraints on the representable frame transformations. First, the axis x_i must be perpendicular to z_{i-1}. Second, x_i has to intersect the axis z_{i-1}.

Once the D-H parameters are identified we can use them to calculate the homogeneous transformation $T_{i+1}^{i}(q_i)$ between two neighboring link frames i and $i+1$

by simply combining four basic transformations:

$$
\begin{aligned}
T_{i+1}^{i}\left(q_i\right) =& Rot\left(\theta_i + q_i\right) Trans\left(d_i\right) Trans\left(a_i\right) Rot\left(\alpha_i\right) \\
=& \begin{pmatrix} \cos\left(\theta_i + q_i\right) & -\sin\left(\theta_i + q_i\right) & 0 & 0 \\ \sin\left(\theta_i + q_i\right) & \cos\left(\theta_i + q_i\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
& \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
=& \begin{pmatrix} \cos\left(\theta_i + q_i\right) & -\sin\left(\theta_i + q_i\right)\cos\alpha_i & \sin\left(\theta_i + q_i\right)\sin\alpha_i & a_i\cos\left(\theta_i + q_i\right) \\ \sin\left(\theta_i + q_i\right) & \cos\left(\theta_i + q_i\right)\cos\alpha_i & -\cos\left(\theta_i + q_i\right)\sin\alpha_i & a_i\sin\left(\theta_i + q_i\right) \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}.
\end{aligned}
$$

Note, that the joint variable q_i corresponds to the D-H parameter θ_i as all joints we use are revolute ones. In case of prismatic joints, the variable would correspond to the parameter d_i. Finally, the forward kinematic function f can be calculated by concatenating the local frame transformations, as done in Eq. 3.2.

There exist other minimal representations besides the Denavit-Hartenberg notation, such as the Hayati-Roberts convention which only relies on four parameters as well. The benefit of using D-H for forward kinematics in place of the general approach vanishes if the constraints mentioned earlier cannot be satisfied. In that case (which also applies for the kinematic structure of our robot Zora), one has to insert extra virtual joints. We avoid this unnecessary overhead and stay with the general approach.

3.6 Inverse Kinematics

Inverse kinematics (IK) is the problem of finding all valid joint configurations $\boldsymbol{q}$ given a manipulator structure and an end-effector position $\boldsymbol{e}$. Let f represent the forward kinematics map (cf. Section 3.5). Then, we have to resolve the nonlinear mapping

$$\boldsymbol{q} = f^{-1}\left(\boldsymbol{e}\right).$$

Depending on the task to execute, the dimension of $\boldsymbol{e}$ varies. There are three dimensions in a position task, whereas a position and orientation task implies six dimensions. Given $\boldsymbol{e} \in \mathbb{R}^3 \times SO\left(3\right)$ the inverse kinematics problem is only well posed if the manipulator's DoFs equal six, i.e., $dim\left(\boldsymbol{q}\right) = 6$.

If $dim\left(\boldsymbol{q}\right) < 6$ the problem is overconstrained and in general no solution exists. Contrary, if $dim\left(\boldsymbol{q}\right) > 6$ the problem is underconstrained and more than one (or

even infinitely many) solutions are possible. The existence of a solution is guaranteed if $\boldsymbol{e}$ belongs to the manipulator dexterous workspace.

Inverse kinematics is a much more complex problem to solve than forward kinematics. There exists a large body of literature originating not only from robotics but also from computer graphics and real-time animation. IK algorithms can be categorized into analytical and numerical ones.

Analytical methods include closed-form and algebraic-elimination-based techniques. In closed-form methods the solution can be deduced directly from a set of closed-form equations. As stated above, this is only possible if the IK problem is well posed, i.e. $dim\,(\text{task space}) = dim\,(\boldsymbol{q})$. Methods based on algebraic elimination express one joint variable as solutions to a high-degree polynomial, while the remaining joint variables are derived in closed-form. Though numerical routines are required to solve all the roots of the polynomial equations, algebraic elimination methods are still classified as being analytical.

Numerical IK approaches iteratively converge to a single solution, depending on the initial guess. In [60] the numerous numerical methods are grouped into three different classes:

- Applications of the Newton-Raphson algorithm which solves systems of nonlinear equations; in this case:

$$f\,(\boldsymbol{q}) - \boldsymbol{e} = 0.$$

- Converting the IK problem into a differential equations by means of joint and end-effector velocities.

- Considering it as a nonlinear optimization problem, using quasi-Newton or conjugate gradient methods.

Note, that this classification is somehow inconsistent, since the Newton-Raphson method is a special case of the resolved motion rate control with a constant time step and a first-order integration scheme. Furthermore, these two classes can easily be viewed as optimization procedures, iteratively minimizing the error $\Delta\boldsymbol{e}$ between actual and targeted end-effector pose.

In general, analytical IK methods should be preferred over numerical approaches, as they are computationally faster, more reliable and always find all admissible solutions. As a major drawback, a general analytical solution exists only for special non-redundant manipulator structures.

Computation of the Jacobian

In the following subsections three numerical methods, which are applicable to arbitrary kinematic structures, are presented in detail. Only the pure IK problem is considered without joint limits or self-collisions.

All these method share the fact, that they linearize the forward kinematics function f using the Jacobian matrix. The Jacobian matrix J is the matrix of all

first-order partial derivatives of the vector-valued function f, i.e. it is a function of the joint variables:

$$J(\boldsymbol{q}) = \begin{pmatrix} \frac{\partial f_1}{\partial \boldsymbol{q}_1} & \cdots & \frac{\partial f_1}{\partial \boldsymbol{q}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \boldsymbol{q}_1} & \cdots & \frac{\partial f_n}{\partial \boldsymbol{q}_n} \end{pmatrix},$$

where n is the number of DoFs of the kinematic structure and m the dimension of the task space. Intuitively, the ith column vector of $J(\boldsymbol{q})$ represents the relative change of the end-effector in workspace caused by a positive change of the joint variable $\boldsymbol{q}_i$ (associated with the ith joint).

To calculate the entries of the Jacobian one distinguishes between revolute and prismatic joints. If joint i is a revolute joint the ith column of $J(\boldsymbol{q})$ is given by the vector

$$\begin{pmatrix} \frac{\partial f_1}{\partial \boldsymbol{q}_i} \\ \vdots \\ \frac{\partial f_m}{\partial \boldsymbol{q}_i} \end{pmatrix} = \hat{\boldsymbol{v}}_i \times (\boldsymbol{e} - \boldsymbol{p}_i),$$

where $\hat{\boldsymbol{v}}_i$ is a unit vector pointing along the rotation axis of joint i, $\boldsymbol{p}_i$ its position, and $\boldsymbol{e} = f(\boldsymbol{q})$ the current end-effector pose. In case of a prismatic joint, this equation turns into the even simpler

$$\begin{pmatrix} \frac{\partial f_1}{\partial \boldsymbol{q}_i} \\ \vdots \\ \frac{\partial f_m}{\partial \boldsymbol{q}_i} \end{pmatrix} = \hat{\boldsymbol{v}}_i,$$

where $\hat{\boldsymbol{v}}_i$ is a unit vector representing the direction of translation. Obviously, the influence of a prismatic joint is independent of its distance to the end-effector.

Once the Jacobian matrix is computed, we can relate end-effector velocity and joint velocities by the forward dynamics

$$\dot{\boldsymbol{e}} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}}.$$

Integrating over time leads to the approximation

$$\Delta \boldsymbol{e} \approx J(\boldsymbol{q})\,\Delta \boldsymbol{q}. \tag{3.3}$$

For sufficiently small changes of joint angles $\Delta \boldsymbol{q}$ this linearization works. Thus, an iterative scheme can be applied, where the joint angles are successively updated ($\boldsymbol{q}_t = \boldsymbol{q}_{t-1} + \Delta \boldsymbol{q}$) and $J(\boldsymbol{q}_t)$ is recalculated every iteration cycle.

3.6.1 The Pseudoinverse Method

One way to choose $\Delta \boldsymbol{q}$ is to solve Eq. 3.3:

$$\Delta \boldsymbol{q} = J^{-1} \Delta \boldsymbol{e}.$$

But inverting the Jacobian J is nontrivial. It can not be assured that the matrix is square or invertible at all. Besides, the resulting $\Delta \boldsymbol{q}$ may be misleading if J is nearly singular.

The pseudoinverse method (PI) uses a generalized inverse, namely the Moore-Penrose pseudoinverse of J, denoted $J^{\dagger}$, in place of the inverse:

$$\Delta \boldsymbol{q} = \alpha J^{\dagger} \Delta \boldsymbol{e}.$$

The pseudoinverse is unique and can be calculated for all matrices J, even for those which are not square or of full rank.

PI gives the best solution to the equation $J\Delta \boldsymbol{q} = \Delta \boldsymbol{e}$ in a least-squares sense. Hence, even if $\Delta \boldsymbol{e}$ is not in the range of J and $J\Delta \boldsymbol{q} = \Delta \boldsymbol{e}$ is not possible (e.g., when the desired target position is outside the reachable workspace), $\Delta \boldsymbol{q}$ has the nice property to minimize $\|J\Delta \boldsymbol{q} - \Delta \boldsymbol{e}\|^2$. Furthermore, it is the unique vector of smallest magnitude, given that there is more than one solution.

In the neighborhood of a singularity, the pseudoinverse method produces unwanted oscillating behavior. Especially when a target position is too distant, the kinematic chain has to be fully extended, encompassing a singular configuration.

One approach to reduce this problem is to clamp the gap between end-effector and target position, using

$$\Delta \boldsymbol{e}_{clamped} = \begin{cases} \Delta \boldsymbol{e} & \text{if } \|\Delta \boldsymbol{e}\|_2 \leq d_{max} \\ d_{max} \frac{\Delta \boldsymbol{e}}{\|\Delta \boldsymbol{e}\|_2} & \text{otherwise,} \end{cases}$$

with $\|\|_2$ being the Euclidean norm and d_{max} a threshold.

A second procedure avoiding shaking and jittering of the manipulator consists of a clamping of the joint updates:

$$\Delta \boldsymbol{q}_{clamped} = \begin{cases} \Delta \boldsymbol{q} & \text{if } \|\Delta \boldsymbol{q}\|_1 \leq a_{max} \\ a_{max} \frac{\Delta \boldsymbol{q}}{\|\Delta \boldsymbol{q}\|_1} & \text{otherwise,} \end{cases}$$

where $\|\|_1$ is the Manhattan norm and a_{max} a threshold. Care should be taken when mixing prismatic and revolute joints.

Exploiting Nullspace

The matrix $\left(I - J^{\dagger}J\right)$ performing a projection onto the nullspace of J is another useful feature of the pseudoinverse. This means that for any vector $\boldsymbol{\xi}$, $J\left(I - J^{\dagger}J\right)\boldsymbol{\xi} = \mathbf{0}$. Incorporating this additional term into the joint update

$$\Delta \boldsymbol{q} = \alpha J^{\dagger} \Delta \boldsymbol{e} + \left(I - J^{\dagger}J\right)\boldsymbol{\xi}$$

still yields an optimal value for $\Delta\boldsymbol{q}$ which minimizes $\|J\Delta\boldsymbol{q} - \Delta\boldsymbol{e}\|^2$. But now, apart from tracking a target position, one can satisfy secondary goals by choosing ξ properly.

The nullspace can also be viewed in terms of redundancy, i.e., the nullspace projection selects an optimal $\Delta\boldsymbol{q}$ from the set of homogeneous solutions in matters of the secondary criterion.

Many authors have exploited the nullspace method, e.g., to avoid collisions with obstacles, joint limits (the first application of the nullspace method, cf. [37]), or even singular configurations (by returning the joint angles back to rest positions or maximizing the manipulability measure, as defined in 3.6.3).

3.6.2 The Jacobian Transpose Method

Instead of calculating the inverse or pseudo-inverse of the Jacobian one can also use the transposed matrix:

$$\Delta\boldsymbol{q} = \alpha J^T \Delta\boldsymbol{e}.$$

This method (JT) was first adopted for inverse kinematics by [64]. Obviously, the transposed Jacobian is not equivalent to the inverse. However, its use is justified by considering the principle of virtual work.

Let external forces f and torques m be combined in $F = [f_x, f_y, f_z, m_x, m_y, m_z]^T$. If we apply F at the end-effector this results in generalized forces τ at the joints of the manipulator (torques in case of revolute joints; forces in case of prismatic ones). Using the principle of virtual work for applied forces this equality is expressed by

$$\begin{aligned} & F^T \Delta\boldsymbol{e} = \boldsymbol{\tau}^T \Delta\boldsymbol{q} \\ \Rightarrow \quad & F^T J \Delta\boldsymbol{q} = \boldsymbol{\tau}^T \Delta\boldsymbol{q} \\ \Rightarrow \quad & \boldsymbol{\tau} = J^T F \qquad . \end{aligned}$$

Assuming that $\Delta\boldsymbol{e}$ is a force pulling the end-effector to its target pose and $\boldsymbol{\tau}$ the resulting vector of joint velocities, we conclude

$$\dot{\boldsymbol{q}} = J^T \Delta\boldsymbol{e}. \tag{3.4}$$

When using constant time steps Δt this yields the new update step $\Delta\boldsymbol{q} = \dot{\boldsymbol{q}}$. In this simplified physical model force is proportional to velocity ($f = mv$) rather than acceleration, neglecting the effects of inertia. But although $\Delta\boldsymbol{q}$ is not exact it gives the right trend.

It turns out that Eq. 3.4 resembles the minimization method of steepest descent which reveals only suboptimal convergence behavior ([45]).

To improve the slow convergence rate one could introduce an additional scaling factor α: $\dot{\boldsymbol{q}} = \alpha J^T \Delta\boldsymbol{e}$. The value α can be thought of as a time step, leading to:

$$\frac{\Delta\boldsymbol{q}}{\Delta t} = \dot{\boldsymbol{q}}.$$

One reasonable way to choose α is so as to minimize the resulting error vector $\Delta\boldsymbol{e}$ after the update. Therefore, we assume that the change in the end-effector pose will be $\alpha JJ^T\Delta\boldsymbol{e}$. Now, α is chosen so as to make this value as close as possible to $\Delta\boldsymbol{e}$:

$$\alpha = \frac{\left\langle \Delta\boldsymbol{e}, JJ^T\Delta\boldsymbol{e} \right\rangle}{\left\langle JJ^T\Delta\boldsymbol{e}, JJ^T\Delta\boldsymbol{e} \right\rangle}.$$

3.6.3 The Damped Least-Squares Method

The damped least squares-method (DLS) is similar to the pseudoinverse technique but additionally avoids the problems which arise when acting near singularities. It is also known as the Levenberg-Marquardt method.

Instead of just minimizing $\|J\Delta\boldsymbol{q} - \Delta\boldsymbol{e}\|^2$ by finding a minimum vector $\Delta\boldsymbol{q}$, DLS minimizes

$$\|J\Delta\boldsymbol{q} - \Delta\boldsymbol{e}\|^2 + \lambda^2\|\Delta\boldsymbol{q}\|^2 = \left\| \begin{pmatrix} J \\ \lambda I \end{pmatrix} \Delta\boldsymbol{q} - \begin{pmatrix} \boldsymbol{e} \\ \boldsymbol{0} \end{pmatrix} \right\|^2,$$

where $\lambda \in \mathbb{R} > 0$ is called the damping constant.

The relevant normal equation

$$\begin{pmatrix} J \\ \lambda I \end{pmatrix}^T \begin{pmatrix} J \\ \lambda I \end{pmatrix} \Delta\boldsymbol{q} = \begin{pmatrix} J \\ \lambda I \end{pmatrix}^T \begin{pmatrix} \boldsymbol{e} \\ \boldsymbol{0} \end{pmatrix}$$

leads to the joint updates

$$\Delta\boldsymbol{q} = \left(J^TJ + \lambda^2 I\right)^{-1} J^T\boldsymbol{e} = J^T\left(JJ^T + \lambda^2 I\right)^{-1}\boldsymbol{e}. \tag{3.5}$$

The two notations in 3.5 differ only by the size of the square matrix being inverted. While in the first case the dimension of J^TJ depends on the DoFs of the manipulator, the second notation JJ^T produces a matrix dimension equal to the dimension of the task space. Since almost always $dim(taskspace) \ll DoF$ holds, the latter expression should be preferred.

The Damping Factor λ

The choice of the damping constant λ has a high impact on the numerical stability of Eq. 3.5. A small λ induces ill-behaved solutions near singularities. In fact, $\lambda = 0$ yields the pseudoinverse method. On the other hand, the larger λ gets the slower is the convergence rate.

There are numerous approaches that tackle this trade-off by adjusting λ dynamically (an overview is given in [17]). They cause low λ-values away from singularities, where no damping is needed and high values in the vicinity of singularities, evading infeasible large joint velocities. Nearly all those techniques rely on measures retrieved from the Jacobian, such as the manipulability measure, condition number or minimum singular value.

The determinant of a singular matrix is zero. Thus, the manipulability measure $w = \sqrt{det(JJ^T)}$ indicates the closeness of a manipulator to a singular configuration. In [43], w is used to update the damping factor

$$\lambda = \begin{cases} \lambda_0 \left(1 - \frac{w}{w_t}\right)^2 & \text{, if } w < w_t \\ 0 & \text{, otherwise,} \end{cases}$$

where w_t is a threshold. No damping is applied when $w \geq w_t$, otherwise damping is progressively increased until the maximum value λ_0 is reached in case of $w = 0$.

An extension to this scheme is to compute λ based on the rate of change of the manipulability measure w, as done in [32]. Consequently, the damping factor is more independent from the actual scaling of the manipulator.

The condition number c of an arbitrary matrix is the ratio of its largest and smallest singular value. One can obtain the singular values using the singular value decomposition, which describes a $m \times n$ matrix J as the product

$$J = UDV^T,$$

where U is a $m \times n$, D a $n \times n$ and V a $n \times n$ matrix. D is a diagonal matrix containing non-negative diagonal elements termed singular values. If at least one singular value is zero, J is singular. In this case, the condition number mentioned above ($c = \frac{\sigma_{max}}{\sigma_{min}}$) is infinite. The matrix J is ill-conditioned if c is too large. These properties of the condition number and the singular values can be used to infer the risk of a probable singularity.

To select the best IK method for our approach, we evaluated them in Section 6.4.

4 Data Acquisition and Preprocessing

4.1 Motion Capturing

The first major step of imitation is to record the motion trajectories of the human demonstrator while executing the skill. This includes the motion of an arm, i.e., its joint configuration as a function of time, as well as the six-dimensional pose (translation plus rotation) of other relevant interaction artifacts in the scene, such as a tool or a manipulated object. There are various techniques in the field of motion capturing, ranging from magnetic [66] over inertial [40] up to optical systems (overview in [41]). A cheap, flexible, though still reliable method only requires passive markers in conjunction with a monocular camera.

