Learning Generalization and Adaptation of Movement Primitives for Humanoid Robots

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften /
Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
DISSERTATION

von

M.Sc. You Zhou

aus Sichuan, China

Date for oral defence: 12.05.2020
First Referee: Prof. Dr.-Ing. Tamim Asfour
Second Referee: Prof. Dr. Ales Ude

Zusammenfassung

Humanoide Roboter, die mit für Menschen geschaffenen Werkzeugen und Umgebungen interagieren, sollen mit der Fähigkeit ausgestattet werden, Bewegungen und Aktionen vom Menschen zu lernen, diese auf neue Aufgaben zu generalisieren und an neue Situationen zu adaptieren. Die Frage, wie Bewegungen für solche Roboter auf eine flexible Art und Weise generiert werden können, ist eine aktive und wichtige Forschungsfrage in der Robotik. Erkenntnisse aus der Neurobiologe zeigen, dass Menschen komplexe Bewegungen durch die Kombination von elementaren Bewegungseinheiten, sogenannten *Bewegungsprimitiven (Movement Primitives)* erzeugen. Diese Sichtweise stellt einen erfolgversprechenden Ansatz zur intuitiven Roboterprogrammierung dar. Deshalb beschäftigt man sich in der Robotik mit der Fragestellung, wie Bewegungsprimitiven aus menschlichen Demonstrationen gelernt werden können und wie sie auf eine flexible und adaptive Art und Weise repräsentiert werden können, um komplexe Roboteraufgaben zu programmieren.

Das Ziel dieser Arbeit ist es, ein generalisierendes und adaptives System für die Generierung von Roboterbewegungen zu entwickeln und an humanoiden Robotern zu evaluieren. Zu diesem Zweck wird eine neue Formulierung für ein parametrisches Modell zur Repräsentation von Bewegungsprimitiven, die *Via-points Movement Primitive (VMP)*, vorgeschlagen und evaluiert. Basierend auf diesem Modell werden verschiedene Ansätze entwickelt, um die Abbildung vom Aufgabenparametern auf Modellparameter zu lernen und die resultierende Bewegung bei der Ausführung an dynamischen Änderungen der Umgebung anzupassen.

Repräsentation von Bewegungsprimitiven

Inspiriert durch die Ideen und die Konzepte der Bewegungsprimitiven wird in dieser Arbeit eine neue Formulierung für die Repräsentation von Bewegungsprimitiven, die sogenannte *Via-points Movement Primitive (VMP)*, entwickelt,

die Vorteile bisheriger Methoden in der Literatur kombiniert und deren Nachteile minimiert. Die neue VMP-Formulierung erlaubt es, gelernte Bewegungen an beliebige Zwischenpunkte der Trajektorie (Via-Points) anzupassen, um veränderten Randbedingungen der Aufgabe Rechnung tragen zu können. Die Möglichkeit, beliebige Via-Points bei gleichbleibender mathematische Beschreibung der zugrundeliegenden Bewegung einfügen zu können, hat den Vorteil, dass keine weiteren Demonstrationen für die Bewegungsausführung in neuen Situationen notwendig sind und somit die Anzahl der erforderlichen Demonstrationen für die erfolgreiche Ausführung einer Aufgabe minimiert wird. Die neue Formulierung wurde in mehreren Experimenten mit mehreren Robotern evaluiert. Die Ergebnisse zeigen die Vorteile gegenüber bekannten Methoden in der Literatur, insbesondere im Hinblick auf die Extrapolationsmöglichkeiten bei der Definition von Via-Points.

Generalisierung von Bewegungsprimitiven

Im Kontext des Lernens aus Beobachtung des Menschen werden Demonstrationen in unterschiedlichen *Modi* und *Modellen* vorgeführt. Dabei kennzeichnen die Modi die unterschiedlichen Arten, mit denen der Mensch eine Bewegungen demonstriert während die unterschiedlichen Modelle die Tatsache ausdrücken, dass der Mensch unterschiedliche Aufgaben mit unterschiedlichen "versteckten"Modellen löst.

Bisherige Methoden zur aufgabenspezifischen Generalisierung von Bewegungsprimitiven berücksichtigen nur ein Modus bzw. ein Modell. Dies führt zu einem Kollabieren des Systems *(Mode and Model Collapse)*, das sich dadurch ausdrückt, dass diese Methoden keine gültige Lösung zur erfolgreichen Ausführung der zugrundeliegenden Aufgabe liefern, obwohl solche Lösungen in den Demonstrationen vorhanden sind. Das liegt daran, dass des verwendete Modus bzw. das Modell nicht alle Aspekte des Demonstrationen, insbesondere ihre Vielseitigkeit, berücksichtigt und somit gültige Lösungen nicht gefunden werden können.

In dieser Arbeit werden zwei Ansätze vorgestellt, die das Problem des *Mode and Model Collapse* lösen. Der erste Ansatz basiert auf der Erstellung eines *Mixture of Experts* Modells, das mit einem in der Arbeit entwickelten Leave-One-Out Expectation Maximization Algorithmus gelernt wird. Der zweite Ansatz sieht die Verwendung eines *Mixture Density Networks* vor, um eine nicht-balancierte Verteilung der Demonstrationen zu berücksichtigen. Die Ergebnisse zeigen,

dass die vorgeschlagenen Ansätze in ihrer Performanz derzeitige Methoden bezüglich des Lernens der Abbildung von Aufgabenparametern auf Parameter von Bewegungsprimitiven übertreffen und gleichzeitig das Problem des *(Mode and Model Collapse)* lösen.

Adaptation von Bewegungsprimitiven

Um die Anforderungen an Bewegungsprimitiven bei Aufgaben der Mensch-Roboter-Interaktion und kontaktreichen Manipulation zu erfüllen, wurde ein Leader-Follower-Framework, die *Coordinate Change Movement Primitive*, entwickelt und evaluiert. Die Grundidee ist dabei, Bewegungsprimitiven des *Followers* im Koordinatensystem des *Leaders* zu lernen und Bewegungen des Followers während der Ausführung der Aufgabe in das globale Koordinatensystem zu transformieren. Dadurch lassen sich Bewegungsprimitiven des Followers an die Bewegung des Leaders anpassen.

Weiterhin wurde eine adaptive Kraftregelung entwickelt, um bei kontaktreichen Aufgaben das gewünschte Verhalten bezüglich der vorgegebenen Kraftprofile bei gleichzeitigem Folgen der exakten Bewegungstraketorie zu realisieren. Die Regelung besteht aus einem Modell, welches das Kraftprofil vorhersagen kann, und einem adaptiven Regler, der die Steifigkeit basierend auf der Information der Kraftrückkopplung anpasst. Die vorgeschlagene Regelung wurde erfolgreich am humanoiden Roboter ARMAR-6 für eine Wischaufgabe evaluiert.

Contents

1 Introduction

Robotic research continues to play a significant role in solving societal challenges and gains increasing attention in different research communities. Robots are intelligent machines that can execute a sequence of motions to achieve a task goal.

In industry, robots are used and installed in production lines to liberate workers from repeated tedious works and increase the production efficiency and quality. However, current industrial robots rely on experts who write the programs and set up the production line. However, a complete inhuman automated factory with programmed robots is expensive and too specialized to generalize to other tasks. There is an increasing demand for cheap and intelligent robot systems that meet different industrial requirements and allow cooperation with humans.

In society, industrial robots are limited helpful as assistants for humans in daily activities. Unlike in a factory, the human-centered environment is dynamic and difficult to predict, and industrial robots are too stiff to be safe in human-robot interaction tasks.

One of the most captivating areas of current robotic research is *artificial intelligence* (AI). AI is a collection of learning algorithms that are designed to extract patterns from data or make decisions based on experience. AI simplifies the efforts of the programmers or allows programming robots by nonexperts who lack the knowledge of the robots or even programming languages. For robot motion generation, AI is reflected by the ability to generalize and adapt learned skills or motions to novel situations.

1.1 Problem Statement

The goal of this thesis is to develop a motion generation system for humanoid robots that allows learning motions from human demonstration, generalization and adaptation of learned motions to novel tasks. The underlying structure of

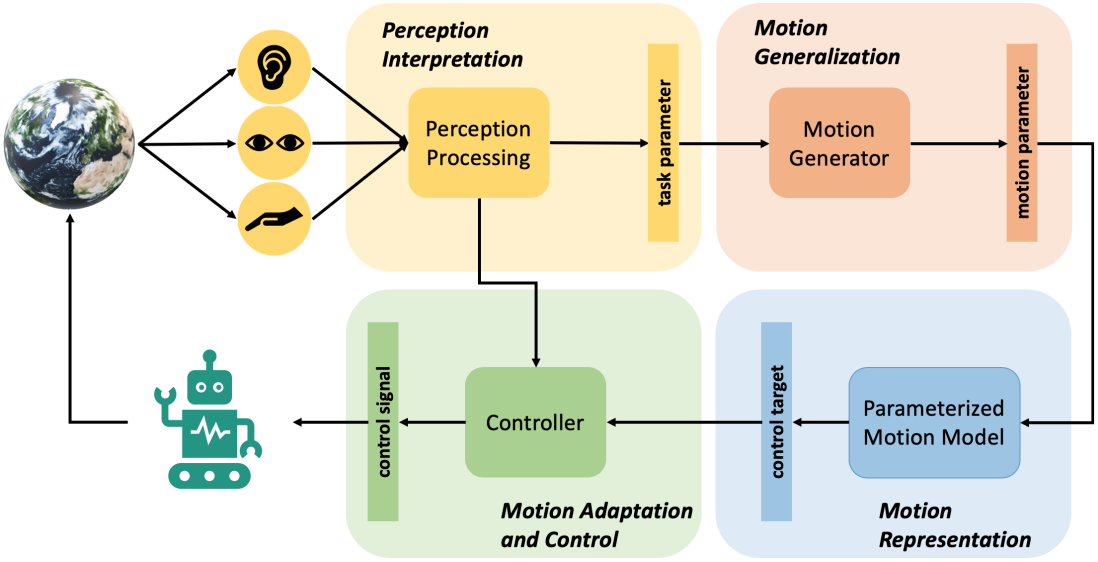

Figure 1.1: A robot motion generation system from sensory inputs to the motor commands consists of *perception interpretation*, *motion generalization*, *motion representation* and *motion adaptation*.

such motion generation system as developed in this thesis is shown in Fig. 1.1 and consists of four building blocks: *perception interpretation*, *motion generalization*, *motion representation* and *motion adaptation*.

Here, we make a difference between motion generalization and adaptation for a clear understanding of the problems and their solutions, as addressed in this thesis.

The *motion generalization* is concerned with the generation of a new motion for a new task based on previously learned motions. A *task parameter* is a set of task features (requirements or constraints), which can be represented as a numerical vector of a fixed dimension. For example, in an obstacle avoidance task, if the number of the obstacles is known and their shapes are homogeneous, a numerical vector that concatenates their positions represents the task parameter.

The *motion adaptation* is concerned with the adaptation of a learned motion to the current task requirements. In many applications, it is difficult to represent the task requirements as task parameter vector (a numerical vector of a fixed dimension). For example, if the number of obstacles is unknown, an obstacle avoidance problem is not easily solvable by motion generalization. In this case, the motion representation or the controller should be adaptable to adjust the trajectory or the control signals to accomplish the task. The development of such motion adaptation methods requires expert knowledge. Hence, it is not

generalizable to other tasks. In contrast, once a method is developed for motion generalization, it can also be used for any other task.

- **Perception Interpretation:**

  How to interpret the sensory inputs and extract the relevant information for the task execution is a challenging topic but is out of the scope of this work. In this work, we assume that the result of the perception interpretation is available and is converted to a numerical vector describing the *task parameters*.

- **Motion Generalization:**

  A motion generator takes the task parameters as input and outputs motion parameters. With an intelligent motion generator, a robot can generalize the learned motions to different situations, which are represented by task parameters. Previous approaches do not take multiple modes and models that exist in human demonstrations into consideration. Here, multiple modes mean that humans accomplish the same task with different types of motions. Multiple models refer to the fact that humans generate motions for different tasks with different hidden models.

- **Motion Representation**

  Motion representation refers to a parametric model, which represents a set of motions that can be used to accomplish a task. This model outputs a sequence of the control targets such as target positions or velocities, describing the *motion trajectory*. A good parametric model can be adjusted to adapt the generated trajectory to some temporal or spatial features such as *via-points* that are required to pass through to satisfy given task constraints. In previous works, different mathematical models are suggested. However, they have limited adaptation capabilities.

- **Motion Adaptation and Control**

  Two kinds of motion adaptation are considered in this work: trajectory and force adaptation. Trajectory adaptation refers to the spatial or temporal changes of the motion trajectory to meet the task requirements. Force adaptation is required when a desired force or torque profile should be ensured during a physical interaction between the robot and the environments.

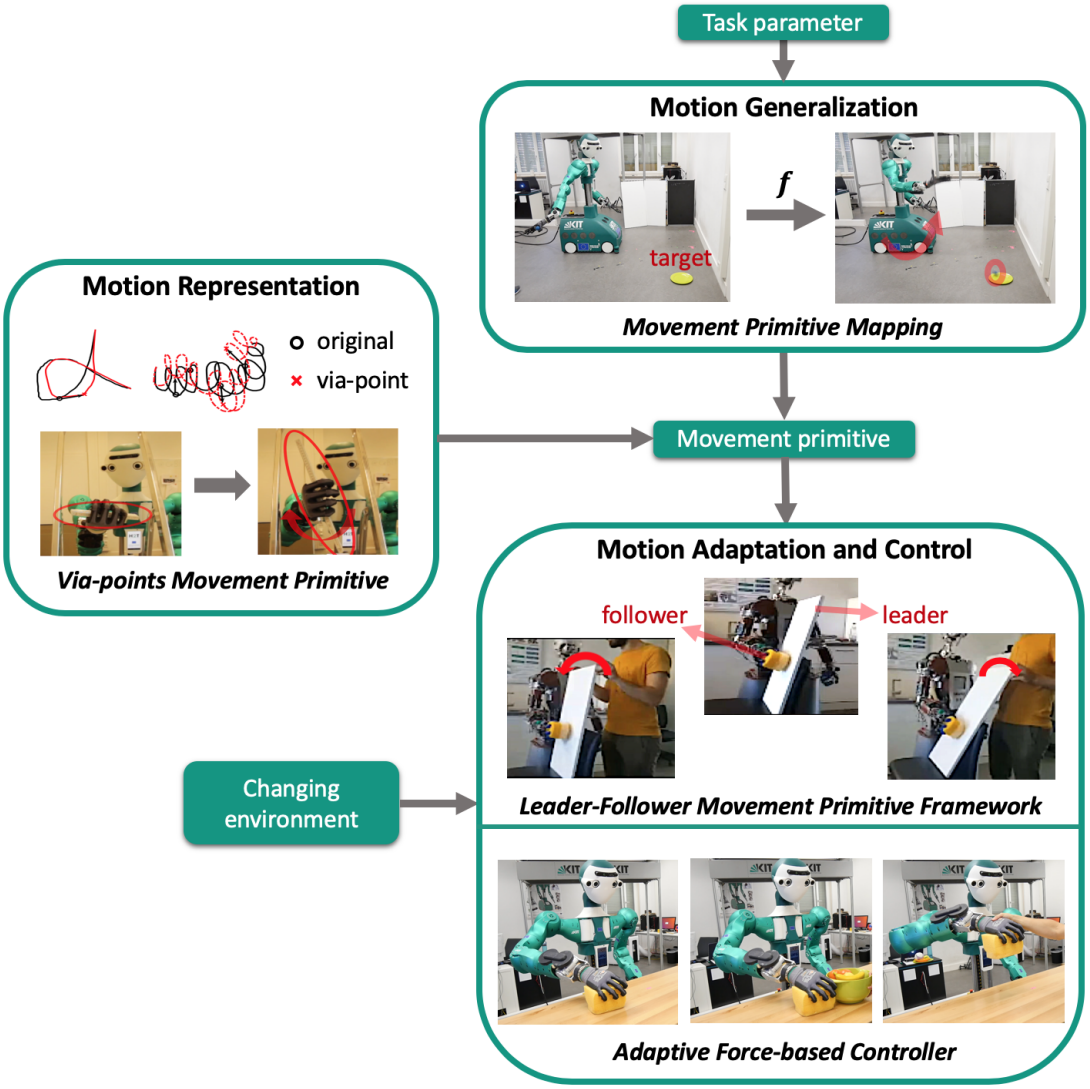

Figure 1.2: The thesis consists of three main parts: movement primitive representation, generalization and adaptation.

1.2 Learning from Demonstration

In order to generate task-oriented motions, different approaches have been developed. Recently, *reinforcement learning* (RL) combined with deep learning receives increasing attention. RL allows a robot to learn skills without supervision autonomously based on trial and error. With a given reward function of each observation or state, an RL agent improves the policy as a parametric model that maps the state to the action, to obtain higher accumulated rewards during the task execution. However, current RL methods require a large number of training epochs. Hence, they are mostly implemented in simulation.

Being different from RL, *learning from demonstrations* (LfD) is a *supervised learning* approach, where a human demonstrates the robot how to execute the task, and the robot learns from human demonstration. For many tasks where a human cannot easily give useful demonstrations, LfD cannot be applied directly. For example, it is not easy to supervise a bipedal robot for walking or running, because the robot does not have the same physical body as human. With RL, this problem can be solved, though it requires a large number of trials. If human demonstrations can be transferred to the robot, LfD is more data-efficient to generalize motions than RL. In some cases, the robot learns a skill with even only one demonstration. Many robotic skills involve only kinematic requirements, such as transporting or throwing objects. Thus, human demonstrations of these skills can be easily transferred to the robot.

In this work, LfD is used for two reasons. One reason is to learn how to generalize motions based on multiple demonstrations corresponding to different task parameters. The other reason is to learn the parametric model for motion representation. According to previous work inspired by neuro-biological research, we call this parametric model a *movement primitive* (MP) and consider it as the building block for a complex skill.