4.1.1 ARToolKit

The ARToolKit [30] is a software library providing the means to extract the orientation and position of a fiducial marker (black square on a white background) from a single camera image. It was originally designed for Augmented Reality applications, but has been successfully applied to motion capturing as well (e.g., see [56]).

Throughout the process, the image is thresholded, labeled into regions and an edge detection filter is applied. Lines are fitted on the resulting contour and corners are extracted at intersections. Using the fitted cube and the intrinsic camera parameters, one can calculate the transformation matrix from the marker frame to the camera frame.

Not only the size of the markers needs to be known by the system a priori but also the patterns used to identify multiple markers in the scene uniquely. The identification is done via template matching between all given patterns and the normalized marker image. The pattern is also used to disambiguate the heading direction of the fitted cube. To reduce the false positive rate and inter-marker confusion rate data matrix-like patterns are employed (cf. [20]). Those data matrix patterns also implement an error correcting code (see [62]).

Shortcomings of the system include range issues (can only be dealt with by increasing marker size), partial occlusions and a significant sensitivity with respect to lighting conditions. Moreover, recognition results depend on the relative orientation of the marker to the camera (recognition degrades the more tilted the marker gets).

Figure 4.1: Marker detection using ARToolKit. During the demonstration of a task six dimensional marker poses are extracted from the camera image to estimate the pose of important objects and the arm configuration. Here, the recognized markers are plotted on top of a captured image.

Generally speaking, the position estimate can be expected to be more reliable than the orientation estimation.

4.1.2 Tracking using Kalman filters

To stabilize the noisy marker measurements over time and complete poses in case of missing marker detections a Kalman filter [63] is used. The Kalman filter is a stochastic state estimation technique for dynamic linear systems modeling the state $x_t \sim N(\boldsymbol{\mu}_t, \Sigma_t)$ as a normal distribution at each discrete time step t. Control inputs $\boldsymbol{u}_t$ and measurements $\boldsymbol{z}_t$, both assumed to be corrupted by Gaussian noise, are successively integrated using a prediction-correction structure. A Kalman filter is fully defined by the state transition model A_t, the control-input model B_t, the process noise covariance Q, the observation model H_t, and the measurement noise covariance R. The a priori estimate $\tilde{\boldsymbol{x}}_t \sim N\left(\tilde{\boldsymbol{\mu}}_t, \tilde{\Sigma}_t\right)$ is given by the prediction

$$\begin{aligned} \tilde{\boldsymbol{\mu}}_t &= A_t \boldsymbol{\mu}_{\boldsymbol{t-1}} + B_t \boldsymbol{u}_t && \text{and} \\ \tilde{\Sigma}_t &= A_t \Sigma_{t-1} A_t^T + Q. \end{aligned}$$

Integrating the measurement $\boldsymbol{z}_t$ leads to the a posteriori estimate

$$\begin{aligned} \boldsymbol{\mu}_t &= \tilde{\boldsymbol{\mu}}_t + K_t\left(\boldsymbol{z}_t - H\tilde{\boldsymbol{\mu}}_t s\right) && \text{and} \\ \Sigma_t &= \left(I - K_t H\right)\tilde{\Sigma}_t, \end{aligned}$$

where $K_t = \tilde{\Sigma}_t H^T \left(H\tilde{\Sigma}_t H^T + R\right)^{-1}$ is the so-called Kalman gain which weights the influence of the correcting observation.

The state of a marker is represented by the vector $[x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, v_\phi, v_\theta, v_\psi]^T$, combining the position and orientation as Euler angles in three-dimensional Euclidean space, as well as translational and rotational velocities. Matrix A_t describes

how the marker pose evolves without noise or control input. This change in position and orientation is only affected by the current estimated velocities:

$$A_t = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where Δt is the time since the last measurement. As there is no external control input, B_t is the zero matrix. Next, we do not sense any velocities but only the pose of a marker. Accordingly, the measurement matrix projects only onto the position and orientation components of the state vector:

$$H_t = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

The velocity estimates in the state vector are update according to the last two measured poses and their temporal difference. Suitable values for the measurement and process noise R and Q are discovered empirically.

4.1.3 Joint Configuration of the Human-Arm-Like Chain

To derive joint variables from marker poses an anthropomorphic arm model as described in Section 3.3.1 is used. Compounds of markers around the teacher's arm bypass the problem of occlusions as this frequently happens with only one camera observing the scene. Normally, not more than one marker of the same compound

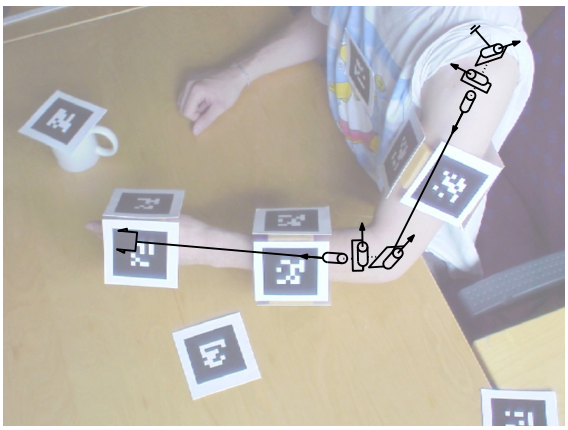

Figure 4.2: Deriving the joint angles using an antropomorphic arm model. To overcome the problem of occlusion, which occurs frequently when using only a single camera, we adopt compounds of markers applied at different segments of the instructor's arm.

is visible. But if two markers m_1 and m_2 referring to the same marker compound M are visible simultaneously (sensing more markers at the same time is impossible due to their mutual orthogonality within one compound), M turns to

$$trans\left(M\right) = lerp\left(trans\left(m_1\right), trans\left(m_2\right), \frac{visibility\left(m_1\right)}{\sum_{i=1}^{2} visibility\left(m_i\right)}\right)$$

$$rot\left(M\right) = slerp\left(rot\left(m_1\right), rot\left(m_2\right), \frac{visibility\left(m_1\right)}{\sum_{i=1}^{2} visibility\left(m_i\right)}\right),$$

where m_1, m_2 and M are homogeneous matrices and *trans* and *rot* mappings to their rotation and translation parts respectively. Function $lerp\left(p_1, p_2, \alpha\right)$ is a simple linear interpolation between two points p_1 and p_2 about factor $\alpha \in [0, 1]$, whereas *slerp* represents an interpolation method for rotations described in Section 3.1.4. The visibility of a marker is measured by the angle between its normal and the unit vector directed from the marker to the camera position:

$$visibility\left(m\right) = \langle rot\left(m\right)_z, \frac{trans\left(m\right) - trans\left(camera\right)}{\|trans\left(m\right) - trans\left(camera\right)\|}\rangle,$$

where *camera* is a homogeneous matrix and $rot\left(\cdot\right)_z$ denotes the third column vector of a rotation matrix in the camera frame. The operator $\langle\cdot,\cdot\rangle$ stands for the inner product of two vectors.

Each compound of markers represents one link in the kinematic chain. Given the measured rotation matrices of two marker compounds M_i and M_{i+1}, the connecting

joint j_i is defined by the rotation matrix

$$rot\,(J_i) = rot\left(M_{i+1}^{-1}\right) rot\,(M_i)\,.$$

To avoid the overdefinition of the joint angles due to the full marker observations, we ignore their translational part. The final joint configuration q_i of revolute joint i is formed by converting every joint rotation matrix into Euler angles as explained in Section 3.1.1. Because both, elbow and shoulder, are modelled as 3-DoF joints all three Euler angles can be considered without restriction. Since the end-effector of the arm plays an important role in our approach we sense it with an additional marker compound. Otherwise, the forward kinematics would sum up the errors of the joint angle measurements leading to an too inaccurate position estimate of the hand.

4.2 Aligning Multiple Demonstrations

Our approach relies heavily on demonstrating the same task multiple times in slightly varying contexts. When combining those demonstrations, the problem of different time scales arises. To cope with these temporal discrepancies, one could align two time series by a simple linear normalization. But a linear rescaling could not account for fluctuating movement velocities in-between repeated executions of the same skill.

A more powerful method called Dynamic Time Warping (DTW) is able to account for local distortions in the time domain resulting in a more intuitive nonlinear alignment. DTW was originally applied in speech recognition, as a distance measure between sequences of acoustic features (cf. [52]).

4.2.1 Dynamic Time Warping

Consider two discrete time series $\mathcal{X} = \{x_1, \ldots, x_{l_\mathcal{X}}\}$ and $\mathcal{Y} = \{y_1, \ldots, y_{l_\mathcal{Y}}\}$, both possessing the same constant sampling rate. Let $l_\mathcal{X}$ be the number of elements of time series $\mathcal{X}$. The results of applying the DTW are two warping functions $\phi_\mathcal{X} : \{1, \ldots, T\} \rightarrow \{1, \ldots, l_\mathcal{X}\}$ and $\phi_\mathcal{Y} : \{1, \ldots, T\} \rightarrow \{1, \ldots, l_\mathcal{Y}\}$ which relate the indices of $\mathcal{X}$ and $\mathcal{Y}$ respectively to a common time axis of length T. Therefore, an optimal alignment warp $\phi = (\phi_\mathcal{X}, \phi_\mathcal{Y})$ is computed, in the sense of a minimum overall distortion:

$$\phi = \underset{\phi}{\operatorname{argmin}}\, \mathcal{D}_\phi\,(\mathcal{X}, \mathcal{Y})\,. \tag{4.1}$$

The distortion or dissimilarity between two time series, given a warping function ϕ, is defined by

$$\mathcal{D}_\phi = \frac{1}{M_\phi} \sum_{k=1}^{T} d\,(\phi_\mathcal{X}(k), \phi_\mathcal{Y}(k))\, m(k),$$

where $m(k)$ is a nonnegative weighting coefficient, M_ϕ a normalizing factor, and $d(\cdot, \cdot)$ a distance measure. When aligning a series of workspace coordinates d is the Euclidean norm. However, in case of joint angles d becomes the 1-norm respecting the wrap-around at $\pi/-\pi$.

The minimization problem in Eq. 4.1 can be solved using dynamic programming techniques. First, a distance matrix $C := (d(\phi_\mathcal{X}(i), \phi_\mathcal{Y}(j)))_{i=1,\ldots,l_\mathcal{X};j=1,\ldots,l_\mathcal{Y}}$ is constructed (see Figure 4.3 for an example). Every warping function can be viewed as a path from one corner of the matrix to the other. Accordingly, its dissimilarity is the accumulated sum of all matrix entries along its way. For a warp ϕ to be meaningful, it has to satisfy the following restrictions:

- *Endpoint constraints*: The warping path has to start at the beginning of both time series and it has to end at their final states, i.e. $\phi_\mathcal{X}(1) = \phi_\mathcal{Y}(1) = 1$, $\phi_\mathcal{X}(T) = l_\mathcal{X}$ and $\phi_\mathcal{Y}(T) = l_\mathcal{Y}$.

- *Monotonicity conditions*: This preserves the chronological order within both time series $\mathcal{X}$ and $\mathcal{Y}$ by ensuring $\phi_\mathcal{X}(k+1) \geq \phi_\mathcal{X}(k)$ and $\phi_\mathcal{Y}(k+1) \geq \phi_\mathcal{Y}(k)$.

- *Local continuity constraints*: These heuristic-based constraints can take many forms and are normally expressed in terms of incremental path changes. Here, type I (cf. [49]) is used, allowing only three kinds of path increments, namely

$$(\phi_\mathcal{X}(k+1), \phi_\mathcal{Y}(k+1)) = \begin{cases} (\phi_\mathcal{X}(k), \phi_\mathcal{Y}(k)+1) \\ (\phi_\mathcal{X}(k)+1, \phi_\mathcal{Y}(k)) \\ (\phi_\mathcal{X}(k)+1, \phi_\mathcal{Y}(k)+1) \end{cases} .$$

  This scheme is directly related to the deletion, insertion and substitution operations during string alignment, leading to the so-called Levenshtein or edit distance.

- *Global path constraints*: There are two types of global path constraints. The first one follows directly from the local continuity constraints which can be characterized by their maximum and minimum possible path expansion (indicating the slope of a single path increment). Correspondingly, there may be certain portions of C that are excluded from the region the optimal warping path can traverse. In our case, the allowable region is not pruned by the local continuity constraints, as the maximum and minimum path expansion values are ∞ and 0 respectively.

  Another kind of global path constraints are range-limiting constraints restricting absolute differences in the warped time scales. They are utilized to avoid excessive time stretch and compression but are not applied here.

- *Slope weighting*: This refers to the weighting function $m(k)$ controlling the contribution of each distance $d(\phi_\mathcal{X}(k), \phi_\mathcal{Y}(k))$. On a "global" scale, the

weighting function can be designed to improve discrimination comparing time series similarity.

As this does not meet our intentions, we are only interested in its utility on a "local" scale. In this respect, the weighting function is usually regulated by the slope of the local path constraint. Among the numerous heuristic slope weighting functions, we apply the following type (cf. [49], often it is referred to as type (d)):

$$m(k) = \phi_{\mathcal{X}}(k) - \phi_{\mathcal{X}}(k-1) + \phi_{\mathcal{Y}}(k) - \phi_{\mathcal{Y}}(k-1),$$

which doubles the costs of a substitution. Consequently, the normalization factor M_ϕ turns to

$$\begin{aligned} M_\phi = \sum_{k=1}^{T} m(k) &= \sum_{k=1}^{T} \left(\phi_{\mathcal{X}}(k) - \phi_{\mathcal{X}}(k-1) + \phi_{\mathcal{Y}}(k) - \phi_{\mathcal{Y}}(k-1)\right) \\ &= \phi_{\mathcal{X}}(T) - \phi_{\mathcal{X}}(0) + \phi_{\mathcal{Y}}(T) - \phi_{\mathcal{Y}}(0) \\ &= l_{\mathcal{X}} + l_{\mathcal{Y}}. \end{aligned}$$

Applying dynamic programming the optimal warp path can be found efficiently by evaluating the following recurrence relation which defines the cumulative distance $\gamma(i, j)$ as the current weighted distance and the minimum of the cumulative distances obeying the local continuity constraints:

$$\gamma(i, j) = C(i, j) + \min\{\gamma(i-1, j), C(i, j) + \gamma(i-1, j-1), \gamma(i, j-1)\}.$$

Figure 4.3 shows an exemplary application of the DTW, parameterized as above. Here, we align two sequences of object positions in two different demonstrations.

Obviously, the time and space complexity lies in $\mathcal{O}(l_{\mathcal{X}} l_{\mathcal{Y}})$, i.e., it is quadratic in the length of the two sequences to be aligned. There exist variants of DTW which reduce this quadratic complexity (e.g. FastDTW [53], an approximative DTW which achieves linear complexity), but always to the expense of not finding the optimal warping path in certain situations.

4.2.2 Derivative Dynamic Time Warping

Though, DTW is successful at aligning local accelerations and decelerations in the time axis, it has major problems when local differences in the Y-axis occur. For example, two datapoints with identical Y-axis values would be mapped on each other, even if one is part of a rising, while the other belongs to a falling trend. Moreover, DTW explains local differences in the Y-axis (e.g., a shallow vs. a deep valley) in terms of the time axis producing unwanted singularities (one-to-many mappings).

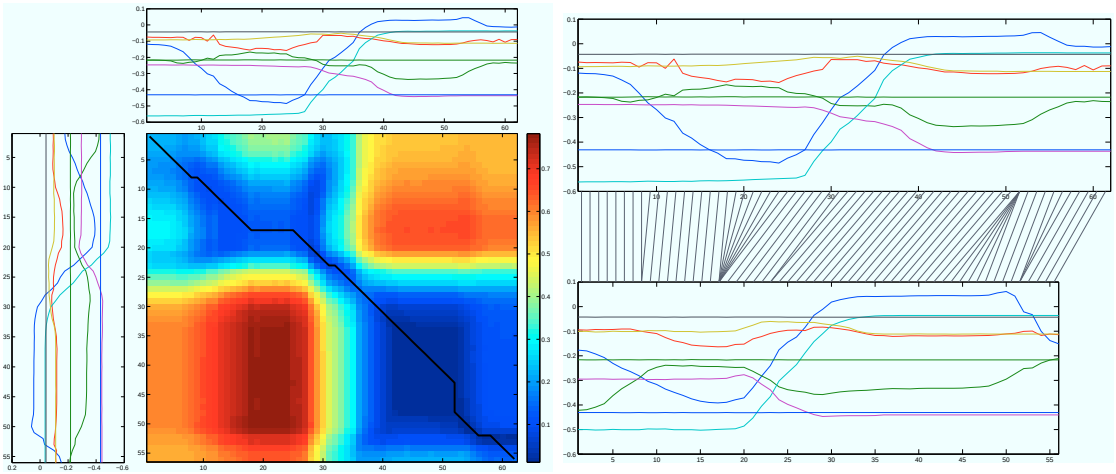

Figure 4.3: Multidimensional Dynamic Time Warping between two demonstrations of the same 3D object positions. **Left**: First, a distance matrix is computed between the two six dimensional functions consisting of 62 and 56 frames respectively. Each entry of the matrix describes the costs associated with the mapping between two frames of both sequences corresponding to row and column of the entry. Costs are defined as the Euclidean distance between the 3D positions given by the functions. Colors range from blue (small distance) to red (high distance). A warping path (bold black line) is calculated resembling the cheapest accumulated costs. **Right**: The found warping path defines a mapping between both functions, illustrated as connecting lines between the X-axes.

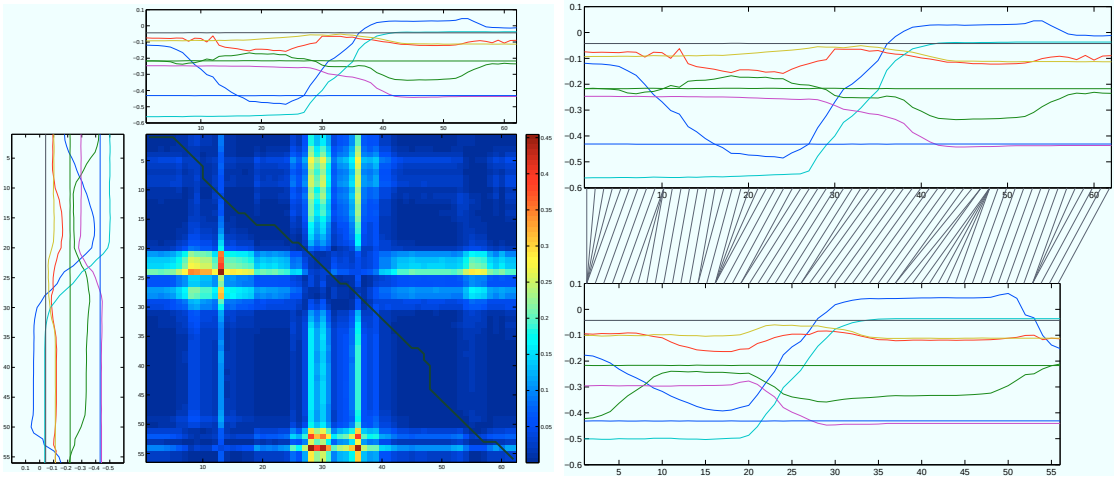

Figure 4.4: Multidimensional Derivative Dynamic Time Warping between two demonstrations of the same 3D object positions. **Left**: First, a distance matrix is computed between the two six dimensional functions consisting of 62 and 56 frames respectively. Each entry of the matrix describes the costs associated with the mapping between two frames of both sequences corresponding to row and column of the entry. Costs are defined as the first derivative of the Euclidean distance between the 3D positions given by the functions. Colors range from blue (small distance) to red (high distance). A warping path (bold black line) is calculated resembling the cheapest accumulated costs. **Right**: The found warping path defines a mapping between both functions, illustrated as connecting lines between the X-axes. Note, that in contrast to standard DTW, DDTW results in a more uniformly distributed mapping producing less pronounced singularities (one-to-many mappings).

To prevent these problems, one could consider higher-level features such as the shape of a time series. This information is given by the first derivative, resulting in an modification of DTW named Derivative Dynamic Time Warping (DDTW, cf. [33]). The algorithm only changes in the distance function

$$d\left(\phi_{\mathcal{X}}(k), \phi_{\mathcal{Y}}(k)\right) = \left(\dot{x}_{\phi_{\mathcal{X}}(k)} - \dot{y}_{\phi_{\mathcal{Y}}(k)}\right)^2,$$

where $\dot{x}_i = \frac{1}{2}\left(x_i - x_{i-1} + \frac{1}{2}\left(x_{i+1} - x_{i-1}\right)\right)$ is a robust estimation of the first derivative at i. At the borders ($i = 0$ and $i = l_{\mathcal{X}}$) the estimate is not defined and therefore substituted by the neighboring ones.

Figure 4.4 shows the application of the DDTW to the same data used in Figure 4.3. In contrast to standard DTW, it can be clearly seen that the resulting mapping possesses less pronounced singularities.

4.2.3 Multiple Alignments

Another problem arises from the fact, that we need to align n demonstrations and often enough $n \gg 2$. If the search space is expanded to n dimensions, the

DTW complexity becomes exponential in the number of demonstrations n. Hence, aligning relatively short time series becomes already infeasible.

Alternatively, one can separate the multiple alignment into pairwise alignments and combining them in a binary tree-like manner. The resulting warping path is suboptimal, depending on the selection of the time series pairs to align. Regarding time complexity, this technique only scales linearly in n.

To align multiple demonstrations of object poses and joint angles in our framework, we utilize DDTW in conjunction with a pairwaise binary tree-like approach.

5 Approach

5.1 Imitation Learning Framework

The procedure of learning by imitation consists of two separate phases. During a first passive part, the robot observes a task performed by the human teacher. In our approach it is essential that the demonstrator carries out the action multiple times. The robot exploits these repeated demonstrations to infer the relevant aspects of a task. This results in an internal task description, the final product of the learning phase.

Subsequently, the robot applies the newly acquired knowledge by acting in a similar setting. This step is called reproduction phase and leads to a visible imitation of the demonstrator.

5.1.1 Computational View

From a computational point of view, we can divide the framework into two consecutively executed parts, as depicted in the schematic outline of Figure 5.1.1. The first part (perception) is responsible for building up a probabilistic representation of the demonstrated skill at a trajectory level. A set of demonstrations given by a human teacher is recognized using a monocular camera and aligned to a fixed time axis via a Dynamic Time Warping method. Chapter 4 describes the details of this process. While the set of demonstrations can continuously grow, a more generalized plan of the skill can be extracted, resembling an incremental learning technique.

During the second part (action) the model of the skill, which is encoded by constraints in Cartesian and joint space, is applied to a new, probably slightly varying context. The final action is calculated by means of inverse kinematics presented in Section 3.6. Note, that the imitation framework is an extension and modification of the system from Calinon and Billard (see, e.g., [10]). Throughout the following sections the individual phases of the imitation procedure are examined in detail.

5.1.2 Imitation via Inference in a Dynamic Bayesian Network

Let us take a more formal look at the problem of imitation and treat it as a stochastic process of length T. We model the process with a dynamic Bayesian network (DBN), depicted in Figure 5.2. During learning, it is used to infer important constraints, which encode relevant aspects of the task. During reproduction we can

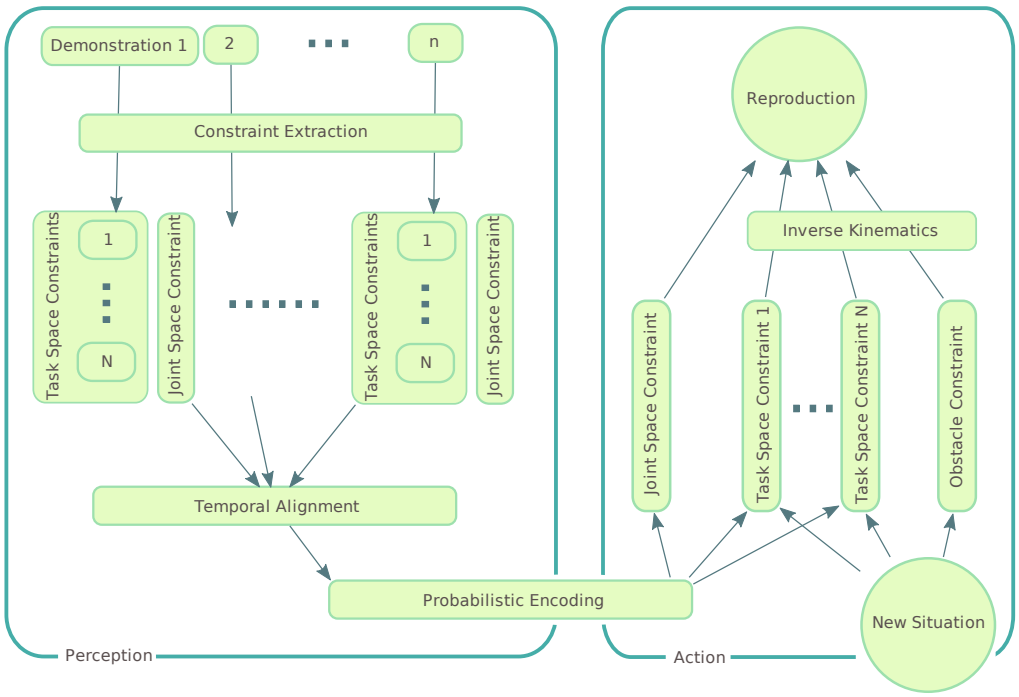

Figure 5.1: The data flow in our imitation framework. Starting with the captured poses of various demonstrations of the same task (upper left) we extract relations in joint space and workspace. After aligning them to a unifed time domain, we derive mean and variance. These relations are then applied in a slightly varying setting. We transform the relations in workspace to joint actions via inverse kinematics which lead to the reproduction of the task (upper right).

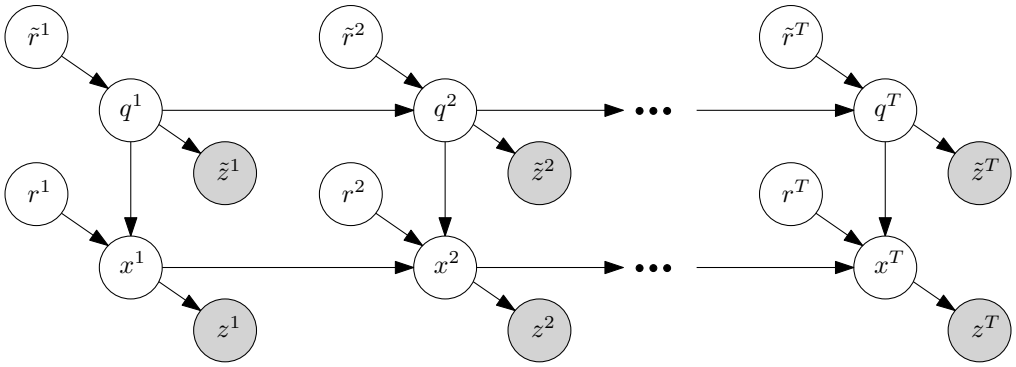

Figure 5.2: Dynamic Bayesian Network (DBN) used for imitation of tasks. The arrows indicate conditional dependencies between random variables. The task demonstration is discretized into T successive time steps. In each time step, $\tilde{z}$ denotes the observation of the demonstrators body configuration which is represented by the joint angles q. z is the observation of the world state x that encodes the position of relevant objects in the scene. The relations between objects r as well as relations of the joint angles $\tilde{r}$ are used to model the action that should be learned and carried out by the robot. During learning, the distributions over the relations $p(r)$ and $p(\tilde{r})$ need to be determined. During reproduction, these relations are known and the best body configuration of the robot q^* is estimated and executed by the robot.

use it to infer optimal actions obeying the learned constraints. Note that we will use the terms relation and constraint interchangeably, referring to the very same concept. The individual components of our DBN are explained as follows:

The latent variable $\boldsymbol{q\,(t)}$ denotes the arm configuration of the human demonstrator or the robot, depending on the phase being executed (learning or reproduction). Accordingly, the dimension of $q\,(t)$ equals the six DoFs of the human arm-like chain or our robotic manipulator.

To relax the problem of body correspondence we assume not only the same dimensionality of demonstrator and imitator but also an identity mapping between their joint variables. Note that this is not a prerequisite as a static mapping could always account for arbitrary differences in the kinematic structure between human and robot.

The evidence node $\boldsymbol{\tilde{z}\,(t)}$ represents the observation of the arm/manipulator configuration $q\,(t)$ at time step t, thus having the same dimensionality. The measurements are assumed to be corrupted by Gaussian noise. As a result, the observation model $p\,(\tilde{z} \mid q)$ is a normal distribution.

The latent variable $\boldsymbol{x\,(t)}$ includes the poses of all relevant objects in the scene at time t, given by

$$x\,(t) = \{x_1\,(t)\,, \ldots, x_n\,(t)\,, x_{a_1}\,(t)\,, \ldots, x_{a_m}\,(t)\},$$

where $x_1(t), \ldots, x_n(t)$ are the poses of n objects and $x_{a_1}(t), \ldots, x_{a_m}(t)$ represent the poses of m points located on the arm. Henceforward, we only consider one point of the arm, namely the end-effector whose pose at time step t is denoted by $x_E(t)$. The world state then becomes

$$x(t) = \{x_1(t), \ldots, x_n(t), x_E(t)\}.$$

Note that we can act on this assumption without loss of generality throughout the remaining calculus.

Depending on the modeled constraints in task space r, the dimensionality of x may vary. Often enough we will only use the Cartesian 3D position, but other dimensions such as orientation can be incorporated as well. Hence, x has $3(n+m)$ or $6(n+m)$ dimensions, respectively.

The evidence node $z(t)$ represents the observation of the object poses $x(t)$ at time step t and has the same dimensionality. The measurement noise is assumed to be Gaussian, leading to a normal distributed observation model $p(z \mid x)$.

The latent node $r(t)$ depicts the geometrical relations in task space, also called task constraints, changing over time t. These constraints describe relative differences in the 3D position between relevant objects in the scene and certain parts of the demonstrator's or robot's body. As already mentioned, we are neither limited to 3D positions nor to world coordinates in a fixed frame of reference.

This way the dimensionality of $r(t)$ sums up to $3nm$. As we are using the end-effector as the only body part the dimensions decrease to $3n$.

We assume that during multiple demonstrations of the same skill the task constraints differ in a Gaussian manner. Thus, the individual r_i can be represented by a mean and a variance for each dimension separately. In the beginning, before any demonstration is shown we have no information about the prior distributions $p(r_i)$. Hence, their variance is set to infinity.

A more detailed look on the modeling of constraints in task space is given in Section 5.2.1.

The latent node $\tilde{r}(t)$ specifies the absolute joint angle values corresponding to an arm configuration at time t. Its dimensionality complies with the number of DoFs of the demonstrator's and robot's body respectively. This leads to six joint angles in our case. Accordingly, $\tilde{r}(t)$ is also referred to as the joint constraints. As it is the case with the constraints in task space, we assume the differences between joint constraints over multiple demonstrations to be Gaussian distributed. Thus, we represent them with a mean and a variance for each joint.

A more detailed look on the modeling of constraints in configuration space is given in Section 5.2.2.

Calculating the Joint Probability of the DBN

To simplify matters we consider only a single time slice of the DBN, henceforth neglecting the time index t. In a Bayesian network the joint distribution of the node values can be written as

$$p(X_1, \ldots, X_N) = \prod_{i=1}^{N} p(X_i \mid parents(X_i)),$$

where the set $parents(X_i)$ refers to all nodes $X_{j \neq i}$ for which exists a directed arc from X_j to X_i. Thus, the joint probability distribution of the described DBN (see Figure 5.2) is given by

$$p(\tilde{r}, q, \tilde{z}, r, x, z) = p(z \mid x) \cdot p(\tilde{z} \mid q) \cdot p(\tilde{r}) \cdot p(r) \cdot p(x \mid q, r) \cdot p(q \mid \tilde{r}),$$

with the definitions of each individual distribution given above.

The posterior about the poses of the objects x in the scene given the arm configuration q and the constraints in task space r is defined as

$$\begin{aligned} p(x \mid q, r) &= p(x_E, x_1, \ldots x_n \mid q, r) && (5.1)\\ &= p(x_E \mid q, r) \cdot p(x_1, \ldots x_n \mid x_E, q, r) && (5.2)\\ &= p(x_E \mid q) \cdot p(x_1, \ldots x_n \mid x_E, r). && (5.3) \end{aligned}$$

The conversion from Eq. 5.1 to 5.2 is done by applying the product rule. Further on, we assume that the poses of objects are independent from the joint configuration given the end-effector. Similarly, we assume independence between the end-effector pose and task space constraints given the joint configuration, obtaining Eq. 5.3. Repeated applications of the product rule lead to:

$$\begin{aligned} p(x \mid q, r) &= p(x_E \mid q) \cdot p(x_1 \mid x_E, r) \cdot p(x_2, \ldots, x_n \mid x_1, x_E, r)\\ &= p(x_E \mid q) \cdot p(x_1 \mid x_E, r) \cdot p(x_2, \ldots, x_n \mid x_E, r) && (5.4)\\ &= p(x_E \mid q) \cdot \prod_{i=1}^{n} p(x_i \mid x_E, r_i) && (5.5)\\ &\approx p(x_E \mid q) \cdot \prod_{i=1}^{n} \mathcal{N}_{r_i}(x_i - x_E). && (5.6) \end{aligned}$$

Assuming that the poses of any two objects in the scene are independent given the constraints and the end-effector results in Eq. 5.4. In conjunction with the product rule we apply this assumption n times and obtain Eq. 5.5.

Finally, the posterior about the pose of the end-effector $p(x_E \mid q)$ is assumed to be Gaussian distributed and directly corresponds to the kinematic function.

5.2 Modeling Constraints

The key ingredient of our approach to imitation are the constraints. They capture the essential nature of a skill, still allowing for an adaptable reproduction in varying setups. We distinguish between two types of constraints: those defined in the workspace of the actor and those qualifying her configuration space.

Given joint configurations can be estimated reliably by the robot observing a human task demonstration, they can be used to calculate task and joint constraints representing a skill in a flexible and more general way.

5.2.1 Task Space Constraints

Consider the skill of picking up a mug and placing it at an arbitrary destination. Regardless of the mug's exact initial pose, the relative displacement between hand (or end-effector in general) and mug always tends to the same constant (ideally zero). This spatial relationship as a function of time is named task space constraint. Certainly, the same holds for the relation between hand and target position.

With only one demonstration being present, each task constraint carries one single relative trajectory making it hard to generalize. But as soon as there are more demonstrations shown, we can independently assign variances to each constraint at every time step (the learning phase, see Section 5.3). Using these variances we can weigh the importance of individual constraints when combining them.

Figure 5.3 illustrates both constraints of the pick-and-place task described earlier. It can be seen easily that the significance of the two constraints (i.e., low variance) reaches its highest level at fundamentally different time steps within the skill execution. These moments are characterized by the grasping and releasing of the mug.

We can formally define the constraints in task space as a set of binary relations between objects x_i in the scene and parts of the demonstrator's or robot's body x_{a_j} depending on time t:

$$r(t) = \{r_{ij}(t) \mid r_{ij}(t) = \left(x_i(t), x_{a_j}(t)\right) \in \{x_1(t), \ldots, x_n(t)\} \times \{x_{a_1}(t), \ldots, x_{a_m}(t)\}\}.$$

However, if we allow the end-effector x_E to be the only interacting body part, the task constraints are simplified to

$$\begin{aligned} r(t) &= \{r_{ij}(t) \mid r_{ij}(t) = (x_i(t), x_E(t))\} \\ &= \{r_1(t), \ldots, r_n(t)\}, \end{aligned}$$

where n equals the number of relevant objects in the scene.

As the modeling of task constraints is done manually, two questions arise:

- Which are relevant interacting objects in the scene, i.e., how to select essential relations r_{ij}?

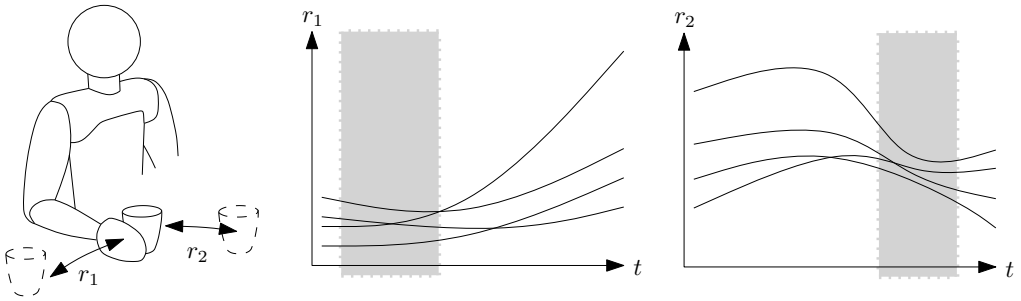

Figure 5.3: Schematic illustration of learning a constraint in task space. In this example, a human moves a mug from A to B. The constraints r_1 and r_2 correspond to the distance between the hand and the starting and target position respectively. Imagine four arbitrary demonstrations, where the human hand always takes a slightly different path to transport the mug from A to B. Here, the shadowed regions in the two plots show the invariant features of this task. While during an early stage of task execution it is important to satisfy constraint r_1 (move the hand to A in order to grasp the mug), near the end, it is crucial to obey r_2, i.e., moving the hand and mug to the target position B.

- What are appropriate dimensions of a constraint?

Both questions highly affect the degree of skill generalization. The selection of relevant interaction elements is intuitively dictated by the nature of a task. While in the pick-and-place task these are end-effector, start, and target position of the manipulated object, in a door-opening task, end-effector and door knob form an essential task constraint. Apart from defining task constraints always relative to the end-effector, one can easily think of skills involving other relevant body parts x_{a_j}, e.g., in an arm wrestling task (elbow and table).