In the literature, trajectory models such as B-spline or fifth-order polynomials are used to generate desired robot trajectories. Almost all those methods are based on higher-order polynomials, because they allow calculating derivatives and hence specifying the boundary conditions such as goal positions and velocities. However, these models have limited representation capability because they have fixed structures and a small number of parameters. Furthermore, a small change of parameters may lead to a big change in the trajectory shape. In LfD, since human demonstration always can not be represented with simple functions such as polynomials, powerful parametric models are needed. An alternative is to use non-parametric models such as Gaussian process to represent motions. The problem of using non-parametric models to represent motions is that we need to store human demonstrations to be able to generate new motions, which requires a large storage space, especially for a complex task. Compared to polynomials and non-parametric approaches, a parametric movement primitive (MP) is a compact solution.

1.3 Contributions

In this work, the focus is on the development of the parametric *movement primitives* (MP), and methods that generalize a movement primitive to different task parameters or adapt it to different task requirements, and answer the question how to control the robot to follow the adapted trajectory. The contributions can be divided into three main parts:

Movement Primitive Representation

In order to improve the current state of the art regarding movement primitive adaptation to different situations, integrating via-points is considered as a rational solution. Via-points are those required points that generated trajectories should pass through. With a set of via-points, the geometrical shape of the trajectory can be adapted to new task requirements.

In this work, a new movement primitive model called *Via-points Movement Primitive* (VMP) is developed. As its name reveals, the structure of VMP allows it to adapt the trajectory to arbitrary via-points. It inherits the advantages of previous models and resolves their drawbacks regarding the adaptation to arbitrary via-points. Furthermore, compared to previous movement primitives, VMP can extrapolate to the via-points outside of the demonstration range. With via-points adaptation, VMPs fulfill a wide variety of robotic application requirements, such as obstacle avoidance.

Movement Primitive Generalization

Learning a mapping from the task parameters to the parameters of movement primitive parametric models is an intuitive way to generalize learned motions. In the literature, different regression models are developed to learn this mapping from human demonstrations. However, they do not consider multiple modes and models that exist in human demonstrations. A human accomplishes the same task specified with the same task parameters by different types of motions, which can be viewed as modes of distribution of the MP parameters. Many regression models associate one single MP parameter to one task parameter in a deterministic way, which causes the collapse of modes. The mode collapse leads to the lose of motion diversity for a task with specific task parameters. Furthermore, for different tasks with different task parameters, a

human uses more than one hidden models to generate motions. Using one single regression model causes the collapse of the hidden models, which can lead to the failure of the MP generalization.

In this work, two different methods are proposed to avoid the mode and model collapse. One method relies on using a mixture of experts model and modifying the *Expectation Maximization* (EM) algorithm to learn the expert models based on a small number of demonstrations. The other method uses a *Mixture Density Network*(MDN) to map task parameters to a mixture of Gaussian distributions of MP parameters. In order to further avoid the collapses caused by training the MDN with a normal *negative-log-likelihood* (NLL) cost, a new cost function, called *entropy cost*, is proposed for a balanced distribution of the training data to different mixture components of MDN. Furthermore, a so-called *failure cost* is introduced to improve the training process by avoiding outputting MP parameters similar to those parameters that lead to failure. In several robot experiments, compared to previous methods, the new methods show better performance.

Movement Primitive Adaptation and Control

For movement primitive adaptation, a leader-follower framework based on *Coordinate Change Movement Primitive* (CC-MP) is developed for tasks involving multiple agents such as wiping a moving surface or bimanual manipulation tasks. The follower's movement primitive is learned in the leader's local frame and can be adapted to the leader's motion, which can also be encoded by a leader movement primitive.

Furthermore, a force-predictive-model based adaptive controller is developed to allow the compliant behavior of a robot while accurately tracking a given trajectory. This controller enables a safe human-robot interaction in collaborative tasks.

1.4 Structure of the Thesis

Chapter. 2 introduces and discusses related works regarding movement primitive representation, generalization and adaptation. In the context of movement primitive representation, different non-parametric or parametric methods are introduced. Their merits and disadvantages are described and compared. For movement primitive generalization, previous works are divided into two

categories: learning a direct mapping or learning a generative model and are discussed and compared. For movement primitive adaptation, methods are introduced for specific tasks such as obstacle avoidance, handover and force adaptation. Furthermore, several control strategies are introduced to enable compliant behavior of robots.

Chapter. 3 discusses further parametric models and introduces a new model, as novel representations of movement primitives, the so-called *Via-points Movement Primitive* (VMP), which inherits the benefits from previous methods and resolves their drawbacks. The comparison between VMP and the previous methods such as *Dynamic Movement Primitive* (DMP), *Probabilistic Movement Primitive* (ProMP) shows the strengths of the VMP representation regarding inter- and extrapolation capabilities for via-points adaptation. The chapter also includes several robot applications based on VMP and shows that the via-points adaptation can simplify robot tasks.

Chapter. 4 introduces two methods to handle multiple modes and models that exist in human demonstration. The first method is learning mixture of experts with a *Leave-One-Out Expectation Maximization* (LOO-EM) algorithm. LOO-EM shows better performance for a small number of demonstrations than the normal EM algorithm. The second method uses a *Mixture Density Network* (MDN) to model a mapping from task parameters to a mixture of Gaussian distribution of MP parameters. In order to further reduce the occurrence of the mode and model collapse, a so-called *entropy cost* is introduced to distribute the demonstrations to different mixture components of MDN in a balanced way. Furthermore, a *failure cost* function is developed to improve the training process by keeping MDN from outputting those MP parameters that lead to failure. Several robot experiments show that MDN outperforms previous methods.

Chapter. 5 introduces a leader-follower framework to enable the adaptation of the robot's motion to humans' behavior and the collaboration between multiple agents. The follower movement primitive is encoded in the leader's local frame. Thus, the follower can adapt its motion to the leader's motion during task execution. With this framework, a robotic wiping system is developed. The chapter also includes an adaptive control framework based on the force predictive model developed to achieve compliant robot behavior while accurately tracking an MP generated trajectory.

Chapter. 6 concludes the work and discusses future possible extensions.

2 Related Work

In this chapter, we separately introduce the methods developed for *movement primitive* (MP) representation, generalization, and adaptation.

The movement primitive representation refers to different mathematical models that represent motions. These models are the basic building blocks for a robot skill. Different motion representations have their advantages and disadvantages. They are usually a parametric model with a set of parameters that uniquely define a set of motions. Section. 2.1 introduces and compares important or popular methods.

The movement primitive generalization is concerned with a regression model that maps the task parameters to the MP parameters. Works in the literature focus on a specific set of task parameters or a better generalization for some particular tasks. Other works investigate different regression models and whether they are suitable for MP generalization or not. With different MPs, generalization can have different meanings. In Section. 2.2, MP generalization approaches are introduced and compared.

For movement primitive adaptation and control, methods are developed to adapt the MP generated trajectory to the dynamic environment or different task requirements. In contact-rich manipulation, one crucial task is to deal with or control the contact force with the environment. An another important task is to control the robot to follow the MP generated trajectory with a high accuracy while being compliant when encountering the external perturbations.

2.1 Motion Representation

For representing robot motions, researchers focus on developing compact motion representations that are simple to learn and convenient to use. The early electrophysiological studies on the frogs and rats by Bizzi Emilio and others (Bizzi et al. (1991), Mussa-Ivaldi and Bizzi (2001)) suggested that complex motions are generated by combining elementary force fields. These force fields

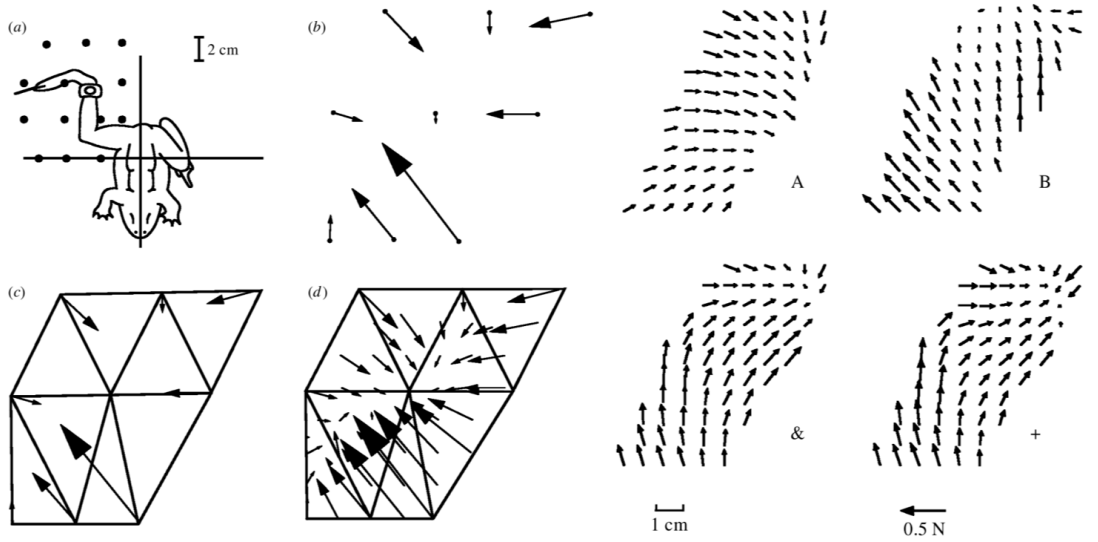

Figure 2.1: The left four figures show the experiment on the spinalized frog conducted by Bizzi Emilio and co-workers. The force field in (d) is obtained by special interpolation with triangles. The right figures show that superposition $(+)$ of two force fields in A and B by stimulating different loci of the spinal cord coincides with the force field generated by stimulating them simultaneously. The experiment supports the assumption that complex motions are realized by the combination of some elementary force fields. (The figures are from Bizzi et al. (1991) and Mussa-Ivaldi and Bizzi (2001))

are called *motor primitives*, or *movement primitive* (MP) because they are fundamental building blocks for complex skills. Inspired by these experiments, in the robotic community, MP refers to those basic motion units that can be executed by the robot. By ordering and sequencing MPs, complex skills can be implemented.

There are a large number of MP representations in literature. They have their strength and weakness because they are developed based on different assumptions and for different purposes. One essential purpose is to simplify the adaptation of motion to the new task requirements. For example, in order to deal with unforeseen objects and changed targets, it is much easier to deform the ongoing trajectory instead of generating new trajectories (Pham and Nakamura (2015)). Hence, many motion representation methods are usually equipped with strategies to handle new targets or obstacles.

Before considering different movement primitives in literature, we introduce here a set of traditional methods for the robotic motion generation.

2.1.1 Trajectory Optimization based on Human Motion Rules

These methods solve a constraint optimization problem to obtain a motion trajectory. The objective function is designed based on a large number of experiments, where human motions are thoroughly studied. The purpose is to create the most human-like motions for robots based on the assumption that human motions follow some basic rules, which can be reformulated by the objective functions to be minimized or maximized. The constraints of those optimization problems are the task requirements which the generated motion trajectories should fulfill.

In Lacquaniti et al. (1983), the authors found that human planar drawing motions follow a so-called two-third power law, which says that the angular velocity ω and the trajectory curvature κ follow the relationship $\omega(t) = \gamma\kappa(t)^{2/3}$ with a constant γ. In Pollick and Sapiro (1997), it is proved that the motion obeying the power law has a constant equi-affine linear velocity. This law was further extended to 3D trajectories in Maoz et al. (2009). In Bennequin et al. (2009), a theory is developed to describe the property of human movements based on the affine invariance: trajectories generated by humans are invariant concerning affine transformations that include equi-affine and Euclidean transformations. This fact gives the birth of motion generation methods based on the optimization with objective functions that consider the affine invariance.

In Pham and Nakamura (2015), the authors proposed a strategy to formulate an optimization problem to create a deformed trajectory, which has the same equi-affine speed profile as the original one. This equi-affine velocity defined in Maoz et al. (2009) is a scalar triple product of the first, second and third derivative of the trajectories:

$$v_{ea}(t) = |\frac{d\boldsymbol{\xi}}{dt}, \frac{d^2\boldsymbol{\xi}}{dt^2}, \frac{d^3\boldsymbol{\xi}}{dt^3}|^{1/6}, \tag{2.1}$$

where $|.,.,.|$ is the scalar triple product and $\boldsymbol{\xi}$ is the corresponding trajectory. In order to generate the trajectory with the same equi-affine speed profile, the authors allow only affine transformations of the original trajectory for the new trajectory which meets the task requirements. Even though this is the case, there still exist a large number of redundant trajectories which meet the same requirements. The objective function of the optimization problem, thus, provide a way to find a unique trajectory, or more precisely an affine transformation matrix M. For example, one cost function minimizes the distance between

the generated trajectory and the original one, namely

$$\min_{M} \sup_{t<T} ||\boldsymbol{\xi}(t) - \boldsymbol{\xi}_0(t)||, \tag{2.2}$$

where T is the terminal time of the trajectory. The affine transformation can be written as $\boldsymbol{\xi}(t) = \boldsymbol{\xi}_0(0) + M(\boldsymbol{\xi}_0(t) - \boldsymbol{\xi}_0(0))$, where $\boldsymbol{\xi}_0(0)$ is the start point of the trajectory. Because

$$||\boldsymbol{\xi}(t) - \boldsymbol{\xi}_0(t)|| = ||\boldsymbol{\xi}_0(0) + M(\boldsymbol{\xi}_0(t) - \boldsymbol{\xi}_0(0)) - \boldsymbol{\xi}_0(t)|| \leq ||M - I||\,||\boldsymbol{\xi}_0(t) - \boldsymbol{\xi}_0(0)||, \tag{2.3}$$

the problem is to minimize the norm $||M - I||$. By flattening the matrix M to a vector $\boldsymbol{m}$, the objective function is approximated by the Frobenius norm which is $||\boldsymbol{m} - \boldsymbol{i}||$, where $\boldsymbol{m}_{kl} = M(k, l)$, $\boldsymbol{i}_{kl} = I(k, l)$ with $0 \leq k, l < d$ and d is the dimension of the trajectory. The optimization with this specific cost function is represented as

$$\min_{m} ||\boldsymbol{m} - \boldsymbol{i}||^2, \tag{2.4}$$

which is a quadratic programming that can include equality or inequality constraints and can be solved by many software packages. Other objective functions were introduced in Pham and Nakamura (2015). The main idea is still to find an affine transformation that deforms the original trajectory for some specific task constraints.

In Meirovitch et al. (2016), the authors suggest a method that solves an optimization problem whose objective function is given as

$$I_{\boldsymbol{\xi}_0,T,\lambda}(\boldsymbol{\xi}, \dddot{\boldsymbol{\xi}}) = \int_0^T |\dddot{\boldsymbol{\xi}}|^2 dt + \lambda^6 \int_0^T |\boldsymbol{\xi} - \boldsymbol{\xi}_0|^2 dt, \tag{2.5}$$

where $\boldsymbol{\xi}_0$ is the template trajectory. The optimization is to find a trajectory $\boldsymbol{\xi}$ that is as close to the original trajectory and smooth as possible. The tradeoff between smoothness and accuracy depends on the Lagrange multiplier λ. Based on the Euler-Lagrange equations, the solution should satisfy the equation:

$$\boldsymbol{\xi}^{(6)}(t) = \lambda^6(\boldsymbol{\xi}(t) - \boldsymbol{\xi}_0(t)), \tag{2.6}$$

where $\boldsymbol{\xi}^{(6)}(t)$ is the 6-th derivative of the trajectory at the time point t. This equation is invariant with respect to the affine transformation because both

colinearity and the differentiation are affine invariant:

$$\frac{d^6}{dt^6} M(\boldsymbol{\xi}(t)) = \lambda^6 \left(M\left(\boldsymbol{\xi}(t)\right) - M\left(\boldsymbol{\xi}_0(t)\right) \right). \tag{2.7}$$

By adjusting the Lagrange multiplier λ, the generated trajectory goes from a minimum jerked approaching motion to an exact reproduction of the original template.

Based on a large number of researches for human motion generation, the resulting trajectories of these optimization problems are theoretically human-like. With the constraints, they can accomplish the tasks as well. The problem of these methods is that it is difficult to generalize them for different task parameters. For each new task parameter, a new optimization problem is to be solved, which is not convenient, especially when the task is complex.

2.1.2 Generative Model of Trajectory Points

In Calinon et al. (2007), the authors used a statistical learning approach to learn a generative model of the trajectory points. The trajectory point is referring to a pair $(\boldsymbol{\xi}_t, \boldsymbol{\xi}_s)$, where $\boldsymbol{\xi}_t$ is the temporal parameter and $\boldsymbol{\xi}_s$ represents the spatial parameter such as positions. After a number of demonstrations, a set of trajectory points is collected. For example, for M demonstrations with T time steps each, the training dataset will have totally $M \cdot T$ points. In Calinon et al. (2007), the authors use a *Gaussian Mixture Model* (GMM) to represent the distribution of the trajectory points:

$$p(\boldsymbol{\xi}_t, \boldsymbol{\xi}_s) = \sum_{k=1}^{K} \boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{\xi}_{t,s} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{2.8}$$

where K is the number of the mixture components, $\boldsymbol{\pi}_k$ is a *mixing coefficient*, and

$$\boldsymbol{\mu}_k = \begin{pmatrix} \boldsymbol{\mu}_{t,k} \\ \boldsymbol{\mu}_{s,k} \end{pmatrix}, \quad \boldsymbol{\Sigma}_k = \begin{pmatrix} \boldsymbol{\Sigma}_{t,k} & \boldsymbol{\Sigma}_{ts,k} \\ \boldsymbol{\Sigma}_{st,k} & \boldsymbol{\Sigma}_{s,k} \end{pmatrix} \tag{2.9}$$

With a fixed number of mixture components K, GMM can theoretically represent any distribution.

Based on the conditional probability, which is used to infer unknown variables conditioning on the known ones, we get a distribution of the trajectory positions $\boldsymbol{\xi}_s$ conditioning on the temporal parameter $\boldsymbol{\xi}_t$. The conditional mean and

covariance of the k-th Gaussian distribution are:

$$\begin{aligned} \hat{\boldsymbol{\xi}}_{s,k} &= \boldsymbol{\mu}_{s,k} + \boldsymbol{\Sigma}_{st,k}(\boldsymbol{\Sigma}_{t,k})^{-1}(\boldsymbol{\xi}_t - \boldsymbol{\mu}_{t,k}) \\ \hat{\boldsymbol{\Sigma}}_{s,k} &= \boldsymbol{\Sigma}_{s,k} - \boldsymbol{\Sigma}_{st,k}(\boldsymbol{\Sigma}_{t,k})^{-1}\boldsymbol{\Sigma}_{ts,k}. \end{aligned} \tag{2.10}$$

GMM also learns the relationship among individual dimensions of the spatial parameter. Hence, it is also possible to determine the trajectory of one dimension from the others. For example, a joint space trajectory can be encoded together with a task space trajectory. Moreover, the null space control targets are obtained when executing a task space motion.

With this generative model, the authors constructed an objective function based on the mean of the conditional distribution mentioned before. Together with the kinematic constraints, an optimization problem can be solved to obtain the desired trajectory.

The method to infer the unknown conditioning on the known variables from a GMM is called *Gaussian Mixture Regression* (GMR). In fact, GMR learns a function f that maps the temporal parameter $\boldsymbol{\xi}_t$ to the spatial parameter $\boldsymbol{\xi}_s$:

$$\boldsymbol{\xi}_s = f(\boldsymbol{\xi}_t). \tag{2.11}$$

In Calinon et al. (2013), the authors further improved their methods by considering different perspectives for the demonstrations. With this improvement, they generalize the approach to the change of specific task parameters. And the method is called *Task-Parameterized Gaussian Mixture Model* (TP-GMM). TP-GMM will be described in detail in Section. 2.2.

2.1.3 Dynamic Movement Primitives

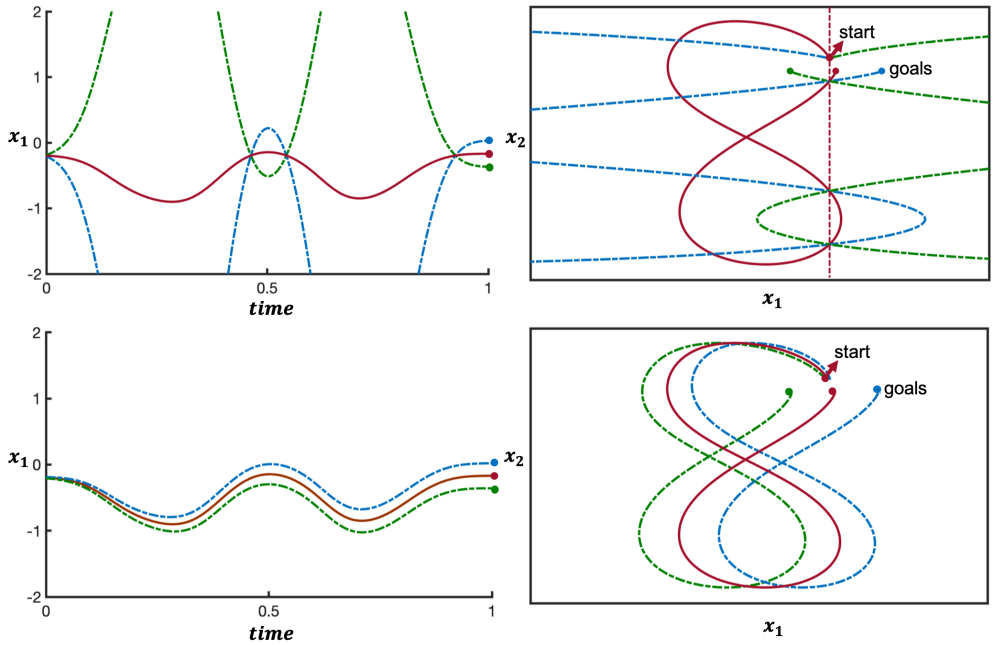

Figure 2.2: Two drawbacks of DMP result from the scaling multiplier $(g - y_0)$. The first drawback is that DMP will generate a mirror trajectory if $(g_{new} - y_0) = -(g - y_0)$. The other one is that DMP generates infeasible trajectories if $g \approx y$ in the demonstrated trajectory. The force term is numerically large and hence unstable when changing to a new goal. The **top-left** figure shows the plots regarding the first dimension of the trajectories of a figure "8" in a 2D plane. The **top-right** figure shows the result drawing. The red point and the red trajectories represent the original goal and the trajectory. The blue trajectories are the results of adapting to the goal on the right side of the start. The green trajectory is the result of adapting to the goal on the left side of the start. The bottom two figures show the results given by the *bio-inspired DMP* given in Hoffmann et al. (2009).

In Ijspeert et al. (2003) and Schaal (2003), the authors proposed the *Dynamic Movement Primitive* (DMP) based on both dyanmical system and statistical learning approaches.

In correspondence to the force field observed in the frog experiment (Fig. 2.1), DMPs assume that the motion is governed by an elastic force field with a global attractor g, namely a well-studied spring system, and a non-linear force term $f(x)$ as a function of a phase variable x, which is learned from the human

demonstration. With a stiffness K and a damping factor D, a DMP outputs a scaled acceleration profile $\dot{v}$ using the so-called *transformation system*:

$$\begin{aligned} \tau\dot{v} &= K \cdot (g - y) - D \cdot v + (g - y_0) \cdot f(x) \cdot x \\ \tau\dot{y} &= v, \end{aligned} \tag{2.12}$$

where $y, v, \dot{v}$ are the position, scaled velocity and acceleration. The phase variable x, also called *canonical variable*, is a time-dependent variable going from 1 to 0 and determined by an exponential decay system. Ijspeert et al. (2003) and Schaal (2003) call it *canonical system*:

$$\tau\dot{x} = \alpha_x x, \quad with \quad x(0) = 1, x(T) = 0, \tag{2.13}$$

where T is the terminal time and α_x is a constant. With the introduction of the canonical system, the previous transformation system is time independent. With the temporal scaling τ, the motion speed is adjustable.

In Ijspeert et al. (2003) and Schaal (2003), the non-linear force term $f(x)$ is parameterized with a kernelized linear regression model :

$$f(x) = \frac{\sum_{i=1}^{N} \boldsymbol{\psi}_i(x)\boldsymbol{w}_i}{\sum_{i=1}^{N} \boldsymbol{\psi}_i(x)}, \tag{2.14}$$

where N is the number of kernels and $\boldsymbol{w}$ is the parameter vector. The function $\boldsymbol{\psi}(\cdot)$ is a *squared exponential kernel* (SEK), also called *radial basis function* (RBF):

$$\boldsymbol{\psi}_i(x) = exp(-\frac{1}{2\sigma_i^2}(x - c_i)^2) \tag{2.15}$$

where σ_i and c_i are the parameters of the i-th SEK, which are predefined or learned from the demonstration. The parameter $\boldsymbol{w}$ is learned with the *Locally Weighted Regression* (LWR) described in Atkeson et al. (1997). The goal is to minimize the error of the parametric function to the desired force term values f_{target}. With one demonstration, M data samples $\{(x_t, y_{demo,t}, \dot{y}_{demo,t})\}_{t=1}^{M}$ are collected along the timeline. The target force term is calculated with the Eq. 2.12:

$$f_{target}(x) = \frac{\tau\dot{v}_{demo} - (K \cdot (g - y_{demo}) - D \cdot v_{demo})}{(g - y_{0,demo}) \cdot x}. \tag{2.16}$$

With a training dataset $\{(x_t, f_t)\}_{t=1}^{M}$, the cost function is

$$L = \sum_{t=1}^{M} \sum_{i=1}^{N} \boldsymbol{\psi}_i(x_t)(\boldsymbol{w}_i - f_t)^2. \tag{2.17}$$

By setting the first derivative of the cost function to zero, the k-th component of the weights vector is obtained by

$$\boldsymbol{w}_k = \frac{\sum_{t=1}^{M} \boldsymbol{\psi}_k(x_t) f_t}{\sum_{t=1}^{M} \boldsymbol{\psi}_k(x_t)}. \tag{2.18}$$

One advantage of LWR is that the weights vector $\boldsymbol{w}$ can be learned in an incremental way. In Gams et al. (2010) and Gams et al. (2016), an incremental learning strategy was developed to learn a periodic motion with DMP. Hereby, the update of the weights vector with each new data point (x_t, f_t) is given by

$$\boldsymbol{w}_{k,t+1} = \boldsymbol{w}_{k,t} + \frac{\boldsymbol{\psi}_k(x_{t+1})}{\sum_{i=1}^{t+1} \boldsymbol{\psi}_k(x_i)} (f_{t+1} - \boldsymbol{w}_{k,t}). \tag{2.19}$$

With a variable

$$P_t = \frac{1}{\sum_{i=1}^{t} \boldsymbol{\psi}_k(x_t)}, \tag{2.20}$$

and its update rule

$$P_{t+1} = \frac{P_t}{1 + P_t \cdot \boldsymbol{\psi}_k(x_{t+1})} = P_t - \frac{P_t^2 \cdot \boldsymbol{\psi}_k(x_{t+1})}{1 + P_t \cdot \boldsymbol{\psi}_k(x_{t+1})}, \tag{2.21}$$

The update rule of $\boldsymbol{w}$ follows:

$$\boldsymbol{w}_{k,t+1} = \boldsymbol{w}_{k,t} + \boldsymbol{\psi}_k(x_{t+1}) \cdot P_{t+1} \cdot (f_{t+1} - \boldsymbol{w}_{k,t}). \tag{2.22}$$

The process starts with each component equal to zero $\boldsymbol{w}_{k,0} = 0$ and $P_0 = 1$.

The DMP formulation is developed for a compact representation of robot motions. With the previously mentioned approach, it is learned from one single demonstration. Once it is learned, it adapts to different speeds, starts, and goals by changing its hyper-parameters τ, g, y_0. Compared to the previous methods, one advantage of a DMP is its adaptation capability, because the motion goal is one of the standard task requirements in many applications. Without any efforts, a DMP directly adapts to the goal change.

However, the DMP formulation proposed in Ijspeert et al. (2003) and Schaal (2003) has the scaling problem when adapting to a new start or goal. This fact is because the multiplier $(g - y_0)$ for the non-linear force term causes the numerical problem when calculating the desired force term from the demonstration. In Fig. 2.2, a DMP is used to learn a figure eight motion in 2D. Since the start and goal are close to each other in the demonstration (red curve), the scaling term $(g - y_0)$ is close to zero, which results in a significant target force term. Af-

ter learning the DMP parameters with the LWR, a small change of the scaling term $(g - y_0)$ causes a dramatic change of the trajectory shape because of the tremendous force term (the blue curve). Furthermore, the generated trajectory flips (the green curve) when the new goal is on the other side of the new start and leads to the sign change of the scaling term $(g - y_0)$.

In order to solve these problems, instead of a damped spring system combined with a force term, in Hoffmann et al. (2009), the authors suggest the superposition of two elastic force fields,

$$\tau \dot{v} = x \cdot K(f(x) + y_0 - y) + (1 - x) \cdot K(g - y) - Dv, \tag{2.23}$$

where $K(g - y)$ corresponds to the original spring system with a fixed global goal g and $K(f(x) + y_0 - y)$ is an elastic force field generated by a moving attractor $f(x) + y_0$, which takes the place of the original force term. The weights of these two force fields change with the canonical variable x going from 1 to 0. In the beginning, the moving attractor mainly governs the motion. During execution, the global goal attractor gradually takes over. As in Hoffmann et al. (2009) described, inspired by the frog experiment in Bizzi et al. (1991)(Fig. 2.1), the authors called their method *bio-inspired DMP*. There is no scaling term in this formulation, and the force term $f(x)$ encodes the offset from y_0 for the moving attractor. The *bio-inspired DMP* scales the motion automatically and solves the scaling problem, as shown in Fig. 2.2.

In Dragan et al. (2015), the authors proved that the *bio-inspired DMP* adapts to the new start and goal by minimizing the norm $A = K^T K$, where K is the finite differencing matrix, such that:

$$\begin{aligned} \underset{\boldsymbol{\xi}}{\text{minimize}} \quad & ||\boldsymbol{\xi} - \boldsymbol{\xi}_D||_A^2 \\ s.t. \quad & \boldsymbol{\xi}(1) = y_0 \\ & \boldsymbol{\xi}(0) = g_{new}, \end{aligned} \tag{2.24}$$

where $\boldsymbol{\xi}$ and $\boldsymbol{\xi}_D$ are the target and the demonstrated trajectory and $||\boldsymbol{\xi}||_M^2 = \boldsymbol{\xi}^T M \boldsymbol{\xi}$. The norm A is a distance measure between two trajectories:

$$C(\boldsymbol{\xi}, \boldsymbol{\xi}_D) = \frac{1}{2} \int (\dot{\boldsymbol{\xi}}(x) - \dot{\boldsymbol{\xi}}_D(x))^T (\dot{\boldsymbol{\xi}}(x) - \dot{\boldsymbol{\xi}}_D(x)) dx = \frac{1}{2} (\boldsymbol{\xi} - \boldsymbol{\xi}_D)^T A (\boldsymbol{\xi} - \boldsymbol{\xi}_D). \tag{2.25}$$

The goal adaptation of the DMP minimizes the difference between the velocity profiles of the demonstrated and the generated trajectory.

In Ijspeert et al. (2013), the authors provided a comprehensive review on DMPs

and its applications. According to Ijspeert et al., a DMP serves as a kinematic planner, and the robot is usually equipped with a tracking controller such as a PD controller with inverse dynamic methods. In order to accommodate the external perturbation, especially in a compliant control mode, the canonical system is modified to incorporate a feedback term called *phase stopping*:

$$\tau\dot{x} = -\frac{1}{T} \cdot \frac{1}{1 + \alpha_{err}||\tilde{y} - y||}, \quad or \quad \tau\dot{x} = \alpha_x \cdot \frac{x}{1 + \alpha_{err}||\tilde{y} - y||}. \tag{2.26}$$

where y is the DMP state and $\tilde{y}$ is the current position. The phase stopping technique for the canonical system makes a DMP a feedback trajectory planner that stops when the controller prohibits to follow the planned trajectory for some reasons such as human interruptions or poor tracking accuracy of the lower-level controller. In real applications, the transformation system of a DMP provides only the forward control signal with y as the so-called DMP state without receiving any feedback signal from the robot.

This control strategy questions the necessity to form the elastic force fields (second-order dynamical systems) in the transformation system instead of directly encoding the trajectory. Moreover, a dynamical system increases the complexity of the problem by requiring a solver for differential equations such as Euler or Runge-Kutta methods and limits the adaptability of DMPs for the trajectory adaptation.

The fact that a DMP can learn from one single demonstration is both its advantage and disadvantage. It is an advantage because it can be easily learned and hence, meets a lot of application requirements, where multiple demonstrations are not available or costly. It is a disadvantage because one demonstration contains only little information about the task. The DMP adaptation is merely due to its structure or its assumption about the movement. Without the knowledge from multiple demonstrations, there is no reason to claim that the DMP adaptation to the new start or goal is "correct" for the task.

2.1.4 Probabilistic Movement Primitives

In order to also consider the variance of the demonstrations for a given task, a probabilistic model called *Probabilistic Movement Primitive* (ProMP) was developed in Paraschos et al. (2013, 2018). With the fundamental operations in probability theory, ProMPs are connectable, combinable, and modifiable.

Instead of a dynamical system, a ProMP directly encodes the positions and

velocities of a trajectory as follows:

$$\boldsymbol{y}_t = \boldsymbol{\psi}_t^T \boldsymbol{w} + \epsilon_y, \quad \boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_{\boldsymbol{w}}\right), \tag{2.27}$$

where $\boldsymbol{y}_t = (\boldsymbol{\xi}_t, \dot{\boldsymbol{\xi}}_t)^T$ is the state of the system, $\epsilon_{\boldsymbol{y}} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_{\boldsymbol{y}})$ is a Gaussian noise and

$$\boldsymbol{\psi}_t = \begin{pmatrix} \boldsymbol{\psi}_1(t) & \dot{\boldsymbol{\psi}}_1(t) \\ \boldsymbol{\psi}_2(t) & \dot{\boldsymbol{\psi}}_2(t) \\ \vdots & \vdots \\ \boldsymbol{\psi}_N(t) & \dot{\boldsymbol{\psi}}_N(t) \end{pmatrix} \tag{2.28}$$

where $\boldsymbol{\psi}_i(t)$ is i-th *squared exponential kernel* (SEK) function (Eq. 2.15) and $\dot{\boldsymbol{\psi}}_i(t)$ is its corresponding first derivative. If considering the time independence and the same canonical system used in a DMP, the time point t is simply replaced by the canonical variable x. Hence, $\boldsymbol{\psi}_i(t) = \boldsymbol{\psi}_i(x_t)$.

Learning a ProMP means inferring the parameters of the Gaussian distribution in Eq. 2.27. The *maximum likelihood estimation* (MLE) of its mean and its variance are the empirical mean and covariance:

$$\boldsymbol{\mu}_w = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{w}_i, \quad \boldsymbol{\Sigma}_w = \frac{1}{M}\sum_{i=1}^{M} (\boldsymbol{w}_i - \boldsymbol{\mu}_w)(\boldsymbol{w}_i - \boldsymbol{\mu}_w)^T, \tag{2.29}$$

where M is the number of demonstrations and $\boldsymbol{w}_i$ is the i-th weights vector corresponding to the i-th demonstration. If a ProMP is used to encode a multi-dimensional trajectory, the weights vectors to represent each dimension are concatenated (Paraschos et al. (2018)). The weights vector $\boldsymbol{w}_i$ for the i-th demonstration is obtained by solving a regularized least square problem:

$$\boldsymbol{w}_i = \left(\boldsymbol{\Psi}^T\boldsymbol{\Psi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Psi}^T \boldsymbol{\xi}_i, \tag{2.30}$$

where

$$\boldsymbol{\Psi} = \begin{pmatrix} \boldsymbol{\psi}(x_0)^T \\ \boldsymbol{\psi}(x_1)^T \\ \vdots \\ \boldsymbol{\psi}(x_T)^T \end{pmatrix}, \tag{2.31}$$

and $\boldsymbol{\xi}_i$ is a vector representing the trajectory of the i-th demonstration.