The question referring to the dimensionality of a task constraint is governed as well by the kind of task being imitated. In most cases, three dimensions in Cartesian space are sufficient. However, when carrying a glass of vine it is preferable to imitate its upright position while being grasped. Hence, we have to expand the dimensions of the task constraints, incorporating orientation.

Furthermore, the relations r_{ij} can be described in terms of a global fixed frame of reference or local object-fixed frames. The latter is invariant to rotation, allowing for a higher degree of generalization. In case of the pick-and-place task for example, the constraint between the mug's initial pose and end-effector would encode the displacement of the handle w.r.t. the mug's center. Moreover, imitating the cleaning of a whiteboard (four constraints; corners vs. end-effector) could account for arbitrary rotated rectangular shapes.

Sticking with the pick-and-place example, we haven't captured the grasping behavior yet. So far, it is virtually impossible to explain it in terms of task space constraints using only one single end-effector. Nevertheless, it can be achieved by introducing constraints in the configuration space.

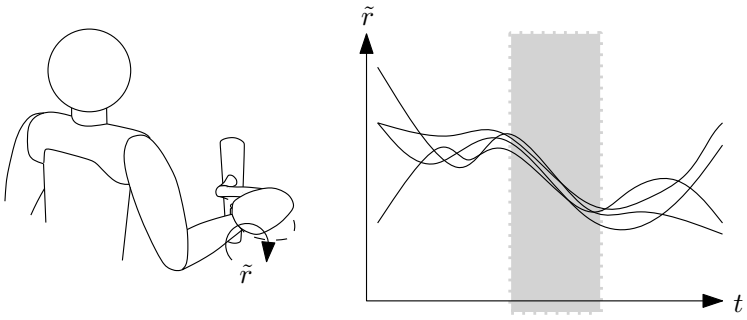

Figure 5.4: Schematic illustration of learning a constraint in configuration space. Imagine a human opening a door. The constraint $\tilde{r}$ describes the joint angle of the wrist. After four demonstrations of the task, the invariant feature emerges (emphasized by the shadowed region in the plot). Despite varying motions of the wrist joint during the approaching to and retraction from the door knob, the pressing down action always results in the same angle shift. This feature can be used to reproduce the action.

5.2.2 Configuration Space Constraints

Imagine the skill of opening a door by pushing down the door knob. Irrespective of how the arm approaches the knob, the wrist joint always performs the same angular motion during the short moment when the knob is pushed down. This is illustrated in Figure 5.4. The same holds for the grasping movement during the pick-and-place task. Only the type of joint differs. These absolute joint configurations of the demonstrator's or imitator's kinematic structure as a function of time are named joint or configuration space constraints.

We define the joint constraints of a kinematic structure with n DoFs at time t as

$$\tilde{r}\left(t\right) = \left(\tilde{r}_1\left(t\right), \ldots, \tilde{r}_n\left(t\right)\right),$$

where each $\tilde{r}_i\left(t\right) \sim \mathcal{N}\left(\mu_i, \Sigma_i\right)$. As our kinematic chains only consist of revolute joints, $\mu_i \in (-\pi, \pi]$ holds.

The major drawback of joint constraints is their dependency on a particular kinematic structure. Even manipulative structures with equal DoFs imitate each other arbitrarily bad when controlled solely on the basis of joint constraints. During reproduction the relative changes in joint constraints, i.e., the joint velocities, are used. However, reproduction becomes strongly biased because the initial configuration has to be set according to $\mu\left(0\right)$. Hence, constraints in configuration space lack generalization.

To represent task knowledge in a model-free mode, we therefor do not to apply neither joint space constraints nor task space constraints involving any other body part than the end-effector.

5.3 Learning Constraints

Based on the alignment of D demonstrations to a unified time axis T it is possible to obtain the constraints r and $\tilde{r}$ over time. We assume Gaussian distributed trajectories of the end-effector and the relevant objects over all observed demonstrations. Accordingly, also their differences, namely the task constraints, are Gaussian distributed, which applies to the joint constraints as well. Formally, a constraint r_i is fully described by a 3D mean vector $\boldsymbol{\mu}_i$ and a diagonal matrix Σ_i (assuming the relations in x, y, and z to be independent):

$$r_i(t) = \mathcal{N}(\boldsymbol{\mu}_i(t), \Sigma_i(t)) = \mathcal{N}\left(\begin{pmatrix}\mu_{i_x}(t)\\ \mu_{i_y}(t)\\ \mu_{i_z}(t)\end{pmatrix}, \begin{pmatrix}\sigma^2_{i_x}(t) & 0 & 0\\ 0 & \sigma^2_{i_y}(t) & 0\\ 0 & 0 & \sigma^2_{i_z}(t)\end{pmatrix}\right).$$

The same accounts for constraints in configuration space $\tilde{r}_i(t)$. Especially the variance is a key element for the tasks descriptions since it describes how accurately the demonstrator enforced this relation during the demonstrations.

To calculate the models $r_i(t)$ and $\tilde{r}_i(t)$, we could easily derive the first and second moment by directly evaluating each discrete time step separately. However, we typically have to deal with a rather small number of demonstrations and therefore rather rough and non-smooth estimates would be obtained.

To overcome this problem we exploit function regression techniques. Generally speaking, regression methods are divided into parametric and non-parametric approaches. Both learn the model underlying a noisy stochastic process by means of training samples. While parametric approaches to regression make strong assumptions about the underlying model they search for, non-parametric techniques, also known as memory- or instance-based, rely directly on the training data when predicting unknown values. We evaluate two non-parametric regression approaches to learn $r_i(t)$ and $\tilde{r}_i(t)$.

5.3.1 Locally Weighted Regression

In density estimation, the kernel estimator, also referred to as Parzen window, approximates the probability density function (pdf) of a random variable based on a set of samples. We expect the constraints at a single time step t to be normal distributed and consequently do not need to estimate arbitrary pdfs. Instead we use the kernel regression to estimate smoothly varying Gaussians $r_i(t)$ and $\tilde{r}_i(t)$ over time. The link from kernel density estimation to regression is reconstructed in Section A in the Appendix.

To compute a relation estimate between objects in the scene x_i and the end-effector x_E, we introduce relation samples $(t, l(t,d,i))$ consisting of time and relative position, with

$$l(t,d,i) = x_i^d(t) - x_E^d(t),$$

where $t = 1, \ldots, T$ and $d = 1, \ldots, D$. D reflects the number of demonstrations. Then, we obtain the kernel regressor (see [42])

$$\boldsymbol{\mu}_i(t') = \frac{\sum_{d=1}^{D} \sum_{t=1}^{T} \kappa\left(\frac{t'-t}{h}\right) l(t,d,i)}{\sum_{d=1}^{D} \sum_{t=1}^{T} \kappa\left(\frac{t'-t}{h}\right)},$$

where κ is some kernel function and h the kernel bandwidth, both determining the influence of neighboring data points.

All instances $l(t,d,i)$ might have an effect on the estimate at t', depending on the form of the kernel and its bandwidth chosen. When applying a homogeneous kernel this effect decreases gradually as the distance $|t'-t|$ increases. The detailed effect of kernel type and bandwidth is explained in the next sections.

Similarly to the mean function, the variance can be estimated given the sample points:

$$\boldsymbol{\sigma}_i^2(t') = \frac{\sum_{d=1}^{D} \sum_{t=1}^{T} \kappa\left(\frac{t'-t}{h}\right) \left(l(t,d,i) - \mu_i(t')\right)^2}{\sum_{d=1}^{D} \sum_{t=1}^{T} \kappa\left(\frac{t'-t}{h}\right)}.$$

This procedure is carried for each object in the scene and accordingly done for the constraints in joint space $\tilde{r}$. The only difference lies in the calculation of the relation samples:

$$\tilde{l}(t,d,i) = \boldsymbol{q}^d(t),$$

where $\boldsymbol{q}(t)$ is the demonstrated joint configuration of the i-th demonstration at time step t.

Kernel Function κ

In theory, any function can be chosen to be κ, as long as it satisfies the following constraints:

$$\forall x : \kappa(x) \geq 0, \qquad \int_{-\infty}^{\infty} \kappa(x)\, dx = 1, \qquad \int_{-\infty}^{\infty} x\kappa(x)\, dx = 0, \qquad \int_{-\infty}^{\infty} x^2\kappa(x)\, dx = 0.$$

This means that κ has to be a symmetric positive function integrating to one.

Mostly, kernels have the property of being a function only of the difference between the arguments, i.e. $\kappa(u) \sim \kappa(x - X_i)$, with $u \in \mathbb{R}$ in the univariate case. These are invariant to translations in input space and thus known as stationary kernels.

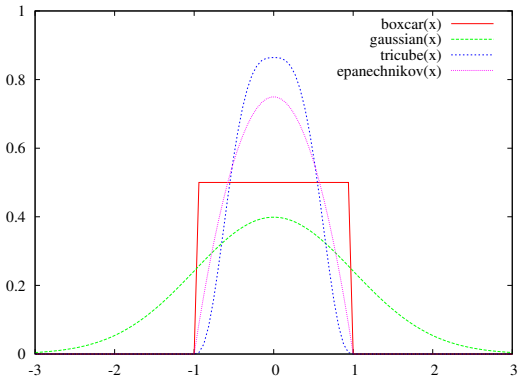

Figure 5.5: Four commonly used kernels with bandwidth 1

As a further specialization, some kernels only depend on the magnitude of the distance (typically the Euclidean one) between the arguments. We have exploited this feature above, which defines $\kappa(u) = \kappa(|x - X_i|)$. Kernels obeying such characteristics are classified as homogeneous or named radial basis functions. The most commonly used homogeneous kernels are:

- *Boxcar*: $\kappa(u) = \begin{cases} \frac{1}{2} & \text{, if } |u| < 1 \\ 0 & \text{, otherwise} \end{cases}$
- *Gaussian*: $\kappa(u) = \frac{1}{\sqrt{2\pi}} e^{\frac{1}{2}u^2}$
- *Tricube*: $\kappa(u) = \begin{cases} \frac{70}{81}(1 - |u|^3)^3 & \text{, if } |u| < 1 \\ 0 & \text{, otherwise} \end{cases}$
- *Epanechnikov*: $\kappa(u) = \begin{cases} \frac{3}{4}(1 - u^2) & \text{, if } |u| < 1 \\ 0 & \text{, otherwise} \end{cases}$

A visualization of these weighting functions is depicted in Figure 5.5. We select the standard choice for κ, namely the Gaussian or squared exponential kernel. But note that the choice of an appropriate bandwidth h is much more relevant for the accuracy of an estimate.

Bandwith h

The kernel bandwidth h, sometimes called smoothening parameter, governs the width of the kernel. A higher value for h leads to a smoother but concurrently more imprecise estimate (see Figure 5.6). For $h \to \infty$ the estimated pdf becomes the global average, while $h \to 0$ results in a nearest neighbor estimate.

Selecting h automatically is most frequently done by reference rules or methods of cross-validation (for a comparison see [29]).

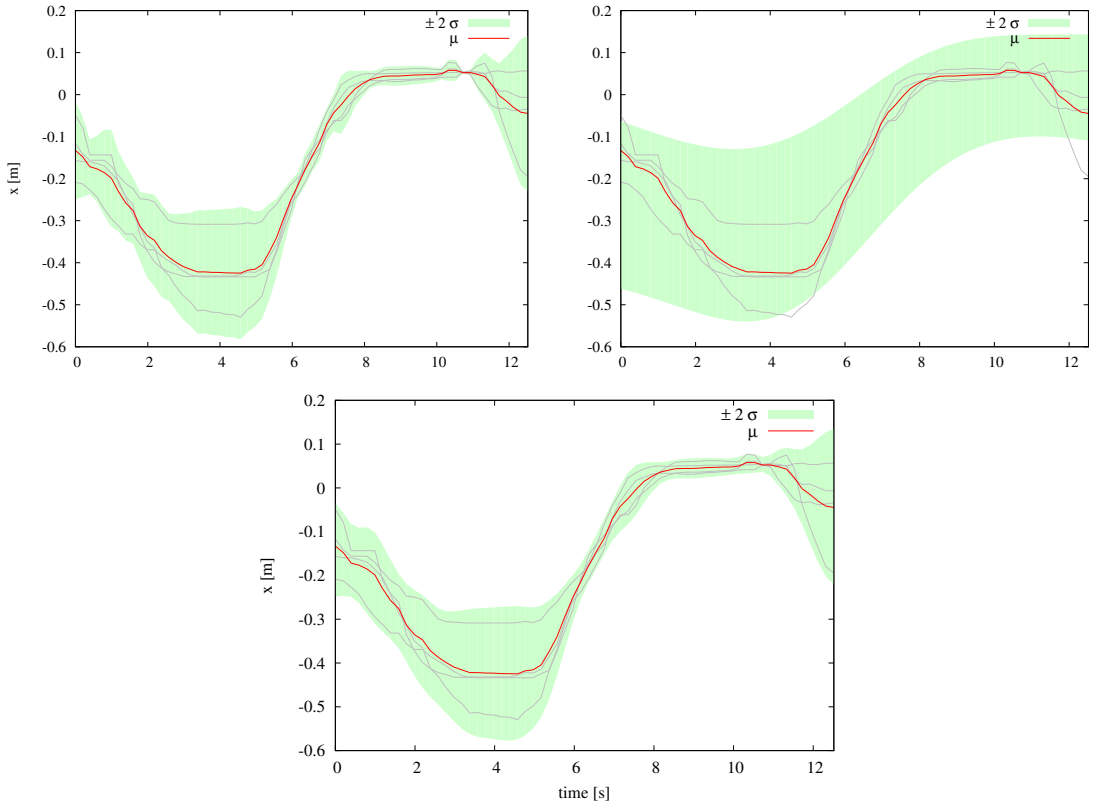

Figure 5.6: Over- vs. underfitting using kernel regression. The same data (gray functions) was regressed applying a Gaussian kernel with varying bandwidth parameter. We estimated a smooth mean and variance function. A small kernel bandwidth ($h = 0.02$) results in an overfitting the data (upper left). In contrast, a nearly constant estimate (upper right), underfitting the data, is obtained by a too large bandwidth ($h = 2.0$). We empirically selected $h = 0.2$ avoiding both extremes (lower plot).

Reference rules make assumptions about the underlying pdf, e.g. a normal density, and find the optimal bandwidth h^* by minimizing the integrated mean square error given that assumption. These rules of thumb produce surprisingly robust results even in case of false assumptions, e.g. non-normal densities.

Methods of cross-validation set h to minimize the crossvalidated error, e.g. the integrated square error, on the training set. Consequently, the selected bandwidth automatically adapts to the smoothness of r_i. The important assumption, that the distribution of the training set is a representative of the underlying pdf may not always be the case.

To regress our relations in task and joint space we use a Gaussian kernel with an empirically derived bandwidth. Inferring h empirically is easily possible as we are only dealing with univariate data, making visualizations manageable. The techniques for automatically selecting the bandwidth outlined above are thus rendered unnecessary.

A general limitation of kernel regression is the poor boundary behavior, often shown by a flattened estimate, in spite of a visible nonstopping trend in the data. Although this is not a major problem in our case (as can be seen in Figure 5.6), in general it can be coped with by using a higher degree local polynomial regression.

The computational complexity of using kernel regression lies in $\mathcal{O}(dT^2)$, i.e., it is quadratic in the number of sample points. We can reduce the complexity by bounding the application of the kernel to the $k \ll dT$ nearest neighbors, e.g., only those not farer away than $3h$. Though the squared exponential kernel never gets zero, sample points outside the 99.7% confidence interval contribute only insignificantly. In this way, the runtime decreases to the linear complexity of $\mathcal{O}(kT)$.

5.3.2 Gaussian Process Regression

Models based on Gaussian processes (GPs, cf. [50]) have become commonplace for problems of regression. The main assumption of GPs is that any finite set of values resulting from an unknown underlying function is Gaussian distributed. This normal distribution is defined by a zero mean (training data has to be preprocessed to keep this condition) and a covariance matrix $K = (k)_{mn}$ given by some covariance function $k_{mn} = cov(s_m, s_n)$. In our case, the target values s are enumerated sample points of constraint i, depending on time and number of demonstration: $s_m = l(t, d, i) = l\left(m \bmod T, \lceil \frac{m}{D} \rceil, i\right)$, where $m \in [1, \ldots, T + D]$. Consequently, the joint distribution of the training data together with the data to predict is again Gaussian. Furthermore, the distribution $pdf(s' \mid s_1, \ldots s_{T+D})$ of the prediction s' is a conditional of a Gaussian and thus again Gaussian. Its mean and variance are calculated by

$$\begin{aligned} \mu &= \mathbb{E}[s' \mid s_1, \ldots, s_{T+D}] = \boldsymbol{k}^T K^{-1} \boldsymbol{t} \\ \sigma^2 &= \mathbb{V}[s' \mid s_1, \ldots, s_{T+D}] = v - \boldsymbol{k}^T K^{-1} \boldsymbol{t}, \end{aligned} \tag{5.7}$$

where v and $\boldsymbol{k}$ are parts of the covariance matrix of the joint distribution

$$p(s_1, \ldots, s_{T+D}) = (\mathbf{0}, \begin{pmatrix} K & \boldsymbol{k} \\ \boldsymbol{k}^T & v \end{pmatrix}).$$

Another way to think about GPs is to view them as distributions over functions. A standard GP is fully defined by the covariance function $cov(\cdot, \cdot)$, its corresponding parameters $\boldsymbol{\theta}$ called hyperparameters and a noise level parameter σ_n^2. The most common covariance function and the one we choose as well is the squared exponential (SE) function:

$$cov_{se}(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{(x_p - x_q)^2}{2l^2}\right) + \sigma_n^2 \delta_{pq}$$

with σ_f^2 and l^2 being the hyperparameters. Parameter σ_f^2, the amplitude, allows for bumpier functions. The characteristic lengthscale l^2 determines the influence

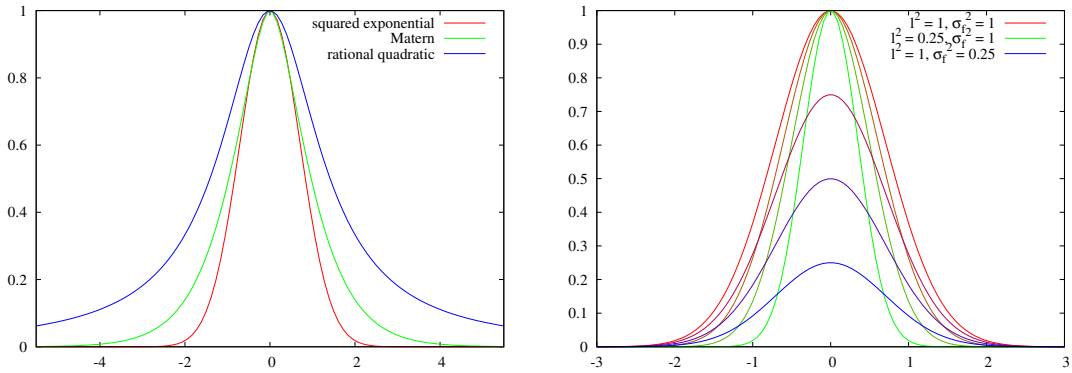

Figure 5.7: Different covariance functions and effects of hyperparameters in Gaussian Process regression. **Left**: Three commonly applied covariance functions. We chose the squared exponential function, plotted in red. **Right**: The squared exponential function is parameterized by two hyperparameters: the amplitude σ_f^2 and l^2 called lengthscale. The effect of both is shown in the plot. Reducing σ_f^2 is indicated by the color transition from red to blue, whereas curves whose colors change from red to green represent a diminishing l^2.

of neighboring training samples as a function of their distance, as depicted in Figure 5.7. A smaller lengthscale leads to a more peaked SE covariance function and accordingly, a less smooth regression. In case of multidimensional data, optimal lengthscales for each dimension can be learned, revealing the relative importance of every dimension (known as automatic relevance detection).