With such probabilistic formulation, a ProMP provides a more compact representation than a DMP. Unlike DMP goal adaptation that changes the attractor of the damped spring system without using any information from human

demonstration, the adaptation of a ProMP is entirely data-driven. As shown in the left-most diagram in Fig. 2.3, the low variance of the Gaussian distribution corresponds to a critical region between two obstacles. During the execution, a ProMP guarantees that the motion goes through these low variance regions, which is not achievable with a DMP.

Since a ProMP directly encodes the trajectory with a probability distribution, it adapts to any via-points, including both the start and goal with the conditional probability. We can use a triple $(x^*, \boldsymbol{y}^*, \boldsymbol{\Sigma}_y^*)$ to denote a via-point, where x^* is the canonical variable that is corresponding to the time at which the trajectory should go through the via-point, $\boldsymbol{y}^*$ is the via-point positions and $\boldsymbol{\Sigma}_y^*$ represents its uncertainty.

For a new via-point, the conditional probability of the weights vector is given by

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{w}}^* &= \boldsymbol{\mu}_{\boldsymbol{w}} + L(\boldsymbol{y}^* - \boldsymbol{\psi}_{t^*}^T \boldsymbol{\mu}_{\boldsymbol{w}}), \\ \boldsymbol{\Sigma}_w^* &= \boldsymbol{\Sigma}_w - L\boldsymbol{\psi}_{t^*}^T \boldsymbol{\Sigma}_w, \\ L &= \boldsymbol{\Sigma}_w \boldsymbol{\psi}_{t^*} (\boldsymbol{\Sigma}_{y^*} + \boldsymbol{\psi}_{t^*}^T \boldsymbol{\Sigma}_w \boldsymbol{\psi}_{t^*})^{-1}, \end{aligned} \tag{2.32}$$

and $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}}^*, \boldsymbol{\Sigma}_{\boldsymbol{w}}^*)$. The mean trajectory definitely goes through the via-point. The new covariance determines the freedom of the generated trajectory around the via-points. In the middle diagram of Fig. 2.3, ProMP adapts the distribution to a new goal. With a low variance, it guarantees that the generated trajectory hits the new target. At the same time, the result trajectories distribution keeps the low variance for the critical region in between the two obstacles. Since a DMP does not learn variance information and adapts to the goal without using knowledge from human demonstration, there is no guarantee that the generated trajectory for the new goal does not collide with the obstacles. As shown in the right diagram of Fig. 2.3, a ProMP also adapts to an arbitrary via-point. While a DMP cannot directly adapt to other via-points except the new start and goal.

In order to utilize ProMPs for robot applications, a stochastic feedback controller developed in Paraschos et al. (2013, 2018) reproduces the mean and the variance for all time steps of a given trajectory distribution. However, this controller was only designed for linear dynamical systems or physical systems, which can be linearized. For multi-joints robot control, a PD feedback controller and an inverse dynamic method after the stochastic controller are still necessary to realize the robot motions as for DMPs. The advantage of a stochastic controller with ProMPs over DMPs is that the trajectory shows more compliant behavior in case of perturbation because the variance of a ProMP allows the

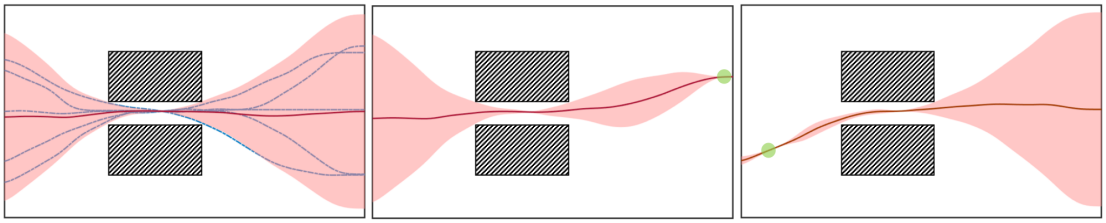

Figure 2.3: ProMP is learned with a number of collision free demonstrations (the blue curves). The learned ProMP represents a trajectory distribution with the red curve as the mean trajectory and the red region shows the variance. ProMP adapts to the new goal and any intermediate via-points shown in the right two diagrams with green balls.

robot to depart from the mean trajectory. While a DMP, with high gains K for accurate reproduction of the demonstrated trajectory, is less compliant regarding the perturbation and tries quickly to return to the original trajectory.

In Ewerton et al. (2015), the authors replaced the Gaussian distribution in the ProMP formulation with a *Gaussian Mixture Model* (GMM). With a GMM, a ProMP has a more powerful distribution representation. As usual, the GMM is learned with the *Expectation Maximization* algorithm (EM).

The weights vector $\boldsymbol{w}$ in the DMP and ProMP formulations can be regarded as a data point in a high dimensional movement primitive (MP) space. With the DMP structure, each MP in this space denotes a family of trajectories that have different starts and goals but are topologically similar (Ijspeert et al. (2013)). While it only represents one single trajectory with a ProMP. If multiple demonstrations for one specific task are available, a ProMP infers the parameters of the Gaussian distribution or a GMM. However, with a small number of demonstrations, the result distribution might overfit the data, especially when a relatively complex model such as a GMM is used. In order to learn a "correct" ProMP, hence, we need a relatively large number of demonstrations, or we can only solve simple tasks, where few demonstrations are enough.

2.1.5 Dynamical System

Appart from DMPs, there is an another popular way to use the dynamical system theory to encode robot motions. These methods are called *Dynamical System* (DS) because they construct an autonomous dynamical system for the observed velocity field from human demonstrations. Considering that the robot

motions that are executed for a specific 2D task are driven by a 2D velocity field (as shown in Fig. 2.4), where each 2D position is associated with a velocity vector, a DS learns a mapping from a position to a velocity vector:

$$\dot{\boldsymbol{y}} = f(\boldsymbol{y}), \tag{2.33}$$

where $\boldsymbol{y}$ is the position and $\dot{\boldsymbol{y}}$ is its time derivative. In Gribovskaya et al. (2011), the authors used a *Gaussian Mixture Model* (GMM) to represent a generative model of the states that appear in the demonstrations. Each state is described by the position and velocity. For a demonstrated trajectory $\boldsymbol{\xi} = (\boldsymbol{y}_t)_{t=1}^{T}$, there are T pairs of samples $(x_t, \boldsymbol{y}_t)$. With M demonstrations, $M \cdot T$ samples are collected into a dataset to train the GMM. In the execution phase, for one specific position $\boldsymbol{y}$, a conditional Gaussian distribution $p(\dot{\boldsymbol{y}}|\boldsymbol{y})$ is calculated, whose mode is considered as the desired velocity at the position $\boldsymbol{y}$. The resulting function can be considered as a state dependent combination of several linear dynamical systems:

$$\dot{\boldsymbol{y}} = \sum_{k=1}^{K} \boldsymbol{\eta}_k(\boldsymbol{y})(A_k\boldsymbol{y} + b_k), \tag{2.34}$$

where $A_k = \boldsymbol{\Sigma}_{\dot{\boldsymbol{y}}\boldsymbol{y}}^k(\boldsymbol{\Sigma}_{\boldsymbol{y}\boldsymbol{y}}^k)^{-1}$ and $b_k = \boldsymbol{\mu}_{\dot{\boldsymbol{y}}}^k - A_k\boldsymbol{\mu}_{\boldsymbol{y}}^k$. K is the number of the mixture components of the GMM, which is usually regarded as a predefined constant. The weights $\boldsymbol{\eta}_k(\boldsymbol{y})$ are obtained by calculating the conditional probability $p(k|\boldsymbol{y})$. To learn the parameters of GMM $\theta = \{p(k), \boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k\}$, the *Expectation Maximization* (EM) algorithm is used. The method here is very similar to the GMM based trajectory generative model described in Section. 2.1.2. However, the mapping learned by the GMR is different. In the trajectory generative model, the learned mapping maps the temporal parameter x_t to the spatial variable $\boldsymbol{y}$. In the DS, the learned mapping maps the spatial parameter $\boldsymbol{y}$ to its corresponding velocity $\dot{\boldsymbol{y}}$.

DS with EM suffers from the stability problem due to the limited number of demonstrations that are not enough to form one asymptotically stable velocity field. The first solution mentioned in Gribovskaya et al. (2011) is to place one GMM component around the global target and augment the training data directing to the target to make sure that the trajectories converge to the target from every direction.

In Khansari-Zadeh and Billard (2011), instead of learning a GMM, the DS is obtained by solving a constraint optimization problem to maximize the likelihood of observed demonstrations with stability criteria as constraints. The DS is represented as a combination of multiple linear dynamical systems similar

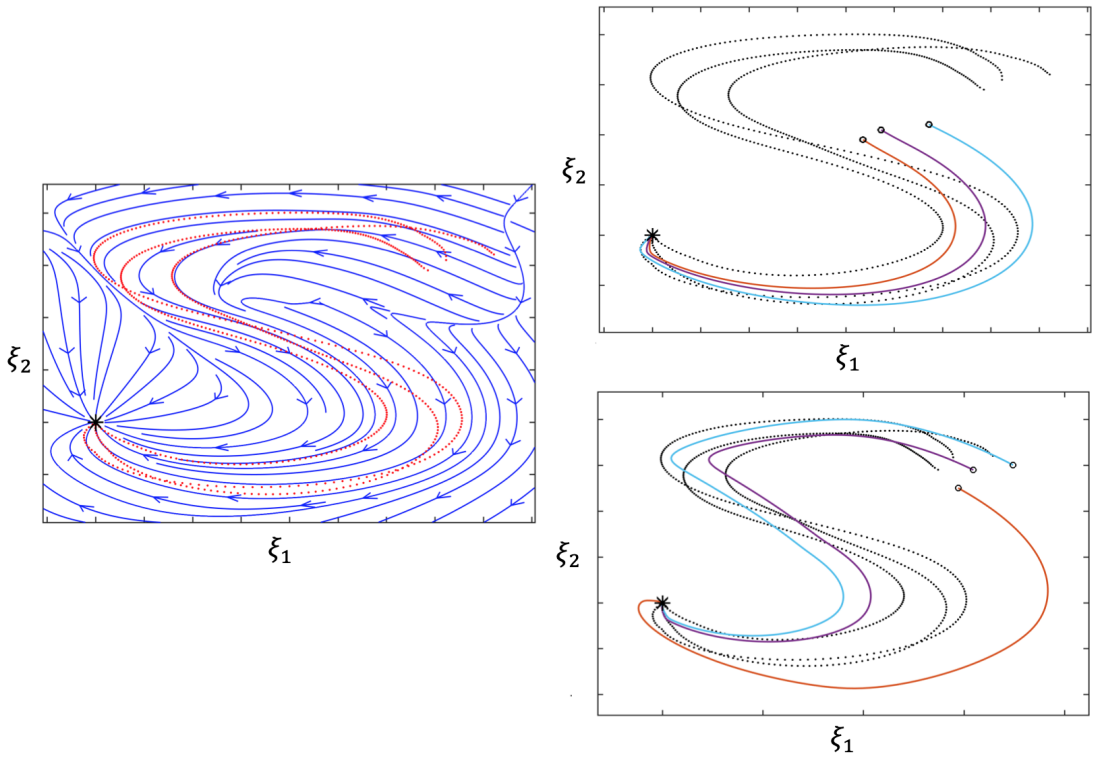

Figure 2.4: **Left**: the velocity field learned and represented by DS; **Right Top**: DS generates a trajectory directly towards the target **Right Bottom**: DS generates different trajectories with slight different starts. (The DS code is from `http://lasa.epfl.ch/sourcecode/`)

to Eq. 2.34.

As shown in Fig. 2.4, a DS is learned to draw a figure "S" in a 2D space based on the demonstrations (red dots). The velocity field represented by the learned mapping is shown with the blue curves (left diagram in Fig. 2.4). Compared to a DMP as another dynamical system approach, a DS is a global method, which means that it learns a stationary global velocity field for task-oriented motions. With a DS, the generated trajectories flow to the target from everywhere in the space. However, they do not keep the motion shape from every start point. As shown on the right side of Fig. 2.4, different start points result in different motion shapes. As mentioned before, we can regard a DMP as a trajectory generator, whose output is followed by a lower level tracking controller. In the case of a perturbation, the controller compensates it by dragging the robot back to the generated trajectory. In contrast, DS generates new trajectories with the current new start towards the global attractor after perturbation. As an example shown in Fig. 2.5, after learning a motion drawing a curve to the target, a DMP generates a backward motion when encountering a push towards the

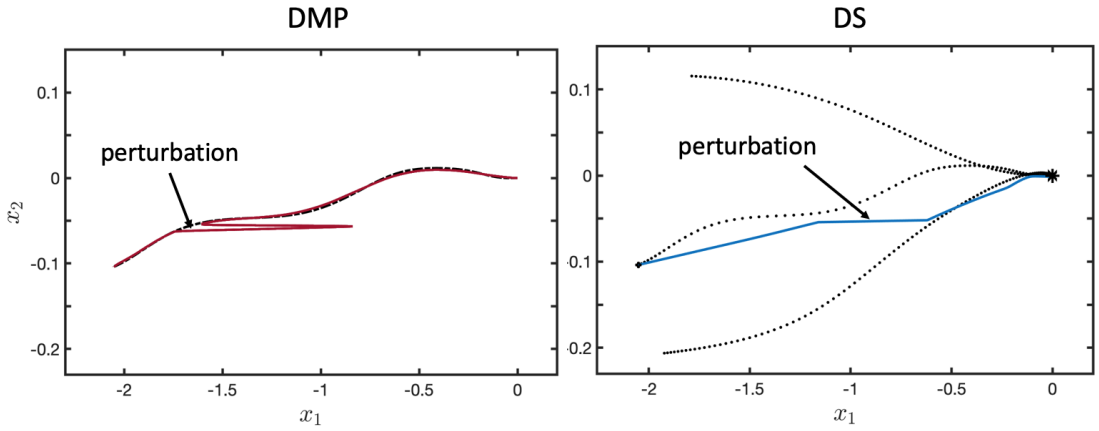

Figure 2.5: **Left**: The black dashed curve is one demonstration and the red curve is generated by the learned DMP. When encountering a perturbation, DMP stops the phase and returns to the original trajectory plan after the perturbation disappears. **Right**: The black dots denote the demonstrations and the blue curve is generated by DS. DS generates a new trajectory towards the global attracter after the perturbation.

target, while a DS continues without a backward motion, the latter behavior seems to be more intuitive.

In neuro-psychology, DS originates from the so-called "final position control" mentioned in Bizzi et al. (1984), which states that a fixed equilibrium point generates the movement, and the physical features of muscles only determine the trajectory. However, Bizzi et al. (1984) provided evidence to the conjecture that animals like monkeys generate motions based on a temporal sequence of equilibrium points, which is called "virtual trajectory." In one experiment, they showed that the deafferented monkeys return their forearms to the intermediate position after an assisting pulse in the motion direction (see Section. 2.1.5). This result coincides with DMPs.

As shown in the right bottom diagram in Fig. 2.4, a slightly different start leads to a different trajectory. This fact is because usually, a limited number of demonstrations are available for the task. Learning a "correct" DS for a specific task is barely possible. If we want to draw a figure "S," the start cannot be far from the start in the human demonstrations. DS defines the behavior with its underlying assumptions in the area that is far from the original demonstrations. Similar to ProMP, DS requires many demonstrations to be able to learn a relatively "correct" velocity field.

2.2 Motion Generalization

The movement primitives introduced in the last section have limited adaptation capabilities. For example, a DMP can adapt to the new start and goal; a ProMP can adapt to intermediate via-points which are not far from the demonstrations range. However, an intelligent robot should be able to consider any task parameters, which are not only spatial or temporal features of the trajectories, to learn how to generalize a learned skill.

In literature, researchers developed different methods for different movement primitives to allow skill generalization to different task parameters. Here, we consider two categories: learning a regression model or learning a generative model.

2.2.1 Learning Direct Mappings

These approaches are usually based on a parametric movement primitive. In the last section, except for the optimization-based methods, all other methods require a parametric function $f(\cdot)$. The meaning of this function is different for different movement primitives. In the DMP formulation, $f(x)$ is the nonlinear force term of the canonical variable x of the transformation system. In the ProMP formulation, $f(x)$ is directly the position or velocity of the trajectory at a certain time point corresponding to the canonical variable x. In the DS formulation, $f(\boldsymbol{y})$ is an autonomous dynamical system. No matter what meanings these functions have, they are associated with a parameter vector $\boldsymbol{w}$. In order to take the task parameter queries $\boldsymbol{q}$ into account, we can either replace the parameters vector $\boldsymbol{w}$ with a function of the task parameter queries $\boldsymbol{\omega}(\boldsymbol{q})$ or directly consider the task parameter queries as a part of the input of the parametric function $f(\cdot, \boldsymbol{q})$. The former methods require two regression models $f(\cdot)$ and $\boldsymbol{\omega}(\boldsymbol{q})$. While the latter ones represent the motion in one single model $f(\cdot, \boldsymbol{q})$. In Stulp et al. (2013), the authors called the former methods as two-steps methods and the latter ones as one-step methods. Here, we keep the same names.

Two-steps Methods

In both DMP and ProMP, the parametric function $f(x)$ is modeled and learned by a linear regression model such that

$$f(x) = \boldsymbol{\psi}(x)^T \boldsymbol{w}, \tag{2.35}$$

where $\boldsymbol{w}$ is the parameter vector of the movement primitive and $\boldsymbol{\psi}(\cdot)$ is the kernel vector that consists of kernel functions, usually *squared exponential kernels* (SEK), also called *radial basis functions* (RBF). The two-steps methods replace the parameter vector $\boldsymbol{w}$ with a parameter function $\boldsymbol{\omega}(\boldsymbol{q})$ of the task parameter queries as follow:

$$f(x) = \boldsymbol{\psi}(x)^T \boldsymbol{\omega}(\boldsymbol{q}). \tag{2.36}$$

The general process to learn this function is to first collect M demonstrations. For each demonstration, the parameter vector $\boldsymbol{w}$ is obtained by learning the parametric function $f(x)$. Then, a dataset $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)_{i=1}^M\}$ is collected to train $\boldsymbol{\omega}(\cdot)$, which is modeled by different regression models, such as *Locally Weighted Regression*.

In the following, we describe common approaches proposed in literature for movement primitive generalization. These are

1. **Locally Weighted Regression**

   In Ude et al. (2010), the *Locally Weighted Regression* (LWR) is used to infer the function value $\boldsymbol{\omega}(\boldsymbol{q})$ for DMP. The idea is to learn the parameter function $\boldsymbol{w} = \boldsymbol{\omega}(\boldsymbol{q})$ by minimizing the objective function for M demonstrations:

   $$\underset{\boldsymbol{w}}{\text{minimize}} \sum_{k=1}^{M} ||X_k \boldsymbol{w} - \boldsymbol{f}_k||^2 K(d(\boldsymbol{q}, \boldsymbol{q}_k)). \tag{2.37}$$

   The matrix X is called *kernel matrix* and defined as follows:

   $$X = \begin{pmatrix} \frac{\psi_1(x_1)}{\sum_{i=1}^N \psi_i(x_1)} x_1 & \cdots & \frac{\psi_N(x_1)}{\sum_{i=1}^N \psi_i(x_1)} x_1 \\ \vdots & \ddots & \vdots \\ \frac{\psi_1(x_T)}{\sum_{i=1}^N \psi_i(x_T)} x_T & \cdots & \frac{\psi_N(x_T)}{\sum_{i=1}^N \psi_i(x_T)} x_T \end{pmatrix}, \tag{2.38}$$

   where N is the number of kernels and T is the terminal time index. ψ_i is the i-th SEK such that:

   $$\psi_i(x) = exp(-\frac{1}{2\sigma_i^2}(x - c_i)^2), \tag{2.39}$$

   where c_i and σ_i are constants. In Eq. 2.37, d is a distance measure defined in the task parameter space. A kernel function K maps the distance to an appropriate weight to form a weighted sum in the output space. In Ude

et al. (2010), a tricube kernel function is used as follows:

$$K(d) = \begin{cases} (1 - |d|^3)^3, & \text{if } |d| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

This function gives almost the same weights to the data points which are close to the target but zero elsewhere. Without loss of generality, we assume that X_k is always the same in all demonstrations, which can be achieved by normalizing the trajectories in the temporal domain to a $0-1$ range. For many applications, this preprocessing makes no difference because DMP can adapt to different speed. By doing this, we can solve the optimization problem directly by calculating its derivative:

$$\boldsymbol{\omega}(\boldsymbol{q}) = \frac{\sum_{k=1}^{M} K\left(d(\boldsymbol{q}, \boldsymbol{q}_k)\right) \boldsymbol{w}_k}{\sum_{k=1}^{M} K\left(d(\boldsymbol{q}, \boldsymbol{q}_i)\right)}. \tag{2.41}$$

Strictly speaking, the method is called *Kernel Regression* (KR) which is a special type of LWR, where the local model is assumed to be a constant. In Atkeson et al. (1997), the authors claimed that KR coincides with LWR when the training data is regularly distributed and the target query is far away from the boundary of the training set. However, the general LWR outperforms KR for irregularly distributed training data. In Zhou and Asfour (2017), the LWR was used for DMP generalization and was proved to be a weighted sum with the weights corresponding to a distance measure in a higher dimensional feature space. The basic idea is to learn a new parameter vector that satisfies

$$\boldsymbol{w}_j(\boldsymbol{q}) = \frac{\sum_{i=1}^{N_q} \psi_i(\boldsymbol{q}) \boldsymbol{\eta}_{ji}}{\sum_{i=1}^{N_q} \psi_i(\boldsymbol{q})}, \tag{2.42}$$

where N_q is the number of kernels defined in the query space, $\boldsymbol{w}_j(\boldsymbol{q})$ is the j-th dimension of the resulting weights vector and $\boldsymbol{\eta}_j$ is the corresponding parameter vector. For each dimension of the MP parameter vector, we get a LWR weight vector $\boldsymbol{\eta}_j$ and have

$$\boldsymbol{w}(\boldsymbol{q}) = \frac{1}{Z} \boldsymbol{H}^T \boldsymbol{\Psi}(\boldsymbol{q}), \tag{2.43}$$

where $Z = \sum_{i=1}^{N} \psi_i(\boldsymbol{q})$, and

$$\boldsymbol{\Psi}(\boldsymbol{q}) = \begin{pmatrix} \psi_1(\boldsymbol{q}) \\ \dots \\ \psi_N(\boldsymbol{q}) \end{pmatrix}, \quad \boldsymbol{H} = \begin{pmatrix} \boldsymbol{\eta}_{11} & \dots & \boldsymbol{\eta}_{N_q 1} \\ \vdots & \ddots & \vdots \\ \boldsymbol{\eta}_{12} & \dots & \boldsymbol{\eta}_{N_q N_q} \end{pmatrix},$$

where the j-th column of the matrix $\boldsymbol{H}$ is related to the j-th component of a DMP. For the j-th component of the weight vector, we minimize the squared error:

$$L_j = \sum_{k=1}^{M} \sum_{i=1}^{N_q} \psi_i(\boldsymbol{q}_k)(\boldsymbol{\eta}_{ji} - \boldsymbol{w}_j(\boldsymbol{q}_k))^2, \tag{2.44}$$

where M is the number of samples. After calculating the derivative and its zero point, we get the optimized weight component $\boldsymbol{\eta}_{ji}$:

$$\boldsymbol{\eta}_{ji} = \frac{\sum_{k=1}^{M} \psi_i(\boldsymbol{q}_k)\boldsymbol{w}_{jk}}{\sum_{k=1}^{M} \psi_i(\boldsymbol{q}_k)}, \tag{2.45}$$

where $\boldsymbol{w}_{jk} = \boldsymbol{w}_j(\boldsymbol{q}_k)$. Replacing $\boldsymbol{\eta}_{ji}$ in Eq. 2.42 results in

$$\boldsymbol{w}_j(\boldsymbol{q}) = \frac{\sum_{k=1}^{M} \sum_{i=1}^{N_q} \psi_i(\boldsymbol{q}_k)\psi_i(\boldsymbol{q})\boldsymbol{w}_{jk}}{\sum_{k=1}^{M} \sum_{i=1}^{N_q} \psi_i(\boldsymbol{q}_k)\psi_i(\boldsymbol{q})}. \tag{2.46}$$

The result is still a weighted sum of the training data. The difference is that the weights are obtained by calculating the distance in a higher dimensional feature space instead of a function of the distance defined in the ordinary query space. The resulting function is given by

$$\boldsymbol{\omega}(\boldsymbol{q}) = \frac{\sum_{k=1}^{M} k(\boldsymbol{q}_k, \boldsymbol{q})\boldsymbol{w}_k}{\sum_{k=1}^{M} k(\boldsymbol{q}_k, \boldsymbol{q})}, \tag{2.47}$$

where

$$k(\boldsymbol{q}_k, \boldsymbol{q}) = \langle \boldsymbol{\Psi}(\boldsymbol{q}_k), \boldsymbol{\Psi}(\boldsymbol{q}) \rangle . \tag{2.48}$$

For a large training dataset, a threshold is used to truncate the contributions from the training queries far from the target one.

2. **Gaussian Process Regression**

In Forte et al. (2012), the authors replaced LWR with *Gaussian Process Regression* (GPR). For each dimension of the MP parameters $\boldsymbol{w}$, a GPR is constructed such that

$$w_i(\boldsymbol{q}) \sim \mathcal{GP}\left(0, k(\boldsymbol{q}, \boldsymbol{q}')\right), \tag{2.49}$$

where $k(\boldsymbol{q}, \boldsymbol{q}')$ is the parametric covariance function. The type and the parameters of the covariance function determines the interpolation behaviour of GPR. The most popular covariance function is the SEK such that:

$$k(\boldsymbol{q}, \boldsymbol{q}') = \sigma_f^2 exp\left(-\frac{1}{2l^2}\sum_{d=1}^{n}(q_d - q_d')^2\right), \tag{2.50}$$

where $\boldsymbol{\theta} = (\sigma_f, l)$ are the hyper-parameters. n is the dimension of the queries. The result of GPR for the i-th component of the MP parameters for a specific task parameter query $\boldsymbol{q}^*$ is given by:

$$\begin{array}{rcl} \bar{w_i^*} & = & \boldsymbol{k}_*^T(K + \sigma_n^2 I)^{-1}\boldsymbol{w}_i, \\ \mathcal{V}[w_i^*] & = & k(\boldsymbol{q}^*, \boldsymbol{q}^*) - \boldsymbol{k}_*^T(K + \sigma_n^2 I)^{-1}\boldsymbol{k}_* \end{array} \tag{2.51}$$

where $\boldsymbol{w}_i$ is the vector concatenating the i-th component of the MP parameters for all demonstrations. K is the covariance matrix whose entries are the covariance between each pair of the queries in the training dataset such that

$$K = \begin{pmatrix} k(\boldsymbol{q}_1, \boldsymbol{q}_1) & k(\boldsymbol{q}_1, \boldsymbol{q}_2) & \dots & k(\boldsymbol{q}_1, \boldsymbol{q}_M) \\ \vdots & \ddots & \ddots & \vdots \\ k(\boldsymbol{q}_M, \boldsymbol{q}_1) & k(\boldsymbol{q}_M, \boldsymbol{q}_2) & \dots & k(\boldsymbol{q}_M, \boldsymbol{q}_M) \end{pmatrix} \tag{2.52}$$

And

$$\boldsymbol{k}_* = [k(\boldsymbol{q}^*, \boldsymbol{q}_1), k(\boldsymbol{q}^*, \boldsymbol{q}_2), ..., k(\boldsymbol{q}^*, \boldsymbol{q}_M)]^T. \tag{2.53}$$

In fact, GPR can be regarded as a weighted sum of the training data, namely

$$\boldsymbol{\omega}_i(\boldsymbol{q}) = \sum_{j=1}^{M} k(\boldsymbol{q}_j, \boldsymbol{q})(K + \sigma_n^2 I)^{-1}\boldsymbol{w}_i. \tag{2.54}$$

Hence, the only difference between LWR and GPR is their different assumption regarding the covariance between the target trajectory and demonstrated trajectories. LWR assumes that this covariance is entirely depen-

dent on the distance between the current query and training queries. GPR considers also the prior knowledge shown as the type and the hyperparameters of the covariance function.

For a large number of demonstrations, GPR is not ideal because the computational cost results from inverting a $M \times M$ large covariance matrix. However, in many applications, only a small number of training demonstrations are available. On the other hand, some local techniques for GPR, such as the one described in Nguyen-Tuong et al. (2009) can be used for a relatively large dataset.

3. **Deep Neural Network**

   In the rise of deep learning, *Deep Neural Network* (DNN) is an alternative to model the mapping $\boldsymbol{\omega}(\boldsymbol{q})$ of task parameter queries to MP parameters. One advantage of using DNN is that it combines powerful structures such as *Convolutional Neural Networks* to build a system that takes the raw image as inputs, and extracts latent task parameters, and outputs MP parameters in an end-to-end manner. As an example in Pahic et al. (2018), the authors used an encoder-decoder structure to learn the mapping from a raw image of a number to DMP parameters that encode the motion to draw that number.

Other regression methods can also be used to model the mapping of task parameters to MP parameters. In Da Silva et al. (2012), the authors used *Support Vector Machines* (SVM). In Matsubara et al. (2010), *Principal Component Analysis* (PCA) was first applied to capture the lower dimensional structure of MP parameters from multiple demonstrations, and GPR was used to learn the mapping from task parameters to the weights for the principle components, called *style parameters*.

One-step Methods

In Stulp et al. (2013), the authors suggest to use one single function $f(x, \boldsymbol{q})$ to directly encode the force term of a DMP. The learned model is more compact and requires fewer parameters than the two-steps methods. In Stulp et al. (2013), this function is learned with LWR and GPR. In Pervez and Lee (2018), the idea is further developed by replacing LWR or GPR with *Gaussian Mixture Regression* (GMR).

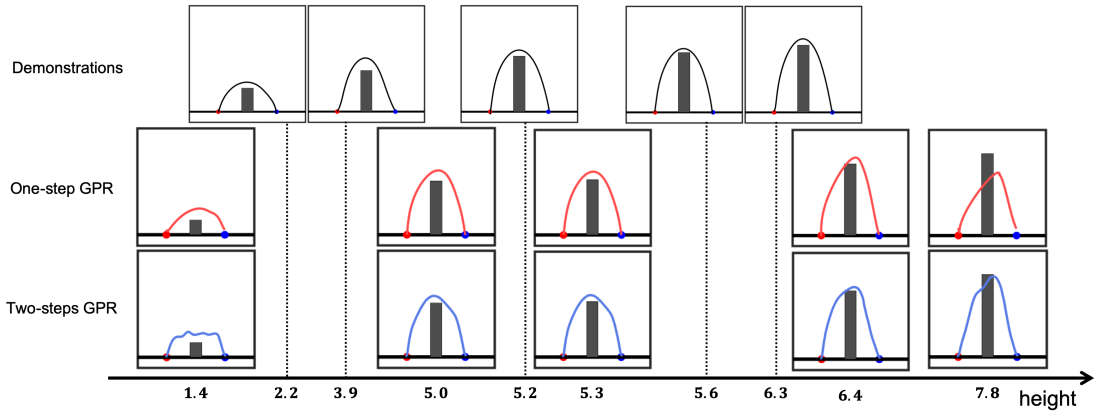

Figure 2.6: Comparison between the one-step GPR and the two-steps GPR for the tall obstacle avoidance.

Experiments for Comparing One-step and Two-steps Methods

In Fig. 2.6, an obstacle avoidance experiment is conducted, where the height of the obstacle is the only task parameter query, and the motion start (red dot) and goal (blue dot) are fixed. The first row shows the demonstrations which are collected for different heights of the obstacle. For the one-step method, GPR is used to learn the force term $f(x, \boldsymbol{q})$. For the two-step methods, GPR is used to learn the generalization mapping $\boldsymbol{\omega}(\boldsymbol{q})$, which outputs the MP parameter vector $\boldsymbol{w}$. In both cases, we use a predefined functions "fitrgp" in MatLab to model the mappings. As in Forte et al. (2012) and Stulp et al. (2013), the DMP is generalized for different heights of the obstacle. Both methods show good interpolation, which indicates that the task parameter queries are in the range of the demonstrations. However, they have only limited extrapolation, which considers the task parameter queries out of the range of demonstrations. For the height of 7.8, both methods fail to generate a collision-free solution.

In Fig. 2.7, an another experiment is conducted, where a 5-th order polynomial such that

$$\boldsymbol{\xi}(x) = \sum_{k=0}^{5} a_k x^k \tag{2.55}$$

is to be fitted. The polynomial parameters $\{a_k\}_{k=0}^{5}$ are given as the task parameter queries. The purpose is to generate trajectories which are similar to a 5-th order polynomial based on the input polynomial parameters. In this experiment, a new movement primitive formulation called *via-points movement primitive* (VMP) is used, which will be introduced in detail in Chapter. 3. VMP is developed based on DMP and ProMP. Here, we consider a special form of

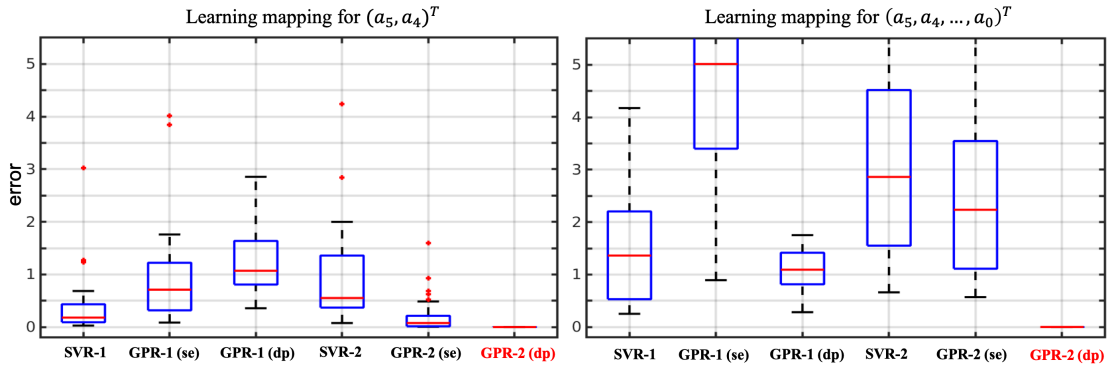

Figure 2.7: The comparison between one-step methods and two-steps methods for fitting the 5-th order polynomial.(The code is based on Pedregosa et al. (2011))

VMP, namely

$$y(x) = (y_0 - g)x + g + f(x), \tag{2.56}$$

where y_0 and g are the start and goal of the trajectory. As one-step methods, we replace $f(x)$ with $f(x, \boldsymbol{q})$. For two-steps methods, we consider $f(x) = \boldsymbol{\psi}(x)^T \boldsymbol{\omega}(\boldsymbol{q})$. Different regression models are used to model $f(x, \boldsymbol{q})$ or $\boldsymbol{\omega}(\boldsymbol{q})$ such as *Support Vector Regression* (SVR) and *Gaussian Process Regression* (GPR). For GPR, we consider two different covariance functions. The one is the *squared exponential kernel* (SEK) mentioned before as

$$k(\boldsymbol{q}_i, \boldsymbol{q}_j) = \sigma^2 exp\left(-\frac{||\boldsymbol{q}_i - \boldsymbol{q}_j||^2}{2l^2}\right). \tag{2.57}$$

The other one is the *dot product kernel* (DPK) such that

$$k(\boldsymbol{q}, \boldsymbol{q}') = \boldsymbol{q} \cdot \boldsymbol{q}' + \sigma_0^2, \tag{2.58}$$

where σ_0 controls the inhomogenity of the kernel (Pedregosa et al. (2011)). We use "se" and "dp" to denote the SEK and the DPK separately. All regression models are implemented based on the machine learning package "scikit". We use "-1" and "-2" to denote the one-step and two-steps methods. In Fig. 2.7, the results of two experiments are shown. One of them is to map from a part of the polynomial coefficients $\boldsymbol{q} = (a_5, a_4)^T$ to the MP encoded trajectory. The other one is to learn the mapping for all the coefficients $\boldsymbol{q} = (a_5, a_4, a_3, a_2, a_1, a_0)^T$.