An often tackled issue about GPs is their cubic time complexity, which results from the matrix inversion needed to compute a prediction (see Eq. 5.7). To reduce this complexity in the number of training samples, various approximations exist, such as clustering the data in advance or using inducing inputs.

Optimizing Hyperparameters $\boldsymbol{\theta}$

To learn a GP means finding the optimal hyperparameters of the chosen covariance function, which is a squared exponential in our case. The optimal hyperparameters $\boldsymbol{\theta}^* = \left(\sigma_f^{2^*}, l^{2^*}, \sigma_n^{2^*}\right)$ maximize the likelihood of the training data:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \left(\log p\left(t_1, \ldots, t_n | x_1, \ldots, x_n, \boldsymbol{\theta}\right)\right) \\ &= \arg\max_{\boldsymbol{\theta}} \left(\underbrace{-\frac{1}{2}\boldsymbol{t}^T (K + \sigma_n^2)^{-1}\boldsymbol{t}}_{\text{data fit}} \underbrace{-\frac{1}{2}\log|K + \sigma_n^2 I|}_{\text{complexity penalty}} - \frac{n}{2}\log 2\pi \right). \end{aligned} \tag{5.8}$$

The data fit term favors the best fit to the data, resulting in an overfitting. By contrast, the complexity penalty prefers the simplest model explaining the training data, leading to an underfitting. Together, they balance the trade-off between model complexity and model fit.

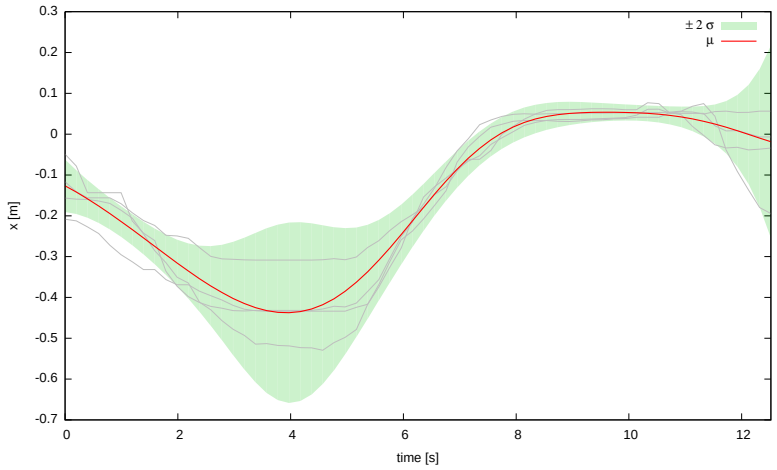

Figure 5.8: We regressed the same data (gray functions) used in the kernel regression example in Figure 5.6 with heteroscedastic Gaussian Processes. The resulting mean and variance function (indicated by the 95% confidence interval) are shown.

As there is no analytical solution to Eq. 5.8, we solve it by minimizing the negative log-likelihood. Thus, we can apply any optimization algorithm, e.g., gradient descent. The problem of local minima is dealt with by using different initial values for $\boldsymbol{\theta}$. However, as the hyperparameter space is often enough good-natured this is only a minor challenge.

Heteroscedasticity

One property of the standard GP model is the assumption of constant noise throughout the domain. As the key element of our approach is the variance of the individual constraints, we heavily rely on heteroscedasticity, i.e., varying levels of noise. There are various approaches addressing the problem of input-dependent noise rates. In [34] a secondary GP is used to model the noise variance, while a third combined GP is estimated approximating the most likely noise. Our data is even simpler to treat as it is equidistant over the whole time domain. Thus, we apply binning and use a hyper GP to learn the empirical variances. The result is illustrated in Figure 5.8.

Equivalence Kernel

If we want to gain a deeper insight into GP regression we have to consider the so-called equivalence kernel (see e.g., [58]). Viewing GP regression this way allows us to directly compare it to the kernel regression technique applied in the previous section. The equivalence kernel (EK) follows the idea that a GP prediction is a linear combination of N distinct kernel functions, each one centered at a training sample. We will soon comprehend that nearly all of the N kernels in our application

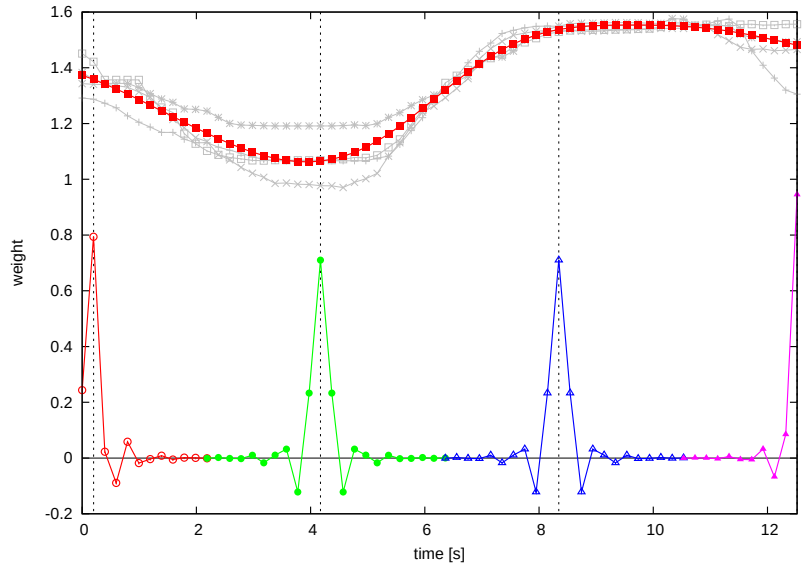

Figure 5.9: To gain a deeper insight in the relations between locally weighted regression and Gaussian Process regression we computed the equivalence kernel of the sample data, which is displayed in the upper part of the plot (including the estimated mean of the GP regression in red). We evaluated the equivalence kernel at four exemplary time steps. Due to illustration reasons they are cutted after converging to zero. Except for minor changes at the boundaries the equivalence kernels are virtually the same because of the equidistant samples provided by the data (gray). This weakens the major advantages of the computationally expensive GP regression compared to kernel regression.

are identical, loosing an important strength of GP regression.

The equivalence kernel EK at input $\hat{x}$ is defined as

$$EK\left(\hat{x}\right) = K_{\boldsymbol{\theta}}\left(x, \hat{x}\right) K_{\boldsymbol{\theta}}^{-1}\left(x, x\right),$$

with K being the covariance matrix. Obviously, the EK only depends on the hyperparameters and the distribution of the training data in the input dimension. We are equipped with equidistant uniform distributed relation samples $l\left(t, d, i\right)$ over time. This, and the fact, that were using a constant lengthscale parameter makes all EKs equal except for the boundaries which can be seen in Figure 5.3.2. In relation to kernel regression, the characteristic lengthscale l^2 has the same effects as the kernel bandwidth h.

After comparing regression using kernel estimation and Gaussian Processes, we decided to apply the kernel regression. Using the Heteroscedastic GPs comes with an additional computational overload that is not justified in our case. As we only regress equidistant data (as a result of the Kalman filtering), the advantages of GP regression, such as extrapolation and dealing with sparse data, vanish. Moreover,

kernel regression demands fewer parameters and outplay GPs in terms of computational complexity.

5.4 Reproduction

Once we have learned constraints in task and joint space we can apply them in our DBN to infer optimal imitative actions during reproduction. Following, we will show two possibilities to find those joint configurations. The first one is an incremental greedy technique, which may fail in case of additional ill-natured online constraints, such as U-shaped obstacles. To overcome this problem we show a second approach to plan a global optimal action sequence.

5.4.1 Incremental Optimization

If we consider only one time step during the optimization, we seek for the configuration q^* that maximizes the joint probability

$$q^* = \arg\max_q p(q, x, \hat{r}, r, z) \tag{5.9}$$

$$= \arg\max_q p(q \mid \hat{r}) p(x \mid r, q) p(z \mid x). \tag{5.10}$$

As discussed before in Section 5.1, the posteriors $p\left(q \mid \hat{c}\right)$, $p\left(x \mid c, q\right)$, and $p\left(z \mid x\right)$ are basically products of Gaussians and lead to a Gaussian distribution again. Thus, to maximize the joint probability, we need to determine the mean of that Gaussian.

We proceed as follows: Consider that we are currently at time step t and want to seek for the joint configuration that maximizes Eq. 5.10 at $t+1$. Each relation between x_i and x_E generates a relative displacement vector Δ_i w.r.t. the imitator's end-effector:

$$\Delta_i\left(t+1\right) = x_i\left(t\right) - x_E\left(t\right) + \mu_i\left(t+1\right),$$

where $x_i\left(t\right)$ and $x_E\left(t\right)$ are the positions of the two interaction elements related by constraint r_i at time step t in the scene.

We utilize the probabilistic models of constraints in task and joint space made up in section 5.3 to reproduce the skill in a new context. The new joint configuration $q(t+1)$ of the kinematic chain evolves from the previous one $q(t)$ by satisfying every constraint weighted according to its variance.

The configuration of the robot has to be specified in terms of joint states. Therefore, we have to convert the constraints expressed in world coordinates into joint space. This is done by a variant of the damped-least squares method which is described in 3.6.3. It is an approximative technique performing a linearization of the kinematic function. Because this transformation is assumed to be linear, each

corresponding change in joint configuration defined by

$$\Delta q_i(t+1) = J\left(JJ^T + \lambda^2 I\right)^{-1} \Delta x_i(t+1)$$

$$\Sigma_i(t+1) = \left(J\left(JJ^T + \lambda^2 I\right)^{-1}\right) \Sigma_i(t+1) \left(J\left(JJ^T + \lambda^2 I\right)^{-1}\right)^T$$

with λ being the damping factor forms also a Gaussian distribution in joint space. $\Sigma_i(t+1)$ is a 3×3 matrix encoding the variances from the relation r_i with the corresponding variances for dimension x, y, and z on the diagonal. Due to the linear mapping, we obtain also a Gaussian in joint space.

In contrast to task space constraints, the effect of the constraints in joint space $\tilde{r}$ is derived straightforward:

$$\Delta q_{N+1}(t+1) = \tilde{r}(t+1) - q(t)$$

$$\Sigma_{N+1}(t+1) = \Sigma_{\tilde{r}}(t+1),$$

where $q(t)$ is the current joint configuration. A fundamental assumption made here is that there exists a mapping between the demonstrator's and imitator's joint configuration.

All constraints resulting from the observation of the demonstrator's joint angle configurations or from the arrangement of objects in the scene are expressed in terms of updates in joint space. This allows us to combine them by multiplying the normal distributions as it has also been done by Calinon and Billard [10]. The resulting distribution is the product over the $N+1$ Gaussians resulting from the N task space constraints plus the joint space constraints. We can use it to directly obtain the next configuration $q^*(t+1)$ according to Eq. 5.10 by selecting the mean from this combined Gaussian, as given by

$$q(t+1) = q(t+1) + \left(\sum_{i=1}^{N} \Sigma_i(t+1)\Delta q\right) \left(\sum_{i=1}^{N+1} \Sigma_i(t+1)^{-1}\right)^{-1}$$

$$\Sigma(t+1) = \left(\sum_{i=1}^{N+1} \Sigma_i(t+1)^{-1}\right)^{-1}.$$

The distribution's mean defines the final configuration of the robot in the next time step that maximizes the probability distribution specified in Eq. 5.10. An iterative application of this scheme leads to an adaptive imitative behavior.

Special importance is accorded to the joint constraints $\tilde{r}$. If those are ignored one obtains a model-free representation of the skill. This in turn avoids the only poorly solved body correspondence problem.

5.4.2 Local Optimization with Additional Unobserved Constraints

The technique described in the previous section can directly be applied to deal with unforeseen obstacles in the scene. Consider that the robot observes an obstacle during the imitation that was not there during the demonstrations. To avoid this obstacle during the reproduction task, we can add additional task constraints between the observed obstacle and points on the robot's body.

Without changing the framework described above, the robot can reactively introduce constraints for avoiding obstacles while carrying out its task as close a possible to the human demonstrator. Let x_O be the position of the object. We then get for the desired movement

$$\begin{aligned}\Delta_O &= -\frac{x_E - x_O}{\|x_E - x_O\|} \\ \Sigma_O &= \exp\left(\alpha \|x_E - x_O\|^2\right) \cdot I,\end{aligned}$$

where α is a scaling factor.

It should be noted that his techniques works well for small or rather simple structured obstacles added to the scene. In case complex or, for example, U-shaped obstacles are found in the environment, this approach is likely to suffer from local minima caused by contradictory constraints.

5.4.3 Global Optimization by Planning

The problem of local minima, however, can be avoided by not incrementally optimizing the joint probability distribution of the DBN for the upcoming time step but optimizing it over all time steps $1 \dots T$ of the task sequence at once:

$$q^*_{1:T} = \underset{q_{1:T}}{\operatorname{argmax}}\, p\left(q_{1:T}, x_{1:T}, \tilde{r}_{1:T}, r_{1:T}, z_{1:t}\right) \tag{5.11}$$

Note that at a particular time step t, only the first $1 \dots t$ observations $z_{1:t}$ are already available and can be included for planning. Doing this optimization on a global level, however, comes with significantly increased computational cost.

One way of rather efficiently estimating $q^*_{1:T}$ is to make use of the well-known A^* algorithm since we are only interested in the most-likely configurations. Given that we properly encode the constraints in the cost function used by A^*, the reported solution is equivalent to Eq. 5.11.

Using a cost function that applies the Mahalanobis distance, incorporating $\mu_i(t)$ and $\sigma_i(t)$ for all constraints r_i, $\tilde{r}$, and the obstacle constrains in a time-dependent way, will lead to a command sequence that guides the robot on the Maximum-Likelihood configuration over time. Since our covariance matrices $\Sigma_i(t)$ are diagonal the Mahalanobis distance reduces to a distance measure called the normalized

Euclidean distance. Thus, for a configuration q at time t, we define

$$cost\,(t,q) = \sum_{i}^{n} \frac{||f\,(q) - \mu_i\,(t)\,||^2}{\sigma_i^2\,(t)} + \frac{||q - \mu_{\tilde{r}}\,(t)\,||^2}{\sigma_{\tilde{r}}^2\,(t)},$$

where n is the number of task constraints and f the forward kinematics function defined in Section 3.5. Then, finding the cost-optimal configurations is equivalent to maximizing the log likelihood of the combined Gaussian and, thus, is the Maximum-Likelihood solution for the DBN.

We have two possible options considering the space we are seaching. If we search the configuration space, as insisted above, we have to transform each joint configuration by means of forward kinematics. In contrast, we could also search the lower-dimensional workspace. As a drawback, each pose in workspace has to be transformed to a joint configuration using inverse kinematics, to assure its validity.

Another problem that increases the dimensionality of the search space regards the available actions that must be considered when expanding a node in the A^* search graph. Taking into account the varying velocities of the manipulator during a motion, these actions amount to a large set.

The runtime of the A^* algorithm depends mainly on the heuristic in use. We select the admissible heuristic h estimating the lowest possible remaining costs to the goal when starting from q at time t':

$$h\,(t',q) = \sum_{t=t'}^{T} cost\left(t, \mu_{\prod_i \mathcal{N}\left(\mu_i(t),\sigma_i^2(t)\right)}\right),$$

where $\mu_{\prod_i \mathcal{N}\left(\mu_i(t),\sigma_i^2(t)\right)}$ is the mean of the product of all Gaussians given by the constraints at time step t.

6 Experiments

We carried out a set of experiments to analyze our approach. A quantitative evaluation of imitation attempts without using a simulator is virtually impossible due to the absence of a ground truth when solving a task by imitation. We execute optimal actions w.r.t. to our imitation metric. Thus, finding an appropriate measure to capture the difference between performed and intended actions would resolve the imitation problem itself. Therefore, we evaluate our method on a qualitative basis, from a human perspective. To imitate the observed behavior, we reproduced the tasks using our real robot Zora equipped with a manipulator and two simulated robots, one with a 6-DoF manipulator and another with a humanoid morphology.

6.1 Pick-and-Place Task

In a first experiment, a pick-and-place skill was learned by imitation. This task is composed of three essential interaction artifacts: the hand, the starting position of an object to be moved, and its destination. The hand approaches the object, grasps it, moves it to the desired target position and retreats.

6.1.1 Learning Phase

A human teacher demonstrated the task repeatedly, a total of eight times (see Figure 6.6 for an example), with the left arm moving. The demonstrations were recorded using a single camera and data was processed as described in Chapter 4. The ARToolKit extracted marker poses at an average rate of 5 frames per second. Each single demonstration varied in the initial position of the object to be picked. As we will see later, this is not a necessary condition for a successful imitation. The duration of a single demonstrated pick-and-place action ranged from 10 to 14 seconds.

The two workspace relations for this task are the relative positions between end-effector and initial position of the mug as well as the relative position between end-effector and the target. Learning these constraints, shown in Figure 6.2, reveals two important aspects of the pick-and-place skill, indicated by a small variance. First, it is important that the end-effector (the hand) is close to the mug (i.e., inside the grasping zone) at the beginning of the task. Second, near the end of the task execution it is crucial for the end-effector to be close to the placing position. The height of the learned transportation trajectory (z-component in Figure 6.2)

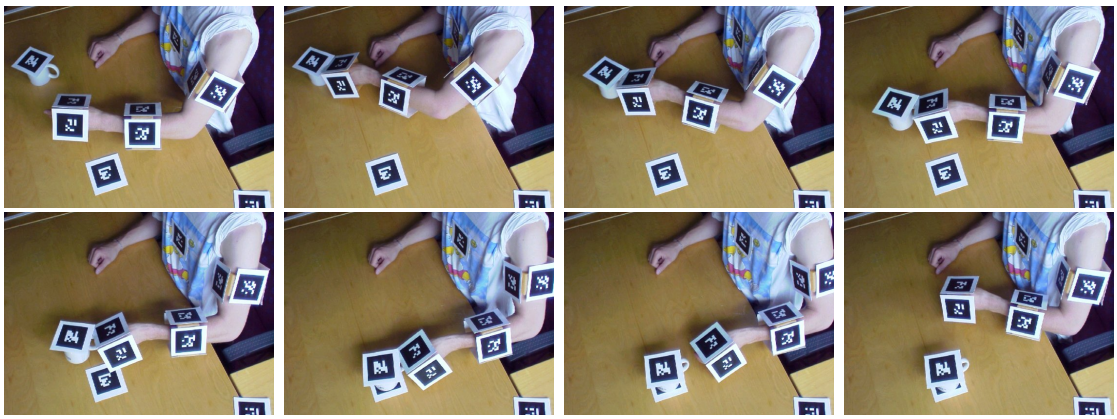

Figure 6.1: A sample demonstration of the pick-and-place task by a human teacher. The mug is placed onto a marker representing the target position (from left to right; top to bottom).

is constraint throughout the whole action as there was no variation during the demonstrations.

Apart from the workspace constraints, six constraints on the observed joint angles of the human-arm-like chain are learned, shown in Figure 6.3. In these plots, two things stand out. First, the joint configurations of shoulder and elbow are more constraint in the second part (between seconds 10 and 14) than before. This is due to the fact, that the starting position of the mug changed while its target always stayed the same. Second, the mug was always moved from right to left which results in a relatively small variance during transportation (between seconds 5 and 10). Here, the direction of the pick-and-place operation is encoded in the constraints.

6.1.2 Reproduction Phase

The reproduction of the pick-and-place motion is first tested using a simulation environment. In this way, the inexact pose estimations of the markers can be bypassed, easing the problem of imitation. Furthermore, in a simulation the imitator can easily take on a wide range of different constitutions.

Imitation using Humanoid Model

We simulated a humanoid robot to imitate the pick-and-place task. The anthropomorphic arm had the same structure as the HAL chain described in Section 3.3.1. Therefore, we could use the learned joint constraints by applying an identity mapping between human demonstrator and robot imitator. Figure 6.4 shows an exemplary imitation result. Note, that considering the joint constraints leads to a fluent, human-like movement. This should not be taken for granted as in this case we are dealing with a redundant manipulator. Its six DoFs versus the three DoFs implied by the taskspace (orientation is ignored) allows for more than one solution

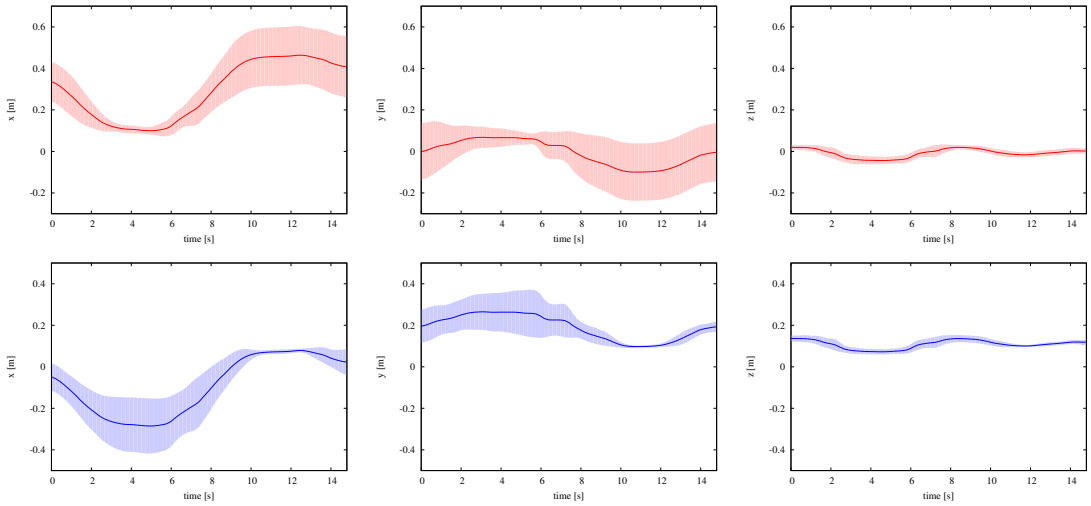

Figure 6.2: The two learned workspace constraints for the pick-and-place task. **Top**: The mean distance and variance between end-effector and picking position of the mug in each dimension. The relation gets highly constraint in the first part of the task (around 4 seconds) when the hand grasps the mug. **Bottom**: The mean distance and variance between end-effector and placing position of the mug in each dimension. During the last part of the task (around 12 seconds) the relation gets highly constraint when the hand reaches the destination.

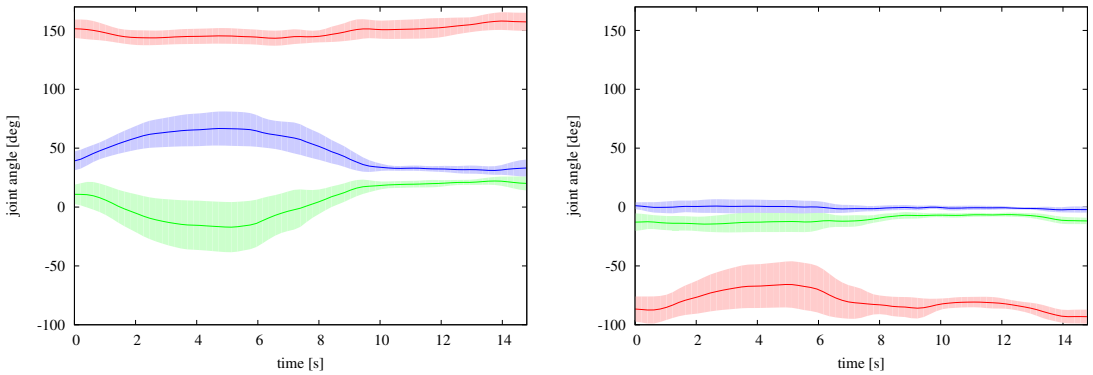

Figure 6.3: The two learned configuration space constraints for the pick-and-place task. **Left**: The mean angle and variance of the 3-DoF shoulder joint in each dimension. **Right**: The mean angle and variance of the 3-DoF elbow joint in each dimension. All joint angles are highly constraint when reaching the target position (around 12 seconds) due to the fact that the destination never changed during demonstrations.

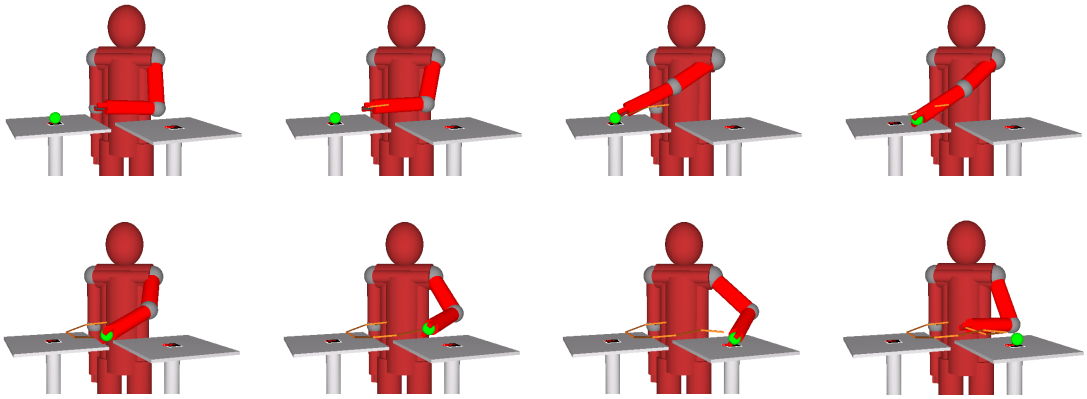

Figure 6.4: A simulated sample reproduction of the pick-and-place task by a 6-DoF HAL chain using task and joint space constraints. The manipulator approaches the green sphere and moves it to the target indicated by a second marker (from left to right; top to bottom). The brown trajectory visualizes the position of the end-effector over time. The motion is initialized with the mean joint configuration.

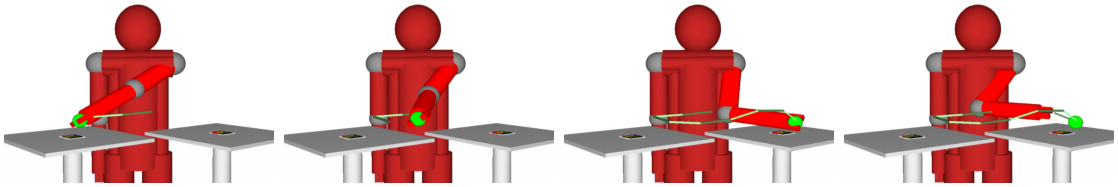

Figure 6.5: The same reproduction of the pick-and-place task as in Figure 6.4 but without considering constraints in joint space. This results in a rather non-humanoid motion imitation.

of the inverse kinematics problem. Thus, ignoring the joint constraints may lead to a little intuitive imitation as can be seen in Figure 6.5.

Imitation using the Manipulator of the Robot Zora

The same skill was reproduced by a simulated version of our 6-DoF robotic manipulator Zora, as presented in Section 3.3.2. As the demonstrator and the imitator have a significantly different body scheme, the joint constraints had been disabled. As can be seen in Figure 6.1, the robot is able to reproduce the task even though the setup between demonstration and imitation has been changed by setting a different target location. Thus, the system was able to generalize over the pick and placing positions of the object.

The imitation was also successfully performed by a real robot, as depicted in Figure 6.8. A reference marker was attached at the base of the robot to calculate the position of the end-effector via forward kinematics and an model of the robot. We repeated these experiments multiple times and never observed failures during

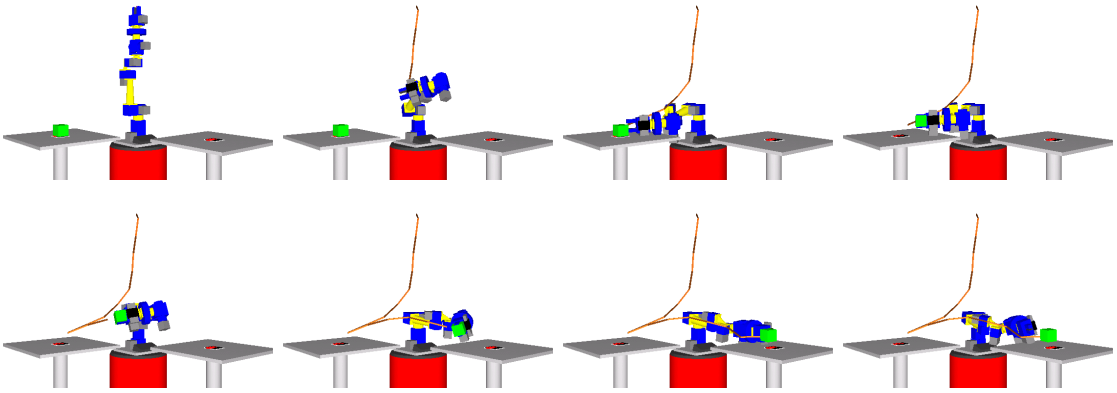

Figure 6.6: A simulated sample reproduction of the pick-and-place task by a 6-DoF kinematic chain. Starting from the zero configuration, the green box is approached and moved to the target indicated by a second marker (from left to right; top to bottom). The brown trajectory visualizes the position of the end-effector over time.

the imitation, even when changing the position of the object and target.

Collision Avoidance during Imitation

We introduced additional obstacle constraints described in Section 5.4.2 which allow the robot to deal with unforeseen obstacles during task execution. The obstacle constraints act similar to a potential field pushing the closest point of the robot to the obstacle away from it. This often results in the characteristic oscillative evasive movement. In Figure 6.7 the obstacle to avoid is a column modelled by four spheres whose centers are indicated by markers. The pick-and-place experiment including obstacles was also conducted on the real robot and is shown in Figure 6.8.

Figure 6.9 shows a comparison of the constraint satisfaction during reproduction between the presence of obstacles and when ignoring them. The evasive movements take place when approaching the picking position from the upright starting configuration (at around 1 second) and a second time when circumnavigating the column on the way to the target marker (at around 6 seconds). As can be seen by the blue end-effector trajectory, the first collision avoidance occurs mainly in the x-dimension while the second time the y- and z-dimension are used. The typical potential field-like oscillations near obstacles are softened when the actions are executed by the controller on a lower level.

Imitation by Planning

The experiments presented above used the reproduction by means of incrementally computing the maximum-likelihood configuration of the DBN. This is can be done efficiently online, but the approach can suffer from local minima, for example, in the presence of U-shaped obstacles. Such an example is presented in Figure 6.10

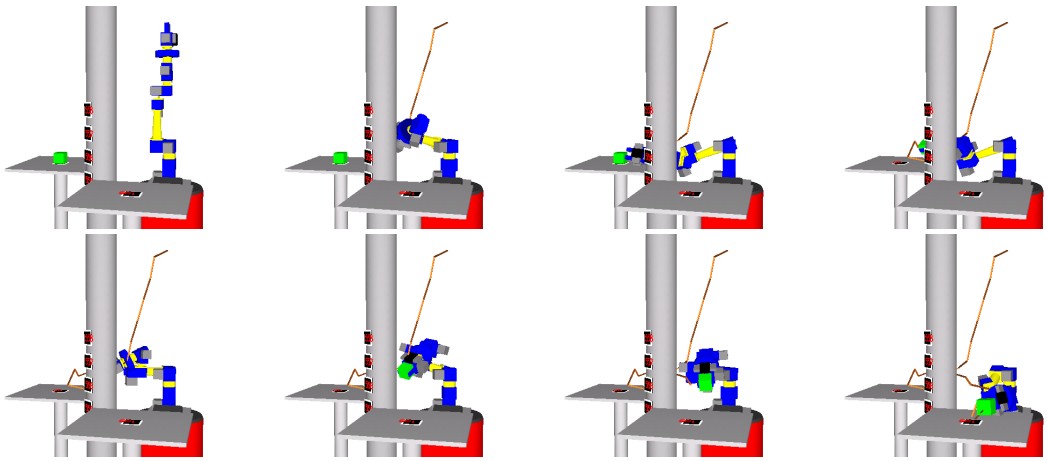

Figure 6.7: A simulated sample reproduction of the pick-and-place task by a 6-DoF kinematic chain with a column-shaped obstacle. Starting from the zero configuration, the manipulator approaches the green box and moves it to the target indicated by a second marker, avoiding the marker-labeled column in-between (from left to right; top to bottom). The brown trajectory visualizes the position of the end-effector over time.

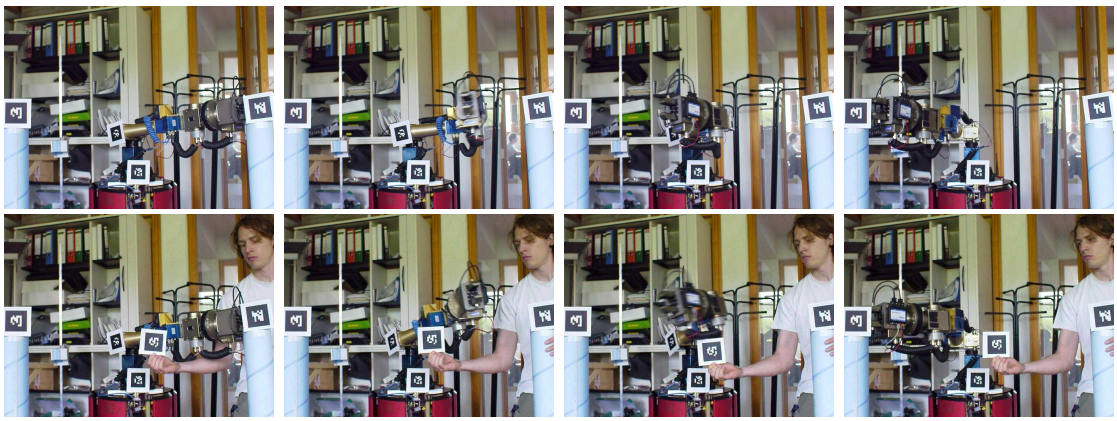

Figure 6.8: Difference of the pick-and-place movement when adding obstacle constraints. The photo sequence in the upper row shows the real robot moving from start to target position (indicated by two markers). Later, an obstacle is introduced in the scene (additional marker in the lower row). The robot observes it an adapts its behavior by circumnavigating the obstacle while still accomplishing the task.

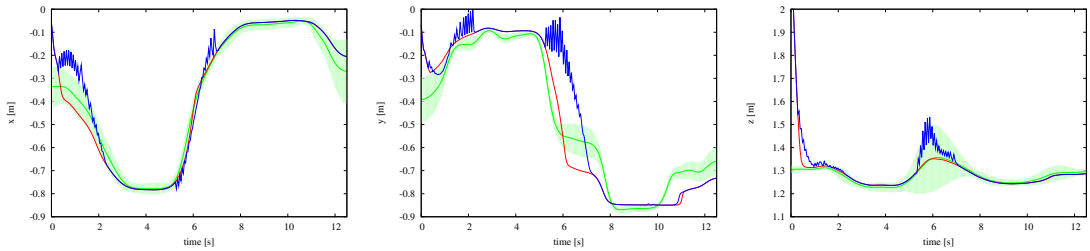

Figure 6.9: Constraint satisfaction when dealing with obstacles in the pick-and-place task. The green function shows mean and variance of all constraints in task space combined except for the obstacle constraint, i.e., the optimal end-effector trajectory during reproduction. Omitting collision avoidance results in the red end-effector trajectory, while the blue one avoids obstacles.

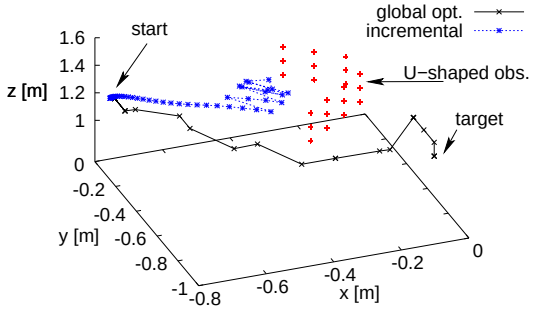

Figure 6.10: The plot shows the end-effector position of the robot over time during two experiments. When applying the incremental method, the end-effector gets stuck in a U-shaped obstacle while the global method solves the task and the end-effector reaches the target location.

where the robot gets stuck.

If one applies the global optimization technique described in Section 5.4.3, one can overcome this problem since the optimal solution over all time steps is computed. Thus, the robot is able to reproduce the task including the avoidance of the U-shaped obstacle. This global method, however, comes with a significantly increased computational load.

6.2 Pouring Task

A second experiment was conducted to show how different dimensions (position and rotation) captured by the task space constraints can be applied efficiently. The task consisted of a pouring motion, i.e., grabbing a bottle, emptying it over a glass, and putting it back.