If we use two-steps methods, the DPK is the best kernels for this task, with which GPR can almost perfectly reproduce the 5-th order polynomial (see the proof in the Appendix. Section. A).

However, for one-step approaches, the GPR with the DPK is not a right choice, because the mapping $f(x, \boldsymbol{q})$ also takes the time-dependent variable x as inputs, which avoids taking advantage of a correct kernel assumption. On the other hand, however, SVR-1 outperforms SVR-2 in both cases. The exact reason behind this is not apparent and was also not discussed in Stulp et al. (2013). One possible reason is that two-step methods learn a mapping to a higher dimensional MP parameter output. If using GPR or SVR, learning this mapping is to means learning multiple regression models, each of which takes responsibility for one dimension. Hence, errors in all regression models are accumulated.

In contrast, the mapping $f(x, \boldsymbol{q})$ to be learned in one-step methods has a relatively small output dimension. Hence, GPR or SVR learns only a small set of regression models. Overall, both one-step and two-steps methods have no big difference regarding the performance.

2.2.2 Learning Generative Models

Instead of learning a direct mapping, some works learn a generative model. The basic idea is to consider the task parameter queries as a part of the demonstrations and learn a model to represent both the task parameters and the motion trajectories. In these approaches, no explicit mapping from the task parameter queries to the MP parameter is learned.

Task-Parameterized Gaussian Mixture Model

As mentioned in Section. 2.1.2, the generative trajectory model is represented by a GMM. The problem of this method is that it cannot directly generalize to the new goals and starts. In order to be able to generate appropriate motions, the local coordinate system is required to encode the demonstrated trajectories. For example, in order to generate the approaching motion for grasping, the target object is considered as the origin of a local coordinate system. The demonstrated trajectories are encoded in this coordinate. The change of the object position is equivalent to the change of the coordinate origin. Hence, the parameters of the learned GMM for the generative model of the trajectories remain the same, and the generated trajectory adapts to different locations of the target object.

In Calinon et al. (2013), the authors extended this idea to allow multiple local frames and called their method *Task-Parameterized Gaussian Mixture Model* (TP-GMM). Each frame refers to a transformation (A, b) in the global coordinate

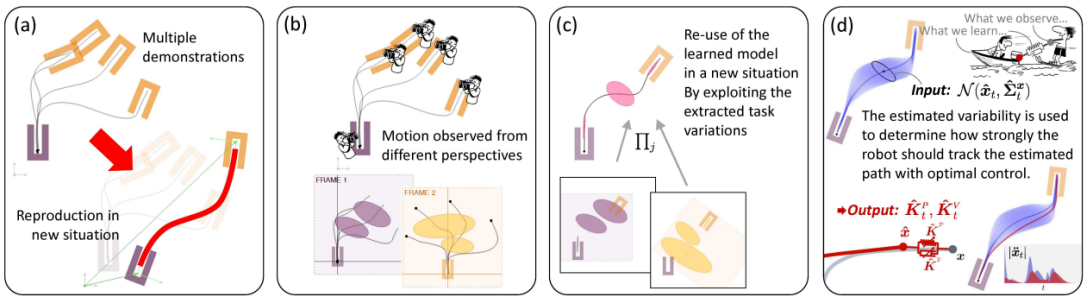

Figure 2.8: TP-GMM uses multiple perspectives and learns GMM for each frame. During the execution, these GMMs are combined to generate a trajectory which corresponds to the maximum likelihood. Then, an optimal tracking controller is used to track the generated trajectory.The picture is taken from Calinon (2016).

and provides a perspective of the demonstrated trajectories. The local GMMs are trained separately from different perspectives. During the execution, the generated trajectory maximizes the overall likelihood concerning all trained GMMs (Fig. 2.8).

The task parameters, which TP-GMM considers, are limited to those which can be represented by the transformations (A, b), thus, they are defined in the same space where GMM is defined. For example, the TP-GMM cannot solve the previous task in which a 5-th order polynomial is to be reproduced based on the polynomial coefficients. Furthermore, the local frames should be manually designed or determined according to different tasks. With inappropriate perspectives, the learned TP-GMM shows worse generalization performance. An another predefined parameter for TP-GMM is the number of mixture components of the learned GMM, which corresponds to the accuracy of the reproduction.

In Fig. 2.9, with one local frame located at the top of the tall obstacle (second row), the learned TP-GMM with 3 components generates all trajectories without collision, which, however, do not have correct starts and goals. With three local frames, which indicate the start, the goal, and the top of the tall obstacle separately, though the generated trajectories of the learned TP-GMM with 3 components have the correct starts and goals, they collide with the obstacles (third row). The reason is that the distance between two local frames remarkably changes during the execution, and the solution with the maximum likelihood still has a small probability. Increasing the number of the Gaussian components can partially solve this problem (fourth row). However, a large number of mixture components increase the learning complexity and might

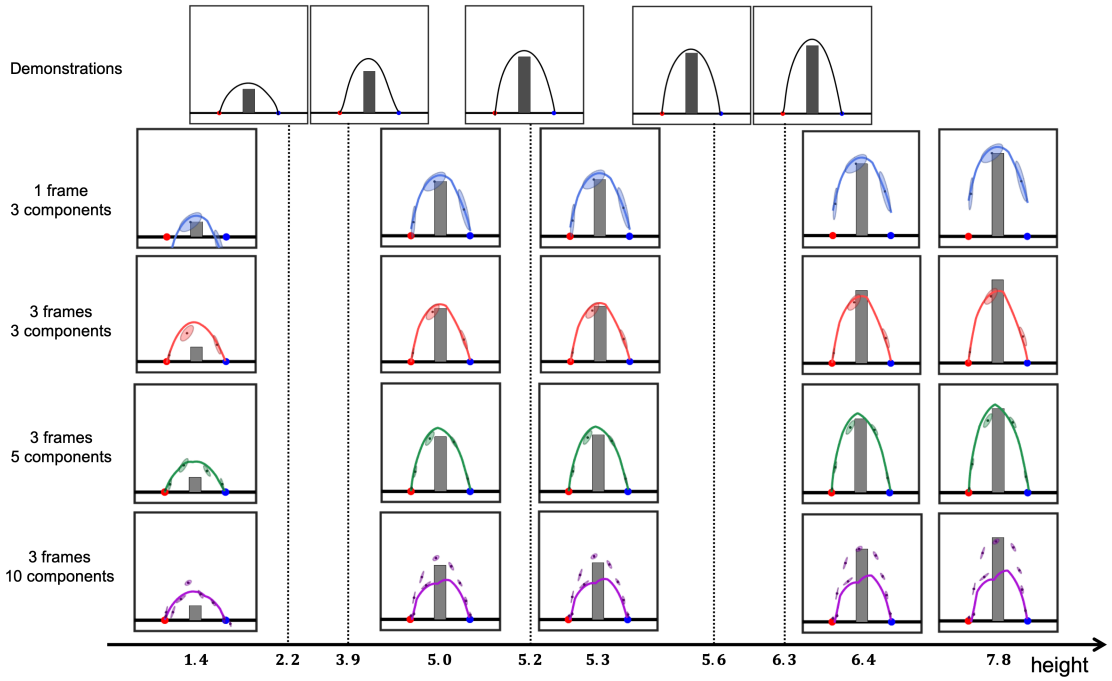

Figure 2.9: TP-GMM is used to learn the model that generates motions avoiding obstacles with different heights.

lead to overfitting of the training data and thus, result in a poor generalization performance (last row).

In Calinon et al. (2013); Calinon (2016), TP-GMM is designed to solve the extrapolation problem. Indeed, it outperforms other methods when the task parameter queries are out of the demonstrations range, as shown in Fig. 2.9. However, the extrapolation capability of the TP-GMM is not unlimited, as it cannot solve the problem where the task parameter queries are too different from those in the training data. For a successful generalization, TP-GMM requires that the user understands the problem well, and know how many local frames and where to locate local frames.

Learning Probabilistic Models

In Section. 2.1.4, the *Probabilistic Movement Primitive* (ProMP) was introduced, where we assume that the parameter vector follows a Gaussian distribution such that $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, the maximum likelihood estimation (MLE) is considered, which is the empirical mean and variance of the parameters $\{\boldsymbol{w}\}_{i=1}^{N}$ that are corresponding to N demonstrations.

In the context of the human-robot interaction, in Maeda et al. (2014), the authors

suggest to encode robot motion trajectories and human motion trajectories with one single probabilistic model. The parameters vector $\boldsymbol{w}$ can be separated by two parts $\boldsymbol{w} = (\boldsymbol{w}_o^T, \boldsymbol{w}_c^T)^T$, where $\boldsymbol{w}_o$ is the parameters for the observed agent (human) and $\boldsymbol{w}_c$ is for the controlled agent (robot). The same linear regression model used for the original ProMP represents the combined trajectory:

$$f(x) = \boldsymbol{\psi}(x)^T \boldsymbol{w}. \tag{2.59}$$

During the demonstrations, both the human motion trajectories $\boldsymbol{y}_o$ and the desired robot trajectories $\boldsymbol{y}_c$ are observed and the combined parameter $\boldsymbol{w}$ is learned by solving a least square problem for Eq. 2.59. The MLE is used to infer the parameters of the Gaussian distribution. For multiple DoFs of the robot and human, the trajectories for each dimension are concatenated leading to

$$f(x) = \boldsymbol{\Psi}(x)\boldsymbol{w}, \tag{2.60}$$

where $\boldsymbol{\Psi}(x)$ is a $(P+Q) \times (PN+QN)$ matrix with P as the number of the human DoFs and Q as the number of the robot DoFs and N as the number of the kernels.

$$\boldsymbol{\Psi}(x) = \begin{pmatrix} \boldsymbol{\psi}^T(x) & 0 & \dots & \dots & 0 \\ 0 & \boldsymbol{\psi}^T(x) & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \boldsymbol{\psi}^T(x) \end{pmatrix} \tag{2.61}$$

During the human-robot interactions, Gaussian conditioning is applied to obtain the distribution of the robot motion trajectories based on the observed human motion trajectories $\boldsymbol{y}_o$. For one specific time point that corresponds to a canonical variable x^*, the update of the Gaussian distribution is as follows:

$$\begin{aligned} \boldsymbol{\mu}^* &= \boldsymbol{\mu} + L\left(\boldsymbol{y}^* - \boldsymbol{H}^T(x^*)\boldsymbol{\mu}\right), \\ \boldsymbol{\Sigma}^* &= \boldsymbol{\Sigma} - L\boldsymbol{H}^T(x^*)\boldsymbol{\Sigma}, \\ L &= \boldsymbol{\Sigma}\boldsymbol{H}(x^*)\left(\boldsymbol{\Sigma}_{y^*} + \boldsymbol{H}^T(x^*)\boldsymbol{\Sigma}\boldsymbol{H}(x^*)\right)^{-1}, \end{aligned} \tag{2.62}$$

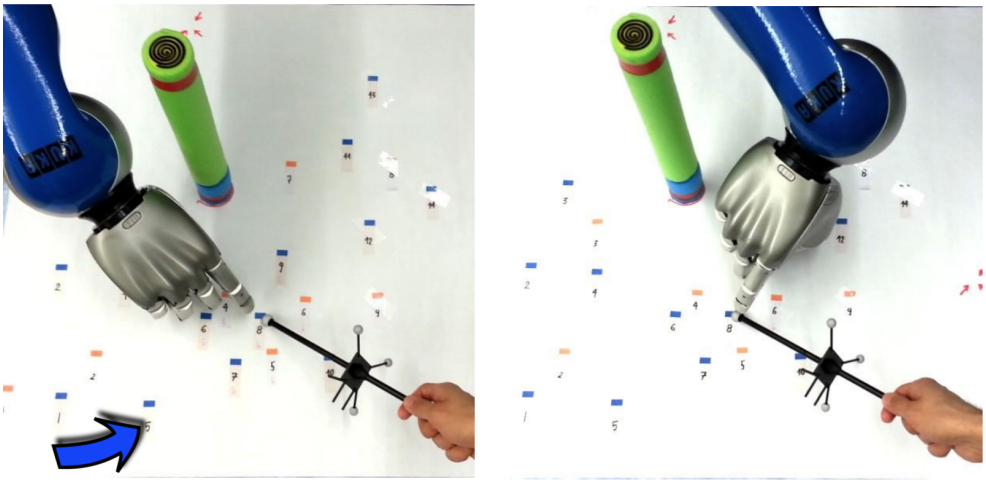

Figure 2.10: GMM based ProMP can handle multiple modes in the demonstrations. The picture is taken from Ewerton et al. (2015).

where

$$\boldsymbol{H}(x) = \left(\begin{array}{ccc:ccc} \boldsymbol{\psi}(x) & 0 & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \boldsymbol{\psi}(x) & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & \mathbf{0} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \mathbf{0} \end{array}\right) \tag{2.63}$$

is a block matrix with the upper-left submatrix having the size $PN \times P$ and the bottom-right submatrix having the size $QN \times Q$. The matrix $\boldsymbol{\Sigma}_{y^*}$ is the observation noise, which indicates the uncertainty of the observed human motions. The updated Gaussian distribution represents the desired robot trajectories distribution.

In Ewerton et al. (2015), the authors suggest to extend the Gaussian distribution to a *Gaussian Mixture Model* (GMM) such that

$$p(\boldsymbol{w}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{2.64}$$

where π_k is the mixing coefficient of the k-th component, and K is the number of the mixture components. With a fixed K, we can infer the parameters $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$ with *Expectation Maximization* (EM) algorithm. The advantage of the GMM over the Gaussian distribution is that the learned ProMP can handle multiple modes (Fig. 2.10) that exist in the human demonstrations. However,

humans can accomplish tasks with the same task parameters with different types of motion trajectories. Hence, one single Gaussian distribution cannot represent the MP parameters for all these trajectories.

The conditional probability is usually used with a learned generative model to infer the unknown or desired motion parameters from known or observed task parameters. Compared to the previous approaches which learn a direct mapping from the task parameters to the MP parameters, a generative model is more flexible and is useful for only partially observable human activities in human-robot interactions. With the conditional probability, we can infer any unknown variables based on the observed variables in the task. However, a generative model is much more challenging to learn than a direct mapping. Usually, it requires a large number of demonstrations and can only accomplish relatively simple tasks.

2.3 Motion Adaptation and Control

After generating the desired trajectory, a robot controller is needed to execute it. During motion execution, it is possible that the environment changes especially in a contact-rich manipulation, or in human-robot interaction tasks. Thus, it is necessary to adapt the generated motion to the changes in the environment. Here, we consider three types of adaptation: trajectory, force, and compliance adaptation.

2.3.1 Trajectory Adaptation

In the trajectory adaptation, the spatial or temporal features of a generated trajectory are adjusted to resolve the task constraints. In the following, we discuss three ways for trajectory adaptation.

Trajectory Adaptation based on the MP Structure

In order to adjust the trajectory, MP must have unique structures. As one example, DMPs have a goal-directed damped spring system involved. Hence, DMPs are adaptable to the new goal. The DMP parameters that can be changed for trajectory adaptation are called hyper-parameters. Thus, a DMP has three hyper-parameters: the goal g, the start y_0, and the temporal factor τ, which can be used to adapt to a new goal, a new start, or a new speed.

Recall that the DMP formulation in Pastor et al. (2009) and Hoffmann et al. (2009) (see Eq. 2.23) is given by

$$\tau \dot{v} = xK(f(x) + y_0 - y) + (1 - x)K(g - y) - Dv. \tag{2.65}$$

In Prada et al. (2013), the authors suggest to use an arbitrary function $w_g(x)$ of the canonical variable to replace the canonical variable by the weights of the two force fields such that

$$\tau \dot{v} = (1 - w_g(x))K(f(x) + y_0 - y) + w_g(x)(K(g - y) + K_v \dot{g}) - Dv, \tag{2.66}$$

where $K_v \dot{g}$ is introduced to avoid the sudden acceleration when the goal changes. With different functions w_g, the generated trajectories with the same DMP can be different. For the handover task, in the paper, this function is given by

$$w_g(x) = 0.5 \left[1 + erf(\frac{x - \boldsymbol{\mu}}{\boldsymbol{\Sigma}\sqrt{2}}) \right], \tag{2.67}$$

where erf is the Gaussian error function. By changing the parameter $\boldsymbol{\mu}$, the trajectory can be adapted to the changing goal. A larger $\boldsymbol{\mu}$ delays the response to an altered goal. The authors used this new DMP structure to accomplish the handover task.

The trajectory adaptation by changing the hyper-parameters is simple and straightforward. However, the result sees no evidence in the demonstrations. Hence, it is hard to say whether the adapted trajectory is "correct" or not.

Trajectory Adaptation with Conditional Probability

In order to generate a "correct" motion for the trajectory adaptation, researchers investigated the use of data-driven methods than altering the hyper-parameters. Similar to the generative model for the task generalization described in the last section, the conditional probability is used to adapt the trajectory based on the observed data.

As mentioned before, a ProMP encodes a trajectory distribution. After learning the distribution from the demonstrations, for a new point $(x^*, \boldsymbol{y}^*, \boldsymbol{\Sigma}_{\boldsymbol{y}^*})$ that the trajectory has to go through at time t, ProMP calculates the conditional probability of the trajectory $p(\boldsymbol{w}|(x^*, \boldsymbol{y}^*, \boldsymbol{\Sigma}_{\boldsymbol{y}^*}))$. The samples from this conditional distribution represent the trajectories going through this specific point.

In Ben Amor et al. (2014), unlike ProMP, the authors assume that the DMP

parameters for a specific task follow a Gaussian distribution. With multiple demonstrations, a Gaussian distribution of DMP parameters is obtained by calculating the mean vector and covariance matrix. The purpose is to correlate the robot motions with the partially observed human motion $\boldsymbol{\xi}_0$. The conditional problem $p(\boldsymbol{w}|\boldsymbol{\xi}_0)$ is solved for the DMP weights vector.