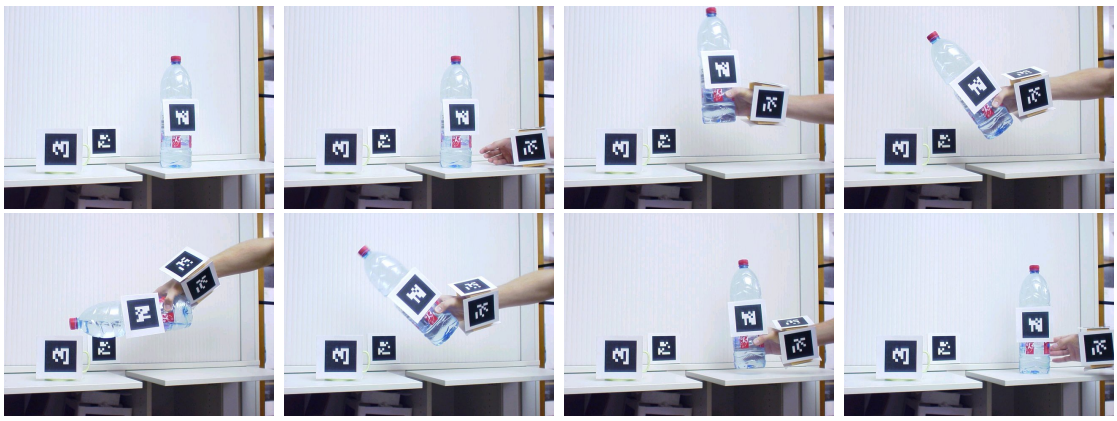

Figure 6.11: A sample demonstration of the pouring task by a human teacher (from left to right; top to bottom).

6.2.1 Learning Phase

A human demonstrated the action a total of four times (see Figure 6.11). During demonstration the initial position of the bottle was varied as well as the position of the glass. This time we learned not only the positional relations between the end-effector, bottle, and glass but also the rotational relations between them. Figure 6.12 shows the resulting constraints between end-effector and glass. It can be seen easily, that the orientation of the end-effector (and consequently the bottle) is constraint throughout the whole task execution. At the beginning and end it is hold upright, while the pouring action itself demands a horizontal orientation. Variance only increases when the bottle was rotated, as there are small temporal differences between demonstrations that the time warping cannot account for. The relative position of end effector and glass is only constraint during pouring because the initial pose of the bottle was varied. Interestingly, the relation between the starting pose of the bottle and the end-effector showed that the bottle was returned to its original position after the pouring took place. A fact, of which even the instructor was not explicitly aware of during demonstration.

6.2.2 Reproduction Phase

We reproduced this task only in simulation using a model of our 6-DoF manipulator. The robot was able to generalize the pouring behavior over arbitrary glass and bottle positions. Note, that the robot will always pour from the right-hand side (from its perspective), as it was presented in the demonstrations. Figure 6.13 depicts a reproduction run in the simulation testbed. To satisfy the relative orientations the inverse kinematic method was extended to solve the problem of finding an joint configuration for a given six dimensional end-effector pose.

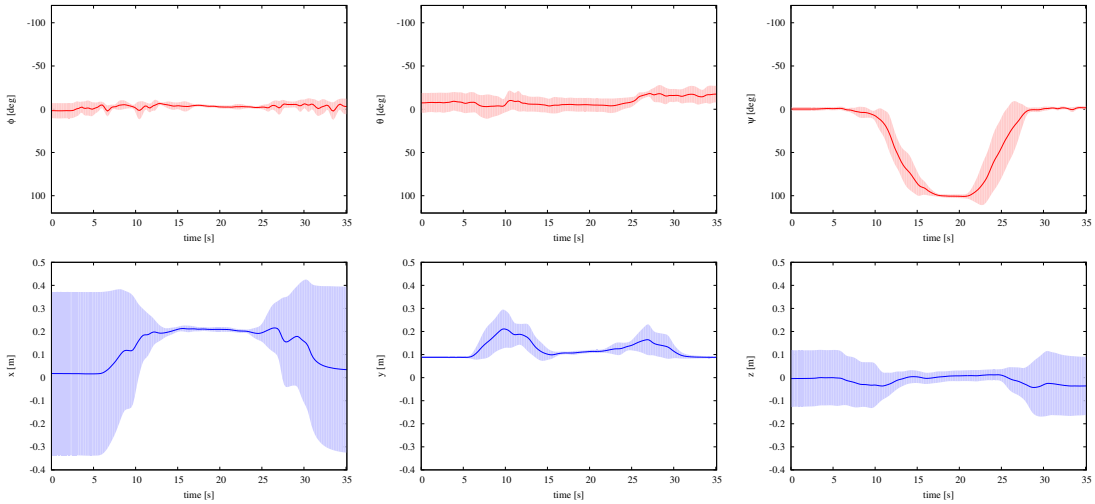

Figure 6.12: Learned constraints for the pouring task in six dimensional task space. **Top**: The relative mean rotation and variance between end-effector and glass in each dimension. It shows the constant upright orientation of the bottle and the change to a horizontal orientation when filling the glass. **Bottom**: The mean distance and variance between end-effector and position of the glass in each dimension. The pouring takes place in the strong constrained middle part (at around 20 seconds).

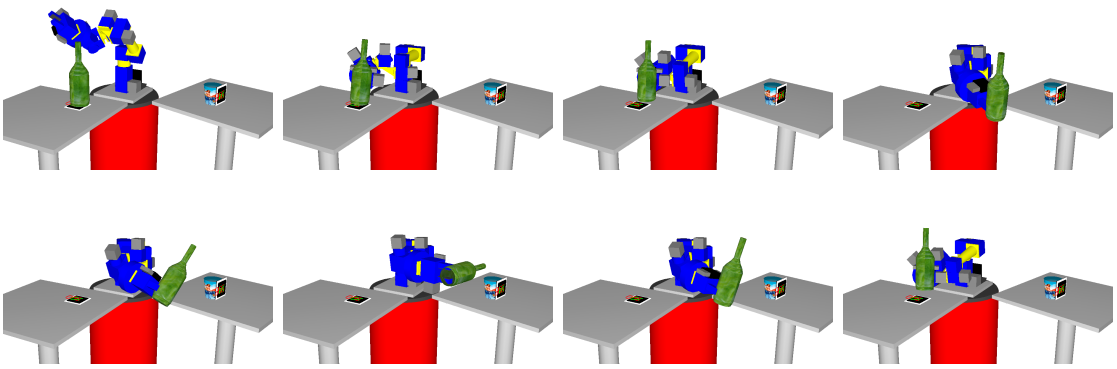

Figure 6.13: A simulated sample reproduction of the pouring task by a 6-DoF kinematic chain (from left to right; top to bottom).

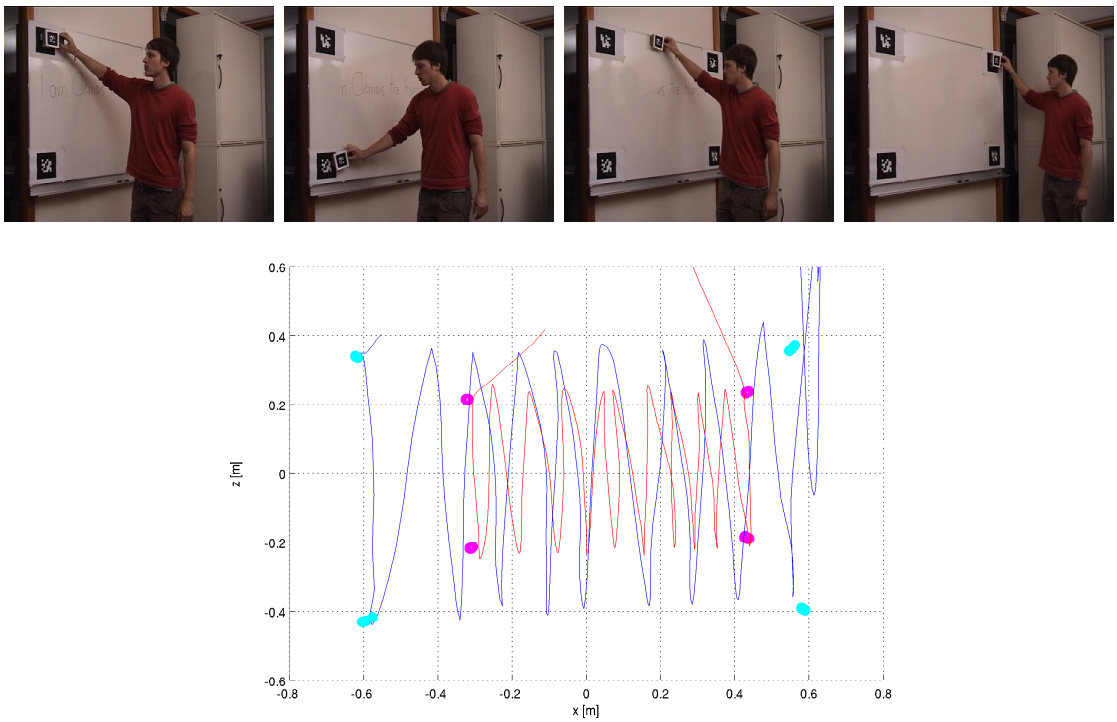

Figure 6.14: The upper photo sequence shows a sample demonstration of the whiteboard cleaning task performed by a human teacher. In total, only two demonstrations were required to generalize over the size of the whiteboard area to be cleaned and its position. The marker observations of the these two demonstrations are displayed in the lower plot, representing the x-z-plane of the 3D space. While one demonstration was conducted on a large whiteboard (cyan dots mark their corners; the blue trajectory shows the sponge position over time, starting in the upper left corner), another one was executed on a smaller area (magenta corners; red sponge trajectory).

6.3 Whiteboard Cleaning Task

In a third experiment, the robot learned to clean a whiteboard by imitation. We intended the robotic manipulator to be able to wipe whiteboards differing in size and position.

6.3.1 Learning Phase

A human instructor wiped the whiteboard using a marked sponge. The area to be cleaned was bounded by markers indicating the four corners. A total of two demonstrations were made, varying in the size of the cleaned area (see Figure 6.14). Relations between each of the four corners and the sponge/end-effector were learned.

The observed end-effector trajectories including marker positions of the whiteboard can be viewed in Figure 6.14. It is important that the shape of the wiping motion used in both demonstrations comparatively match each other including

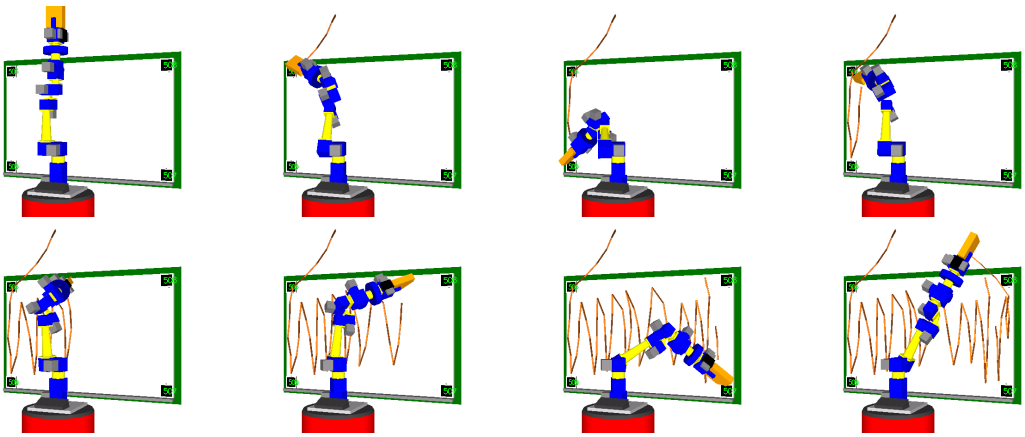

Figure 6.15: A simulated sample reproduction of the whiteboard-cleaning task by a 6-DoF kinematic chain using task space constraints between the whiteboard corners and the end-effector (sponge). The manipulator cleans the board in a zig-zag fashion as demonstrated by the teacher (from left to right; top to bottom). Additionally, it generalizes over the size of the board, limited by the four corner markers. The brown trajectory visualizes the position of the end-effector over time.

Figure 6.16: The reproduction of the board cleaning task by our 6-DoF manipulator. Starting at the upper left corner indicated by one of the four markers, the robot arm performs a zig-zag movement cleaning the whiteboard with a sponge grasped between its two fingers (from left to right). Note that the learned task representation allows for cleaning arbitrary sized surfaces.

starting and end position of the sponge at the upper left and right board marker respectively. This shortcoming of our approach is caused by the fact that it generalizes at the trajectory level and not in terms of higher-level effects (e.g., a clean whiteboard).

6.3.2 Reproduction Phase

Figure 6.15 shows the reproduction of the cleaning skill by the simulated 6-DoF manipulator. A sponge was attached to the real robot to accomplish the same task as can be seen in the photo sequence of Figure 6.16. We modified the size of the area to clean for illustrating the capabilities of generalization.

Figure 6.17: The board cleaning task with an obstacle reproduced by our 6-DoF manipulator. In the first frame the position of the obstacle is shown to the system via a marker. Next, the manipulator cleans the whiteboard as shown in figure 6.16, additionally trying to avoid the obstacle (from left to right; top to bottom). The evasive movements are performed primarily along the normal of the whiteboard surface. As can be seen in the last frame, part of the text located underneath the obstacle marker was not wiped.

Collision Avoidance during Imitation

As a further challenge we presented an obstacle marker to the robot during the reproduction phase (see first image of Figure 6.17), which was not existent when learning the skill. Consequently, the manipulator had to clean the whiteboard while avoiding obstacles and thus not wiping the area covered by the marker. For reasons of illustrations, we removed the marker during the experiment but kept it in the internal memory of the robot. In this way, one can see that the robot did not clean the corresponding area. A sequence of eight photos is depicted in Figure 6.17. To avoid the region marked as an obstacle zone, the manipulator lifts the sponge away from the whiteboard (in the direction of the observing camera).

6.4 Comparison of IK Techniques

In the reproduction phase of our approach the inverse kinematics problem has to be solved. We chose a suitable inverse kinematic (IK) method among the three approaches explained in detail in Section 3.6: the Jacobian transpose (JT), the pseudoinverse (PI), and the damped least squares (DLS) method. As all approaches work iteratively, they were evaluated regarding convergence rate and robustness. A simulated 6-DoF serial manipulator (see Section 3.3.1) with no joint limits was required to position its end-effector at arbitrary targets.

It is shown that there exists a trade-off between the two desired features: Faster convergence to a solution nearly always comes with less robustness when facing singular configurations. However, the DLS algorithm proves to be the most efficient method that fits our needs.

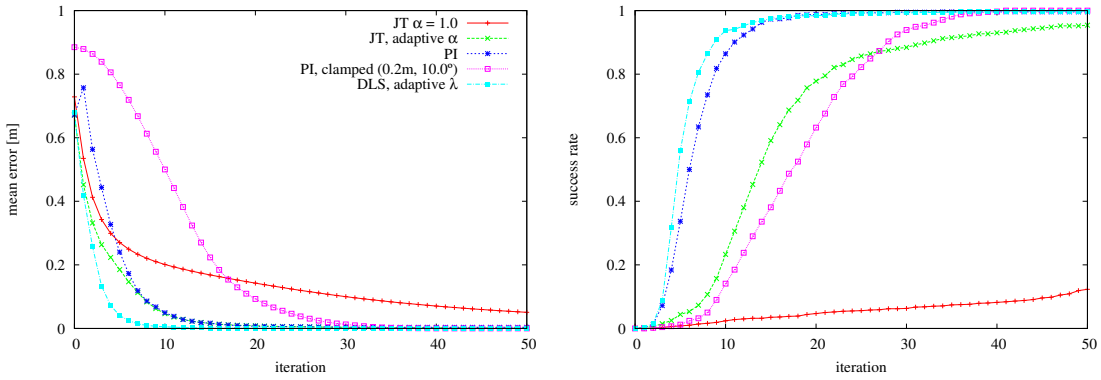

Figure 6.18: Comparison of IK techniques in reachable workspace. 10000 target positions were sampled uniformly in the reachable workspace of a 6-DoF kinematic chain. **Left**: The mean Euclidean distance over all runs is plotted against the first 50 iteration steps of the methods. The fastest convergence is shown by the damped least squares (DLS) algorithm. **Right**: If the distance between end-effector and target is equal to or lower than 1 *cm* a successful IK execution is registered. The plot shows the mean number of successes depending on the iteration step. Variances did not differ significantly and are omitted due to illustrative reasons.

Reachable Workspace

In a first trial 10000 positions were sampled in reachable workspace each of which had to be approached by the kinematic chain starting from the zero joint configuration. Target orientation was not considered. After a maximum of 200 iterations the method executions were stopped. As performance measure the mean Euclidean distance between end-effector and target was calculated for each iteration step.

Figure 6.18 shows the comparative results of the first 50 iterations. JT with a static parameter value $\alpha = 1.0$ reflects the slowest convergence rate, while applying an adaptive α shows comparably good results keeping in mind the simplicity of the technique. The vanilla PI performs quite good as well, but already reveals some cues to its oscillating behavior when dealing with singularities (notice the increase in the error during the first few iterations) which will become worse in the next section. Therefore, it is inevitable to clamp the length of the target directed vector and the maximum joint angle change per iteration. However, these sanctions result in slower convergence. Finally, the DLS using an adaptive parameter value λ outperforms all other approaches.

Unreachable Workspace

A second trial was conducted, trying to evaluate the stability of the techniques in singular configurations. In case of our simulated manipulator this occurs when the chain is fully extended. Hence, we sampled 10000 target positions outside of the reachable workspace; in particular points on the surface of a sphere with a 0.2 m

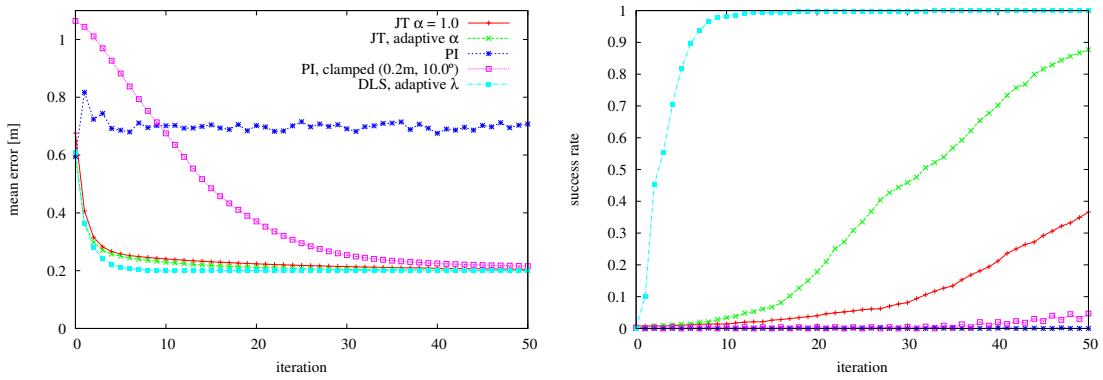

Figure 6.19: Comparison of IK techniques in unreachable workspace. To examine the performance of the different IK approaches when facing singular configurations, we sampled 10000 target positions outside the reachable workspace, i.e., exactly 20 *cm* away from the nearest possible end-effector position. **Left**: The mean Euclidean distance over all runs is plotted against the first 50 iteration steps. Again, the fastest convergence is shown by the damped least squares (DLS) algorithm. **Right**: A successful IK execution was defined when the error between end-effector and target was equal or lower than 21 *cm*. The plot shows the mean number of successes depending on the iteration step.

larger radius than the length of the fully outstretched kinematic chain. Again, the mean distance between end-effector and target was computed for each iteration. Due to the position of the sampled targets, the smallest possible error was always 0.2 *m*.