Solving the conditional problem provides a more "correct" trajectory than the methods which change the hyper-parameters because the former is driven by the observed demonstrations. However, this kind of method has a limited usage for trajectory adaptation. The learned distribution from the demonstrations limits the working range of the adaptation. For example, if a new goal is required, which is far away from the demonstrated trajectory in the space, ProMP will not perform well because a goal far from the range of the demonstrations is against the learned Gaussian distribution from the demonstrations.

Trajectory Adaptation for Obstacle Avoidance

One important application of the MP adaptation is to avoid obstacles during the motion execution. The online obstacle avoidance requires the change of the spatial or the temporal parameters of the trajectory. In the literature, for different MP representations, different methods were proposed for obstacle avoidance.

In Park et al. (2008), the authors added one extra repellent acceleration term $\boldsymbol{\phi}(\boldsymbol{y}, \boldsymbol{v})$ to the DMP formulation (Eq. 2.23), where $\boldsymbol{y}$ is the current state relative to the obstacle position and $\boldsymbol{v}$ is the current velocity. The acceleration term takes the form:

$$\boldsymbol{\phi}(\boldsymbol{y}, \boldsymbol{v}) = \lambda(-\cos\theta)\frac{||\boldsymbol{v}||}{p(\boldsymbol{y})}\left(\nabla_{\boldsymbol{y}}\cos\theta - \frac{\cos\theta}{p(\boldsymbol{y})}\nabla_{\boldsymbol{y}}p(\boldsymbol{y})\right), \tag{2.68}$$

where λ is a constant for the strength of the entire field. The angle θ is taken between the current velocity and the relative position: $\cos\theta = \frac{\boldsymbol{v}^T\boldsymbol{y}}{||\boldsymbol{v}||p(\boldsymbol{y})}$. The function p defines the distance between the current position and the object position. With this acceleration term, DMP is able to adjust the trajectory to avoid the obstacle.

In Huber et al. (2019), a method based on the *Dynamical System* (DS) described in Section. 2.1 is presented. The basic idea is to modulate the whole velocity field with a modulation matrix: $\dot{\boldsymbol{y}} = \boldsymbol{M}(\boldsymbol{\xi})\boldsymbol{f}(\boldsymbol{\xi})$. The authors described the ob-

stacle with a reference point $\boldsymbol{\xi}^r$ and a distance function

$$\Gamma(\boldsymbol{\xi}) = \sum_{i=1}^{d} (||\boldsymbol{\xi}_i - \boldsymbol{\xi}_i^c||/\boldsymbol{R}(\boldsymbol{\xi}))^2, \tag{2.69}$$

where $\boldsymbol{\xi}^c$ is the center of the obstacle, and $\boldsymbol{R}$ is the distance from the reference point within the obstacle to the surface in direction $r(\boldsymbol{\xi})$, which directs from the reference point to the current position $\boldsymbol{\xi}$. The gradient of this distance function denotes the direction in which the robot is getting close at the fastest to the obstacle. The method decomposes the original velocity $\boldsymbol{f}(\boldsymbol{\xi})$ into both the gradient direction and the perpendicular one. By reducing the velocity towards the obstacle and increasing the velocity in the other direction, the robot avoids the obstacles.

In Koert et al. (2019), the authors use constrained optimization methods to extract one single weights vector $\boldsymbol{w}$ from a trajectory distribution encoded by a ProMP. The optimization problem for the online spatial deformation is given as

$$\begin{aligned} \underset{\boldsymbol{w}}{\text{minimize}} \quad & (\boldsymbol{w} - \boldsymbol{\mu_w})^T diag(\boldsymbol{\Sigma_w^{-1}})(\boldsymbol{w} - \boldsymbol{\mu_w}), \\ s.t. \quad & \forall t,\ \boldsymbol{\xi}_o > \Delta(\boldsymbol{\psi}_{t-1}, \boldsymbol{\psi}_t, \boldsymbol{w}, \boldsymbol{O}_t), \\ & \boldsymbol{\xi}_w > (\boldsymbol{\psi}_t^T \boldsymbol{w} - \boldsymbol{\psi}_t^T \boldsymbol{w}_{curr})^T (\boldsymbol{\psi}_t^T \boldsymbol{w} - \boldsymbol{\psi}_t^T \boldsymbol{w}_{curr}). \end{aligned} \tag{2.70}$$

The objective function is the exponent of the Gaussian distribution learned by a ProMP. By minimizing this objective function, the probability of the trajectory is maximized with respect to the learned trajectory distribution. $\boldsymbol{\xi}_o$ denotes the bound for the minimum distance to an obstacle, $\boldsymbol{O}_t$ denotes a set of obstacles at time t. Δ denotes the minimum distance of the discretized robot trajectory to the obstacles. $\boldsymbol{\xi}_w$ limits the change of the weight vector in the current position of the trajectory. The minimum distance to one of the obstacles o is given by

$$d_{min}(\boldsymbol{\psi}_t, \boldsymbol{\psi}_{t-1}, \boldsymbol{w}, o) = \frac{|\boldsymbol{v}_1 \times \boldsymbol{v}_2|}{\boldsymbol{v}_1}, \tag{2.71}$$

where $\boldsymbol{v}_1$ denotes the trajectory velocity such that $\boldsymbol{v}_1 = \boldsymbol{\psi}_t^T \boldsymbol{w} - \boldsymbol{\psi}_{t-1}^T \boldsymbol{w}$ and $\boldsymbol{v}_2$ is the relative velocity to the obstacle: $\boldsymbol{v}_2 = \boldsymbol{\psi}_{t-1}^T \boldsymbol{w} - o$. The minimum distance to the obstacle set is then $\Delta = \underset{o \in \boldsymbol{O}_t}{min} \quad d_{min}(\boldsymbol{\psi}_t, \boldsymbol{\psi}_{t-1}, \boldsymbol{w}, o)$.

The trajectory adaptation adjusts the shape of the generated trajectory to meet some task requirements. We can also use it for the contact-rich manipulation, where we need to guarantee a target force profile. For this purpose, the MP parameters can take the offset, which generates the contact force, into the consideration.

2.3.2 Force Adaptation

In Gams et al. (2016), the authors discusse several methods to realize the force adaptation with DMPs. The purpose is to adjust the DMP parameters to accommodate the force requirements. Thus, the DMP is first initialized with the reference trajectory and then learned in an incremental manner (Eq. 2.22). Updating the DMP parameters only makes sense when a periodic task is executed, where not only the motion but also the environment does not change from one period to the other. For discrete tasks, however, the environment can hardly remain unchanged. In this case, one can directly adapt the motion to the target force without the DMP update.

One of the methods described in Gams et al. (2016) is based on velocity resolved control strategy (Villani and De Schutter (2008)), where the desired DMP position is first calculated with double integrals of the output DMP accelerations. The force controller outputs a velocity, for example, it can be $\boldsymbol{v}_f(t) = K_p(F_d(t) - F_c(t))$, where F_d is the desired force and F_c is the measured one. Thus, the desired position for the lower level position controller is

$$\boldsymbol{y}_d = \boldsymbol{y}_{dmp} + \boldsymbol{S}_F(\int \boldsymbol{v}_f(t)dt), \tag{2.72}$$

where $\boldsymbol{S}_F$ is the selection matrix which chooses the direction, in which the force control is activated. In fact, it can also be considered that DMP outputs the desired velocity with one integral of the output such that

$$\boldsymbol{y}_d = \boldsymbol{y}_0 + \int \boldsymbol{v}_{dmp}dt + \boldsymbol{S}_F(\int \boldsymbol{v}_f(t)dt), \tag{2.73}$$

where y_0 is the start position. This classical velocity resolved approach has well-known stability properties and behaviour. For a rapid changing target force profile $F_d(t)$, however, it is difficult to guarantee the high tracking accuracy because the integral causes a delay of the controller.

Given the above, the MP adaptation is reduced to a control problem. The DMP provides a desired trajectory that the controller should follow. By decoupling the motion generation and the control problem, different control strategies can be used.

Since a DMP outputs the acceleration, one other option is to add a coupling term directly to the acceleration:

$$\tau\dot{v} = K \cdot (g - y) - D \cdot v + (g - y_0) \cdot f(x) \cdot x + C(F), \tag{2.74}$$

where the coupling term can be a force proportional controller such that $C(F) = K_p(F_d - F_m)$. This is similar to the one mentioned before for the obstacle avoidance (Park et al. (2008)). In Gams et al. (2014), the authors propose a so-called coupled DMP to have both acceleration and velocity level modulations, and formulate the problem as

$$\begin{aligned} \tau\dot{v} &= K \cdot (g - y) - D \cdot v + (g - y_0) \cdot f(x) \cdot x + c_2\dot{C} \\ \tau\dot{y} &= v + C \\ C &= cF(t), \end{aligned} \tag{2.75}$$

with $F(t)$ is the force control signal, $\dot{C}$ is its derivative, c and c_2 are two scaling constants. According to the authors, the derivative $\dot{C}$ serves as a damping term like in a PD controller. The authors again used the *Iterative Learning Control* (ILC) to adjust the coupling term C. ILC can only be applied for a repeatable task. The idea of a simple ILC is to add the error term at the same time point in the last execution into the current coupling term C in a feed forward manner.

The force adaptation can also be used for the bimanual task, where a displacement of both end-effectors needs to be guaranteed. In Gams et al. (2014), the authors suggest to use two DMPs, each of which encodes the motion of one end-effector. The coupling term is then $F(t) = k(d_d - d_a)$, where d_d is the desired distance between the end-effectors and d_a is the actual difference.

2.3.3 Movement Primitive Control and Compliance Adaptation

As mentioned before, MP adaptation can be realized by separating the trajectory generation from the design of a tracking controller. Even if a coupling term is added into the original DMP formulation for adaptation, we still need a tracking controller to track the desired position or velocity trajectories.

For the tracking controller, an intuitive choice is impedance control, where the attractor of a virtual damped spring system is moved with the desired trajectory. With inverse dynamics based on the robot model, the desired torque $\boldsymbol{\tau}_d$ for each joint is obtained. If the learned MP encodes task space trajectories, the desired torque is calculated such as

$$\boldsymbol{\tau}_d = J^T(\boldsymbol{q})\boldsymbol{F}_m, \tag{2.76}$$

where $\boldsymbol{F}_m$ is the external force obtained by the admittance control, and $\boldsymbol{J}$ is the

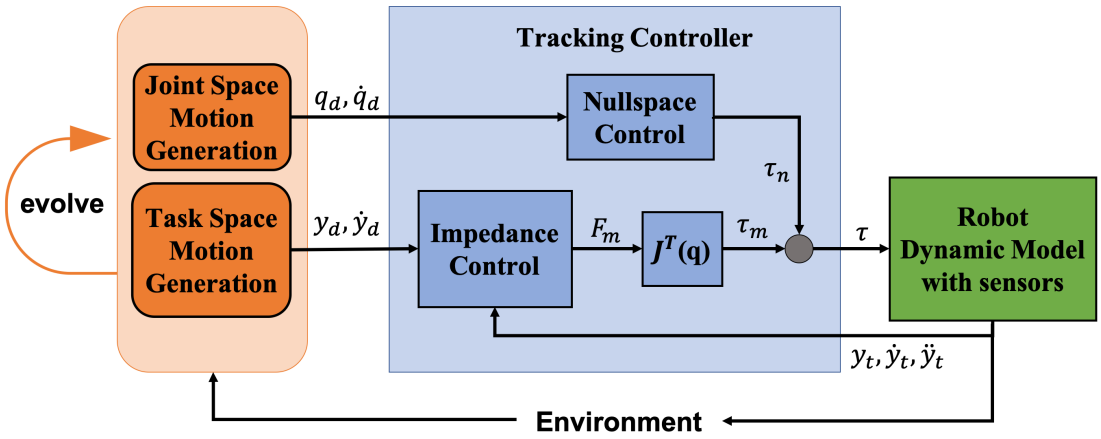

Figure 2.11: The control framework for the task space motion generation, developed and used in this thesis.

Jacobian matrix for the current joint values $\boldsymbol{q}$.

In many robot applications, we prefer encoding task space trajectories to joint space trajectories because it is more intuitive and convenient to take the task requirements, which are usually defined in the task space, into consideration. If the robot learns from the observation of human demonstrations, it is also much easier to learn a task space trajectory, i.e., an end-effector trajectory in the Cartesian space, because learning joint space trajectories requires solving a non-trivial retargeting problem. If the robot learns by kinesthetic teaching, both task and joint space trajectories can be easily learned. The generated joint space motions can be used for the null space control to realize more natural robot motions for the task. In Fig. 3.5, a control framework for the task space MP is shown. Many experiments in this work were implemented based on this control framework.

Passivity based Controller

In Kronander and Billard (2016) and Kramberger et al. (2018), passivity based control strategies are developed for DS and DMP respectively. The idea of passivity based control is to design an energy storage function based on sensory feedback signals. The system is passive if the time derivative of the storage function is smaller than the supplied power. For a robot system, the input can be considered as the generalized force $\boldsymbol{F}$, and the output is the velocities of the robot $\dot{\boldsymbol{y}}$. If the task space is considered, the generalized force can be the real external force exerted upon the robot's end-effector, and the velocities are the

task space end-effectors' velocities. For the joint space, the generalized force consists of the force exerted on each motor, and the velocities are joints' velocities. The supplied power is represented by $\boldsymbol{F}^T\dot{\boldsymbol{y}}$.

In Kronander and Billard (2016), the authors separate the learned velocity field into a conservative and a non-conservative part

$$\boldsymbol{f}(\boldsymbol{y}) = \boldsymbol{f}_C(\boldsymbol{y}) + \boldsymbol{f}_R(\boldsymbol{y}), \tag{2.77}$$

where a single attractor generates the conservative part. The non-conservative part is the offset between the learned and the conservative velocity field. The target velocity for the controller is determined by combining these two velocity fields with a specific weight

$$\boldsymbol{\tau}_c = -\boldsymbol{D}\dot{\boldsymbol{y}} + \lambda_1 \boldsymbol{f}_c(\boldsymbol{y}) + \beta_R(z,s)\lambda_1 \boldsymbol{f}_R(\boldsymbol{y}), \tag{2.78}$$

where $\boldsymbol{D}$ is the damping coefficient, λ_1 is a constant and $\beta_R(z,s)$ is the specific weight function determined by the virtual energy s and the power supply of the non-conservative part $z = \dot{\boldsymbol{y}}\boldsymbol{f}_R(\boldsymbol{y})$. At the beginning of the motion, the initial stored energy is $\bar{s}$. The energy flow is controlled by

$$\dot{s} = \alpha(s) \cdot \dot{\boldsymbol{y}} \cdot \boldsymbol{D} \cdot \dot{\boldsymbol{y}} - \beta_s(z,s)\lambda_1 z. \tag{2.79}$$

Both α and β_s should satisfy some conditions (Kronander and Billard (2016)) to guarantee that the energy amount is fixed and is distributed between the robot and the virtual energy storage s. For example, if the current velocity is quite different from the non-conservative velocity field, the virtual energy is increasing. According to the design of those functions, β_R should be large to guarantee the tracking accuracy.

In Kramberger et al. (2018), a passivity observer is applied to the coupled DMP. The input power of the coupled DMP with a task space impedance controller in interaction with a passive environment is defined as

$$P_{in} = \dot{\boldsymbol{c}}^T \boldsymbol{F}_d - \dot{\boldsymbol{y}}_{d,0}^T \boldsymbol{F}_{ext}, \tag{2.80}$$

where $\boldsymbol{c}$ is the coupling term given by $\boldsymbol{y}_d - \boldsymbol{y}_{d,0}$ and $\boldsymbol{y}_{d,0}$ is the output of DMP. $\boldsymbol{F}_d$ is the desired force and $\boldsymbol{F}_{ext}$ is the actual force. The desired power profile of the reference system is

$$P_{in}^* = \dot{\boldsymbol{y}}_{d,0}^T \boldsymbol{F}_d. \tag{2.81}$$

The active power $P_{act} = P_{in} - P_{in}^*$ is used to iteratively adjust the DMP's goal

to guarantee the passvity of the system.

Learning Compliant Control

In Deniša et al. (2016), the authors proposed the *Compliant Movement Primitive* (CMP). Instead of only representing the spatial trajectories like a DMP, a CMP also encodes the torque profile. Hence, a CMP can be desired by a tuple $\{\boldsymbol{w}_q, \boldsymbol{g}_q, \boldsymbol{w}_\tau, v_k\}$. $\boldsymbol{w}_q$ is the weights vector to represent the force term of the DMP formulation for the trajectories in the state space. $\boldsymbol{w}_\tau$ is the weights vector of a linear regression model to represent the torque profile concerning the canonical variable x such that

$$\tau(x) = \boldsymbol{\psi}(x)^T \boldsymbol{w}_\tau, \tag{2.82}$$

where $\boldsymbol{\psi}(x)$ is the kernel function. v_k is the time duration of the whole motion, and $\boldsymbol{g}_q$ is the hyper-parameters of the DMP state. The first weights vector $\boldsymbol{w}_q$ can be easily learned by observing the state-space trajectories. $\boldsymbol{w}_\tau$ is learned by observing the torques from the torque sensors when executing the motion once with a PD controller with high stiffness. With the learned torque profile, the designed controller can utilize the feedforward control signal to reduce the gains of the feedback control part. By doing this, it generates compliant motions. However, with the learned torque profile, the proposed method is difficult to generalize to different situations. For example, the goal adaptation of the original DMP cannot be used anymore. In order to solve this problem, the authors suggest learning a mapping from the task parameters $\boldsymbol{c}$ to $\boldsymbol{w}_\tau$ using multiple demonstrations.

2.4 Conclusion

In this chapter, we follow the structure of the thesis mentioned in Chapter. 1 to introduce the previous methods in the literature for motion representation, generalization, adaptation and control.

Motion Representation

In this work, we concentrate on parametric models. Compared to those non-parametric models, a parametric model simplifies motion generalization and adaptation. Inspired by the works in neuro-psychology such as Bizzi et al.

(1984, 1991), researchers in robotics society are searching for mathematical formulations, so-called *movement primitive*. Movement primitives can be combined or connected for a complex task.