The results of the first 50 iteration steps in Figure 6.19 indicate that PI performs worst. While the vanilla version starts oscillating without ever reaching the target, the clamped version converges slowly but robustly. Both JT methods approach the targets reliably and are only beat by DLS, which is the fastest method to converge.

Following the overall results, we selected the DLS method to be applied during reproduction (see Section 5.4). To choose an optimal parameter λ we conducted separate experiments which featured the same trade-off between robustness and speed of convergence. Small λ-values allowed for a faster target approaching but simultaneously lead to intense oscillations near singular joint configurations. Consequently, we employed a technique that dynamically adapts λ as described in Section 3.6.3.

Summing up, our experiments showed the generalizing abilities of the imitation approach in a variety of basic movement tasks and with different robotic manipulators. Even when dealing with additional unlearned constraints, such as obstacles, the method showed pleasant results.

7 Summary

7.1 Conclusion

In this thesis, we presented an approach to imitation learning that extends and enriches a recently published method of Calinon and Billard. It enables a robot to observe, generalize, and reproduce tasks from watching a skilled human demonstrator. Imitation takes place at a trajectory level and benefits from multiple presentations of the same task.

We formalized the imitation learning problem applying a dynamic Bayesian network which is used for learning spatial relationships between observed positions of objects and body parts of the instructor. The learned relations provide an insight into the essential invariant features of the corresponding task. Apart from the constraints learned from demonstrations, additional constraints can be added online, for example, to avoid unforeseen obstacles during reproduction.

To imitate the action of a human, we estimate the actions that maximize the joint probability distribution represented by the dynamic Bayesian network. We presented two methods to find optimal imitative action sequences. First, a greedy incremental search that can be computed efficiently online but may suffer from local minima, especially in case of malicious obstacle structures. Second, a global optimization over the whole action sequence that comes at significantly increased computational load.

We evaluated the efficacy of the approach and showed that a real robot equipped with a manipulator can learn and reproduce demonstrated actions. Based on a pick-and-place, a pouring, and a whiteboard cleaning task, we illustrated the flexibility of the method to generalize over different spatial setups.

7.2 Future Work

One shortcoming of our approach are the lacking capabilities of the temporal alignment of multiple demonstrations. Although it can map nonlinearities, a restrictive condition is the continuity of temporal events. Imagine the task of laying a table. The order of setting the individual pieces is quite arbitrary, except that, e.g., the saucer will always be laid before the cup (assuming only single-arm movements). Another critical situation concerns the grasping of an object, which is highly dependent on approaching it first. To solve problems like these (there exist already approaches [47] which infer graph-based task representations), our approach could be extended by higher-level temporal constraints. Just like the spatial ones, they

can be defined relatively between elementary symbolic actions using mean and variance. Accordingly, it would be possible to interactively set the table with a robot through corporation.

Another interesting aspect tackles the recognition of already demonstrated actions. Is this possible only relying on basis of their generalized task description? And if not, what information is missing allowing a successful classification? Moreover, this could be used to quantitatively evaluate different approaches to learning by imitation.

Other challenges correspond to the much more fundamental question of when to imitate. Our approach primarily answers the questions of what and how to imitate, assuming that the demonstrated behavior is an optimal solution to the searched problem. This comprehends the imitator's capacity to infer the demonstrator's intentions. But maybe instead of cleaning the whiteboard, the person was just creating a specific zig-zag pattern with the sponge on the board. A loosely related idea to the understanding of the goal of a task is to incorporate not only "good" examples of movement (like we have done in our experiments) but as well to benefit from the demonstration of "bad" examples, e.g. in obstacle avoidance.

Finally, the up to now inefficient global optimization of imitative actions by planning (A^*-algorithm), described in Section 5.4.3, can be improved. Techniques such as rapidly-exploring random trees (RRTs, see [36]) or probabilistic roadmaps (see [31]) are not guaranteed to find optimal solutions but may perform in real time.

Bibliography

[1] Al-Zubi, S., and Sommer, G. Learning to imitate human movement to adapt to environmental changes. In *18th International Conference on Pattern Recognition, ICPR 2006, August 20-24, Hong-Kong* (2006), vol. 1, pp. 191–194.

[2] Alissandrakis, A., Nehaniv, C. L., and Dautenhahn, K. Imitation with ALICE: learning to imitate corresponding actions across dissimilar embodiments. *IEEE Transactions on Systems, Man, and Cybernetics, Part A 32*, 4 (2002), 482–496.

[3] Arras, K. O., and Cerqui, D. Do we want to share our lives and bodies with robots? a 2000-people survey. Technical Report EPFL-ASL-TR-0605-001, Autonomous Systems Lab, Swiss Federal Institute of Technology Lausanne, June 2005.

[4] Asfour, T., Gyarfas, F., Azad, P., and Dillmann, R. Imitation learning of dual-arm manipulation tasks in humanoid robots. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)* (2006).

[5] Bakker, P., and Kuniyoshi, Y. Robot see, robot do: An overview of robot imitation. In *In AISB96 Workshop on Learning in Robots and Animals* (1996), pp. 3–11.

[6] Bentivegna, D., Atkeson, C., and Cheng, G. Learning similar tasks from observation and practice. In *Proceedings of the International Conference on Intelligent Robots and Systems* (2006), pp. 2677–2683.

[7] Billard, A., Calinon, S., Dillmann, R., and Schaal, S. Robot programming by demonstration. In *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2008.

[8] Billard, A., and Mataric, M. Learning human arm movements by imitation: Evaluation of biologicallyinspired connectionist architecture. *Robotics and Autonomous Systems 941* (2001), 1–16.

[9] Calinon, S., and Billard, A. Active teaching in robot programming by demonstration. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* (August 2007), pp. 702–707.

[10] Calinon, S., and Billard, A. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)* (September 2008).

[11] Calinon, S., Guenter, F., and Billard, A. Goal-directed imitation in a humanoid robot. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* (2005).

[12] Chalodhorn, R., Grimes, D. B., and Rao, R. P. N. Learning to walk through imitation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (San Mateo, CA, 2007), Morgan Kaufmann.

[13] Cole, J. B., Grimes, D. B., and Rao, R. P. N. Learning full-body motions from monocular vision: Dynamic imitation in a humanoid robot. In *IEEE International Conference on Intelligent Robots and Systems (IROS'2007)* (San Francisco, CA, 2007), IEEE Press.

[14] Craig, J. J. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[15] Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A., and Turransky, A., Eds. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA, 1993.

[16] Denavit, J., and Hartenberg, R. S. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics 23* (1955), 215–221.

[17] Deo, A. S., and Walker, I. D. Overview of damped least-squares methods for inverse kinematics of robot manipulators. *Journal of Intelligent and Robotic Systems V14*, 1 (1995), 43–68.

[18] di Pellegrino, G., Fadiga, L., Fogassi, L., Gallese, V., and Rizzolatti, G. Understanding motor events: A neurophysiological study. *Experimental Brain Research 91* (1992), 176–180.

[19] Dufay, B., and Latombe, J.-C. An approach to automatic robot programming based on inductive learning. In *In Proceedings of the 1st International Symposium on Robotics Research* (1984), MIT Press, pp. 3–20.

[20] Fiala, M. ARTag, a fiducial marker system using digital techniques. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 590–596.

[21] Gams, A., and Lenarcic, J. Humanoid arm kinematic modeling and trajectory generation. *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob 2006)* (2006), 301–305.

[22] Grassia, F. S. Practical parameterization of rotations using the exponential map. *J. Graph. Tools 3*, 3 (1998), 29–48.

[23] Grimes, D., Chalodhorn, R., and Rao, R. Dynamic imitation in a humanoid robot through nonparametric probabilistic inference. In *Proceedings of Robotics: Science and Systems* (Philadelphia, USA, August 2006).

[24] Grimes, D. B., Rashid, D. R., and Rao, R. P. Learning nonparametric models for probabilistic imitation. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, Cambridge, MA, 2007, pp. 521–528.

[25] Guenter, F., Hersch, M., Calinon, S., and Billard, A. Reinforcement Learning for Imitating Constrained Reaching Movements. *RSJ Advanced Robotics, Special Issue on Imitative Robots 21*, 13 (2007), 1521–1544.

[26] Hafner, R., and Riedmiller, M. Neural reinforcement learning controllers for a real robot application. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* (2007), pp. 2098–2103.

[27] Hafner, V. V., and Kaplan, F. Interpersonal maps and the body correspondence problem. In *in Proceedings of the Third International Symposium on Imitation in animals and* (2005), pp. 48–53.

[28] Ijspeert, A., Nakanishi, J., and Schaal, S. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2002)* (2002), pp. 1398–1403. (received the ICRA2002 best paper award).

[29] Jones, M. C., Marron, J. S., and Sheather, S. J. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association 91*, 433 (1996), 401–407.

[30] Kato, H., and Billinghurst, M. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99)* (San Francisco, USA, Oct. 1999).

[31] Kavraki, L. E., claude Latombe, J., and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Methodsand Models in Automation and Robotics, 2005* (1997), MorganKaufmann, pp. 566–580.

[32] Kelmar, L., and Khosla, P. Automatic generation of kinematics for a reconfigurable modular manipulator system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '88)* (April 1988), vol. 2, pp. 663 – 668.

[33] Keogh, E. J., and Pazzani, M. J. Derivative dynamic time warping. In *In Proc. of the 1st SIAM Int. Conf. on Data Mining (SDM-2001)* (2001), pp. 5–7.

[34] Kersting, K., Plagemann, C., Pfaff, P., and Burgard, W. Most likely heteroscedastic gaussian process regression. In *ICML '07: Proceedings of the 24th international conference on Machine learning* (New York, NY, USA, 2007), pp. 393–400.

[35] Kulić, D., Takano, W., and Nakamura, Y. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *Int. J. Rob. Res. 27*, 7 (2008), 761–784.

[36] Lavalle, S. M. Rapidly-exploring random trees: A new tool for path planning. Tech. rep., 1998.

[37] Ligeois, A. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *Systems, Man and Cybernetics, IEEE Transactions on 7*, 12 (Dec. 1977), 868–871.

[38] Meltzoff, A. N., and Moore, M. K. Imitation of facial and manual gestures by human neonates. *Science*, 198 (October 1977), 75–78.

[39] Meltzoff, A. N., and Moore, M. K. Newborn infants imitate adult facial gestures. *Child Development*, 54 (1983), 702–709.

[40] Miller, N., Jenkins, O., Kallmann, M., and Mataric, M. Motion capture from inertial sensing for untethered humanoid teleoperation. *Humanoid Robots, 2004 4th IEEE/RAS International Conference on 2* (Nov. 2004), 547–565.

[41] Moeslund, T. B., and Granum, E. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding: CVIU 81*, 3 (2001), 231–268.

[42] Nadaraya, E. A. On estimating regression. *Theory of Probability and its Applications 9*, 1 (1964), 141–142.

[43] Nakamura, Y., and Hanafusa, H. Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASME J. Dynamic Systems, Measurement and Control 108* (1986), 163–171.

[44] Nehaniv, C. L., Ab, H. A., and Dautenhahn, K. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In *Interdisciplinary Approaches to Robot Learning* (2000), World Scientific Press, pp. 136–161.

[45] Nocedal, J., and Wright, S. J. *Numerical Optimization*. Springer, August 1999.

[46] Ogawara, K., Takamatsu, J., Kimura, H., and Ikeuchi, K. Extraction of essential interactions through multiple observations of human demonstrations. *IEEE Transactions on Industrial Electronics 50*, 4 (Aug. 2003), 667–675.

[47] Pardowitz, M., and Dillmann, R. Towards life-long learning in household robots: The piagetian approach. In *Proc. 6th IEEE International Conference on Development and Learning, London, UK* (2007).

[48] Peters, J., Vijayakumar, S., and Schaal, S. Reinforcement learning for humanoid robotics. In *Proc. of the IEEE/RSJ Int. Conf. on Humanoid Robots (Humanoids)* (2003).

[49] Rabiner, L., and Juang, B.-H. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[50] Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, December 2005.

[51] Rizzolatti, G., Fogassi, L., and Gallese, V. Neurophysiological mechanisms underlying the understanding and imitation of action. *Nature Reviews Neuroscience 2*, 9 (September 2001), 661–670.

[52] Sakoe, H., and Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on 26*, 1 (Feb 1978), 43–49.

[53] Salvador, S., and Chan, P. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal. 11*, 5 (2007), 561–580.

[54] Schaal, S. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences 3*, 6 (June 1999), 233–242.

[55] Sciavicco, L., and Siciliano, B. *Modelling and Control of Robot Manipulators*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.

[56] Sementille, A. C., Lourenço, L. E., Brega, J. R. F., and Rodello, I. A motion capture system using passive markers. In *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry* (New York, NY, USA, 2004), ACM, pp. 440–447.

[57] Shoemake, K. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), ACM Press, pp. 245–254.

[58] Sollich, P., and Williams, C. K. I. Using the equivalent kernel to understand gaussian process regression. In *Advances in Neural Information Processing Systems 17* (2005), MIT Press, pp. 1313–1320.

[59] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[60] Tolani, D., Goswami, A., and Badler, N. I. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models 62* (2000), 353–388.

[61] Tso, S., and Liu, K. Hidden markov model for intelligent extraction of robot trajectory command from demonstrated trajectories. In *Proceedings of The IEEE International Conference on Industrial Technology (ICIT)* (1996).

[62] Wagner, D., and Schmalstieg, D. ARToolKitPlus, for pose tracking on mobile devices. In *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)* (Feb. 2007).

[63] Welch, G., and Bishop, G. An introduction to the kalman filter. Tech. rep., 2004.

[64] Wolovich, W. A., and Elliott, H. A computational technique for inverse kinematics. *The 23rd IEEE Conference on Decision and Control 23* (December 1984), 1359–1363.

[65] Wu, G., van der Helm, F. C. T., Veeger, H. E. J. D., Makhsous, M., Van Roy, P., Anglin, C., Nagels, J., Karduna, A. R., McQuade, K., Wang, X., Werner, F. W., and Buchholz, B. ISB recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion - Part II: shoulder, elbow, wrist and hand. *Journal of Biomechanics 38*, 5 (May 2005), 981–992.

[66] Yabukami, S., Kikuchi, H., Yamaguchi, M., Arai, K. I., Takahashi, K., Itagaki, A., and Wako, N. Motion capture system of magnetic markers using three-axial magnetic field sensor. *IEEE Transactions on Magnetics 36*, 5 (2000), 3646–3648.

A Kernel Regression

Given N samples $X_1, \ldots, X_N$ from a random variable X, we can approximate the underlying pdf $f(x)$ using a kernel density estimator:

$$\hat{f}(x) = \frac{1}{Nh} \sum_{d=1}^{N} \kappa \left(\frac{x - X_i}{h} \right), \tag{A.1}$$

where κ is the symmetric positive kernel function and h the kernel width or parzen window size.

To regress a function we calculate the average target variable y conditioned on the input variable x, i.e. the conditional expectation $\mathbb{E}[y|x]$. As the probability distribution of both variables is described by a pdf and a version of the Bayes' theorem for continuous distributions exists, the expected value can be written as

$$\mathbb{E}[y|x] = \int_{-\infty}^{\infty} y f(y|x) dy = \int_{-\infty}^{\infty} y \frac{f(x,y)}{f(x)} dy = \frac{\int_{-\infty}^{\infty} y f(x,y) dy}{f(x)}.$$

Using the kernel density estimate as stated in Eq. A.1, we can approximate the joint distribution $f(x,y)$ and the marginal distribution $f(x)$. While estimating $f(x)$ is straightforward, $f(x,y)$ can be approximated employing the multiplicative kernel density estimator. This results in

$$\begin{aligned}
\frac{\int_{-\infty}^{\infty} y f(x,y) dy}{f(x)} &\approx \frac{\int_{-\infty}^{\infty} y \hat{f}(x,y) dy}{\hat{f}(x)} \\
&= \frac{\int_{-\infty}^{\infty} y N^{-1} h^{-1} g^{-1} \sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right) \kappa\left(\frac{y-Y_i}{g}\right) dy}{N^{-1} h^{-1} \sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right)} \\
&= \frac{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right) \int_{-\infty}^{\infty} \frac{y}{g} \kappa\left(\frac{y-Y_i}{g}\right) dy}{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right)}.
\end{aligned}$$

Keeping in mind that the kernel function is symmetric ($\kappa(u) = \kappa(-u)$) and integrates to one ($\int \kappa(u)du = 1$), the integral in the numerator can be resolved using the substitution $s = \frac{y-Y_i}{g}$:

$$\frac{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right) \int_{-\infty}^{\infty} \frac{y}{g} \kappa\left(\frac{y-Y_i}{g}\right) dy}{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right)} = \frac{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right) \int_{-\infty}^{\infty} (sg + Y_i)\, \kappa(s)\, ds}{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right)}$$

$$= \frac{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right) (Y_i \overbrace{\int_{-\infty}^{\infty} \kappa(s)\, ds}^{=1} + g \overbrace{\int_{-\infty}^{\infty} s\kappa(s)\, ds}^{=0})}{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right)}$$

$$= \frac{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right) Y_i}{\sum_{i=1}^{N} \kappa\left(\frac{x-X_i}{h}\right)}.$$

This is called the Nadaraya-Watson kernel regression model, calculating prediction values $\hat{f}(x)$ by using locally weighted averages.

B Technical Data

B.1 Servo-Electric Rotary Actuator

Type	PR 070
Nominal torque $[Nm]$	23.0
Max. torque $[Nm]$	46.0
Rotating angle $[°]$	360.0
Repeatability of positioning (accuracy)$[°]$	0.02
Weight $[kg]$	1.7
Dimensions $[m] \times [m] \times [m]$	$0.05 \times 0.05 \times 0.10$
Swiveling time (90°) with mean attached load $[s]$	0.02
Max. angular velocity $[°/s]$	150.0
Max. acceleration $[°/s^2]$	600.0
Voltage supply $[VDC]$	24.0

Table B.1: Technical data of the Schunk PowerCube. All details are the manufacturer's specifications.

B.2 The Robot Zora

Figure B.1: Zora is a mobile B21R robot. On top of the base, it has a 6-DoF manipulator consisting of Schunk PowerCube modules.