Dynamic Movement Primitive (DMP), introduced in Schaal (2003) and Ijspeert et al. (2013), consists of a damped spring system and a non-linear force term. Due to its structure, a DMP can adapt to new goals by changing the attractor of the damped spring system. However, a DMP cannot adapt to any intermediate via-points that are required to fulfill particular task requirements.

Probabilistic Movement Primitive (ProMP), described in Paraschos et al. (2013), learns a distribution of weights vectors of a linear regression model, which is used to represent trajectories. With the operation in probability, a ProMP can adapt to any via-points including starts and goals. However, with a small number of demonstrations, a ProMP cannot learn a "correct" distribution.

Dynamical System (DS), introduced in Gribovskaya et al. (2011), learns directly an autonomous dynamical system, hence, is independent of time. When encountering perturbations, a DS behaves different than a DMP or ProMP. With enough demonstrations, a DS gives "correct" motions that start with arbitrary points. However, a DS cannot reproduce a trajectory in an accurate way. When current position is not covered by demonstrations, a DS generates a velocity directing to the goal.

In Chapter. 3, these methods are further compared regarding via-points adaptation. Since via-points are useful for many robotic applications, via-points adaptation is an important feature of movement primitives. Then, a new movement primitive called *Via-points Movement Primitive* (VMP) is introduced, which inherits the advantages of DMP and ProMP and resolves their drawbacks.

Motion Generalization

In this section, two kinds of generalization methods are introduced: learning a direct mapping or learning a generative model.

According to Stulp et al. (2013), for learning a direct mapping, we further divide approaches into one-step and two-steps methods. In one-step methods, only one regression model is learned to map a canonical variable and task parameters to a required variable that is different for different movement primitives. For example, in a DMP, the required variable is the non-linear force term. In a ProMP, it is directly a point in the trajectory. In two-steps methods, besides a parametric function mapping a canonical variable to a required variable, we

learn an another mapping from task parameters to the parameter of this parametric function. Several experiments show that both one-step and two-steps methods have no big difference regarding their performance.

For learning a generative model, several methods are introduced.

Task-Parameterized Gaussian Mixture Model (TP-GMM), introduced in Calinon et al. (2013), considers learning motions in the local frames and execute the most likely motion in the global frame. With the idea of local frames, TP-GMM outperforms direct mappings with respect to the extrapolation. However, TP-GMM requires the knowledge about how many local frames are necessary and where they should be located. It cannot be used for arbitrary task parameters and can only work for those task parameters that can be represented by local frames.

Gaussian Mixture Model based Probabilistic Movement Primitive (GMM-ProMP), introduced in Ewerton et al. (2015), learns motions and the corresponding task parameters together and uses conditional probability to infer an appropriate motion based on a given task parameter. The proposed method can be used for human-robot interaction tasks.

However, learning a generative model requires more data than learning directly mappings. With a small number of demonstrations, learning those models can result in overfitting or local optimum, thus, affect the generalization.

In this work, we consider learning a direct mapping. However, the previous regression models used for the mapping do not consider multiple modes and models that exist in human demonstrations. The mode collapse leads to the lose of motion diversity. The model collapse causes the average problem and leads to failure of task execution. To solve these problems, we propose two different approaches in Chapter. 4. One approach is to use a mixture of experts. We suggest using *Leave-One-Out Expectation Maximization* (LOO-EM) to learn these expert models when only a small number of demonstrations are available. The other approach is to use *Mixture Density Network* (MDN) to model a mapping from task parameters to the distribution of movement primitive parameters. We suggest using new costs to avoid mode and model collapses during training MDN.

Motion Adaptation and Control

In this section, different adaptation approaches are introduced for particular tasks such as obstacle avoidance, handover and force control. The motion

adaptation requires expert knowledge. For the controller, we describe different strategies to realize compliant motion execution in literature.

In this work, we develop a leader-follower framework to solve cooperative and human-robot interaction tasks. We mainly focus on developing a robotic wiping system, where robot's wiping motions should adapt to the human movement. Furthermore, a force predictive model based adaptive controller is developed to realize compliant motion execution. Unlike previous approaches, we consider learning a force predictive model and detect abnormal force profiles during the task execution. Based on the current force profile, an adaptive controller decides how to adjust the stiffness of all PID controllers involved to fulfill a task.

3 Movement Primitive Representation

As a mathematical model for analyzing biological movements, movement primitives facilitates intuitive robot programming by demonstrations. A good MP representation with strong adaptation capabilities would allow flexible and compact representation of robot motions as it allows changing the trajectory shape to accomplish multiple tasks.

In this chapter, we first compare the adaptation capabilities of different MPs introduced in Chapter. 2. Then, we introduce a new formulation called *Via-points Movement Primitives* (VMP). The VMP formulation is developed based on two previous works: DMP and ProMP. It resolves their drawbacks and inherits their benefits. Finally, we evaluate the VMP performance in different applications and create a new framework for the robot learning from human demonstrations to allow the robot to learn from the failure.

The work described in this chapter has been published in Zhou et al. (2019).

3.1 Adaptation Capability of Existing Movement Primitives

Here, three different methods described in Chapter. 2 are considered. They are the *Dynamical System* (DS) approach, *Dynamic Movement Primitives* (DMP) and *Probabilistic Movement Primitives* (ProMP). In the following, we discuss their adaptation capabilities.

Dynamical System

As a global method, DS adapts to a new goal by translating and rotating the whole velocity field. In Gribovskaya et al. (2011), a comparison between DMP and DS shows that the motion generated by DMP for adapting to a new goal

during the execution may have an unexpectedly excessive curvature. The reason is that a discontinuous change of the goal attractor causes a sudden acceleration.

With the phase stopping technique, DMP pushes the robot "back" to the "original" trajectory. In order to resolve this sudden acceleration, an another P-controller can be used to change the DMP goal to the new target gradually.

On the other hand, DS remembers what the human demonstrator does at some near points and generates the motion directly to the target. If human demonstrations cover the whole workspace, and the goal is the only task constraint, DS generates more correct motions than DMP does when the goal is changing, or perturbations exist. Unfortunately, human demonstrations can hardly cover the whole workspace because humans are only willing to provide a minimal number of demonstrations and similarly accomplish many tasks, even though there are multiple demonstrations. It is not always realistic to ask a human to demonstrate a task with different initial states, and it is tedious to design the data collection process. If no training data exists around some points in the state space, most of DS variants generate velocities based on stability constraints. There is no reason to claim that the motion generated by a DS around the point where there is no training data is better than scaling the demonstrated trajectory with a DMP. Furthermore, DMPs are much simpler than DS concerning the number of parameters. In order to acquire the correct number of mixture models, the training process in the case of DS is more complicated than DMP. Furthermore, DS cannot adapt to any intermediate via-points between starts and goals because it only allows one global attractor.

Dynamic Movement Primitive

A DMP introduced with Eq. 2.12 in Chapter. 2 is a nonlinear dynamical system and assumes that the motion is governed by a damped spring system with a non-linear force term. In the *bio-inspired DMP* mentioned in Chapter. 2 (see Eq. 2.23), two elastic force fields are used, which is inspired by the frog experiment in Bizzi et al. (1984). We can reformulate Eq. 2.23 as follows:

$$\tau \dot{v} = K \cdot (g - (g - y_0) \cdot x + x \cdot f(x) - y) - D \cdot v, \tag{3.1}$$

which is a PD tracking controller that tracks a moving attractor, consisting of a straight line trajectory with a constant velocity profile defined in the phase space and a non-linear trajectory $x \cdot f(x)$ that supports reproducing the human

demonstrations.

A DMP learns the offset of the demonstrated trajectory from the straight line connecting the start and the goal directly. By learning only this offset, DMPs can adapt to a new goal with a straight line connecting to the goal. This learned offset guarantees a similar trajectory shape by the goal adaptation in the temporal domain. For multi-dimensional trajectories, DMPs treat each dimension separately. Hence, the similarity in the temporal domain does not necessarily result in the similarity in the spatial domain. For example, imagine that a DMP reproduces a U-shaped parabola. If we rotate the goal to a new position with 90^o, a DMP generates a counter-intuitive trajectory that might overshoot the target. In Paraschos et al. (2013) and Gribovskaya et al. (2011), the authors claimed that their methods resolve this drawback of DMP.

In Dragan et al. (2015), the authors related a DMP with a Hilbert space trajectory optimization problem that minimizes the acceleration difference between the generated trajectory for new goals and the demonstrated one. The line trajectory with a constant velocity profile in the virtual trajectory is the only changing part for a new goal. The difference between two line trajectories is still a line trajectory with a constant velocity profile, which has zero acceleration everywhere. Hence, it corresponds to the result of the optimization problem.

However, like DS, DMPs cannot adapt to any intermediate via-points because DMPs learn the virtual trajectory. In neuro-psychology, Gomi and Kawato found that the virtual trajectory is more complex than the real one, especially when the stiffness is low (Gomi and Kawato (1997)). Thus, it is difficult to manipulate the real trajectory by adjusting the virtual trajectory.

The question is whether it is necessary to represent the virtual trajectory, especially for robot applications. To answer the question, we absorb x into the non-linear function $f(x)$ (see Eq. 3.1) and split it into two parts:

$$\tau \dot{v} = K \cdot (g - (g - y_0) \cdot x + f_1(x) - y) - Dv + f_2(x), \tag{3.2}$$

where $g - (g - y_0) \cdot x + f_1(x)$ represents the real trajectory that corresponds to the demonstrated one, and $f_2(x)$ is an additional acceleration term that guarantees accurate tracking of the real trajectory. It means that DMPs also learn the compensation to the error of a PD tracking controller to accurately track the real trajectory. In robot applications, however, DMP is usually not directly used to control the robot. The reason is that the learned extra acceleration $f_2(x)$ is good for tracking the demonstrated trajectory, but cannot guarantee the accuracy when the goal is changed. The most used strategy is to choose a high

stiffness K to reduce the amount of $f_2(x)$. The higher is the stiffness, the closer is the learned virtual trajectory $g - (g - y_0) \cdot x + f(x)$ to the real trajectory. Nevertheless, a high gain PD controller is not ideal for compliant robot control. Hence, another lower-level tracking controller with a relatively lower gain is necessary. Thus, a reasonable strategy is to separate the tracking control from DMPs.

Different methods guarantee the tracking accuracy in a compliant way, such as *Iterative Learning Control* (ILC). Hence, an alternative strategy is to use MP only to represent the real trajectory and use a tracking controller to realize compliant control.

Probabilistic Movement Primitive

A ProMP assumes that the task-oriented motion trajectories follow a Gaussian distribution (see Section. 2.1.4). ProMP separates the motion representation from the robot control. The assumption of Gaussian distribution allows it to handle via-points in an intuitive way. A ProMP considers a via-point as an observed data point and draws a trajectory from a conditional Gaussian distribution (see Eq. 2.32).

Learning ProMPs is to infer the parameters of the Gaussian distribution of the parameters vector $\boldsymbol{w}$. The same argument for DS also applies to ProMPs. The human demonstrations can hardly cover the whole workspace. The learned Gaussian distribution is only correct in an area close to the demonstrations. This erroneous prior probability affects the performance of ProMP to extrapolate for the via-points outside of the demonstrations region, where the probability of the via-points based on learned probability is so small to cause conflicts.

Fig. 3.1 shows one example, where the learned ProMP converges to one specific goal with relatively low variance. A slight displacement of the goal results in a dramatic change in the generated trajectory. In contrast, the DMP performs reasonably.

The behavior of these MPs in the area where no training data points exist is highly dependent on their assumptions. A DS assumes a stable velocity field that governs the motion. Hence, it approaches the goal directly when no data exists. DMP assumes that the virtual trajectory consists of a goal-directed line and a non-linear offset. Thus, it scales the trajectory and preserves the shape in the temporal domain for a new target. ProMP, instead, assumes a Gaussian trajectory distribution. Hence, it does not allow any trajectory that goes through the region with very low probabilities.

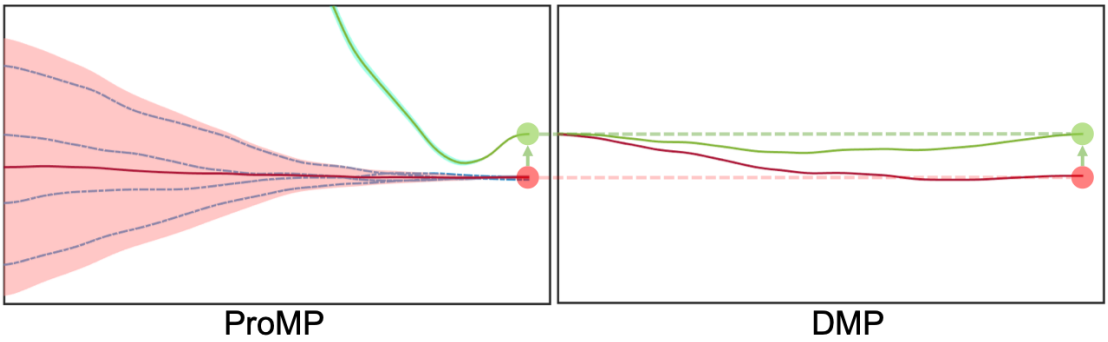

Figure 3.1: To generalize the reaching motion to a new target, ProMP generates an abnormal motion (green curve) because the learned prior probability of the trajectory distribution (red region) has a low variance when approaching the goal. A slightly different target (from the red ball to the green one) causes numerical problems when calculating the conditional probability with Eq. 2.32. In contrast, DMP can naturally generalize to the new goal because a goal directed elastic force field is one of its two components (see Eq. 2.23).

For goal adaptation, a DS translates and rotates the velocity field to match the attractor to the goal. If data is not correctly collected or not enough to cover the whole workspace, the translation and rotation might expose many points to the non-data area, where a DS generates velocities based on its stability criterion. A DMP draws a line from the start to the new goal and keeps the learned offset from this line. If this line is rotated too much in the spatial domain, a DMP might generate non-intuitive trajectories because it treats each dimension separately. ProMP calculates the conditional probability based on the learned prior probability of the skill-associated trajectories. It can adapt well to the new goal that is around the demonstrated trajectories. However, a goal out of the region will end up with an infeasible motion for the robot.

One might argue that fully data-driven methods such as DS and ProMP generate relatively correct trajectories because they obey the human demonstrations. For a task requirement regarding the area which is not covered by the training data, we should instead apply an additional learning process or require more human demonstrations.

However, as mentioned before, a human demonstrator is probably not willing to provide a large number of demonstrations. A versatile learning system based on MP should be able to generate reasonable motions for unseen cases. Furthermore, it is difficult to apply fully data-driven methods in practical applications. In order to acquire a skill, the robot needs to observe more than two demon-

	DS	DMP	ProMP	VMP
probabilistic formulation	✓	✗	✓	✓
free translation	✗	✓	✗	✓
start/goal interpolation	✗	✓	✓	✓
start/goal extrapolation	✗	✓	✗	✓
intermediate via-point interpolation	✗	✗	✓	✓
intermediate via-point extrapolation	✗	✗	✗	✓

Table 3.1: A feature list of movement primitives

strations for these methods. The demonstrations should be also under control or carefully designed. In contrast, a DMP learns from only one demonstration and can adapt automatically to the target. As far as the target is not too different from the original one, a DMP generates reasonable motions. Although there is no proof that its generated trajectories are correct, they meet a large number of practical applications. However, a DMP cannot learn from multiple demonstrations, which also limits its application, especially when there are multiple demonstrations available. Several methods extended DMP to a probabilistic model (Matsubara et al. (2010) Meier and Schaal (2016)). These methods demonstrate an improvement of motion representation using DMPs, but they do not improve their adaptation capabilities.

In this chapter, we concentrate on the via-points adaptation and propose a new formulation for movement primitives, the *Via-points Movement Primitive* (VMP) that inherits the advantages of ProMP and DMP and resolves their drawbacks. In Table 3.1, we show a feature list of these movement primitives. Except DMPs, all other MPs take probabilistic formulations. Free translation means that one MP can encode all trajectories in the space that differ from each other with a constant translation (Dragan et al. (2015)). Only DMPs and VMPs have free translations. DMPs cannot adapt to the via-points except the start and goal. ProMPs cannot handle extrapolation. As DMPs, DSs cannot handle intermediate via-points. However, DSs can adapt to a new goal by shifting the learned velocity field. In this case, however, we actually change the parameters of the DS. Strictly speaking, hence, the shifted DS is not the original one. VMPs, introduced in this chapter, have all desired features regarding adaptation capability as a motion representation.

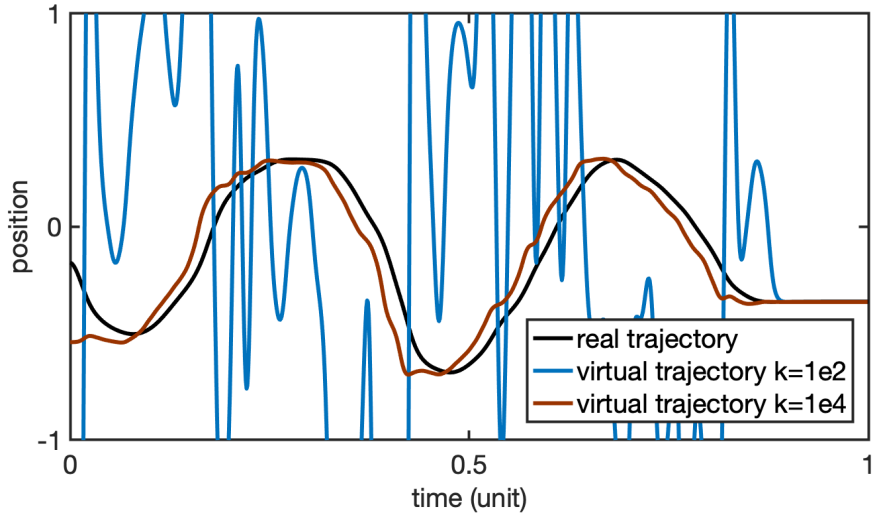

Figure 3.2: The virtual trajectory is more complicated than the real trajectory. Only when the stiffness of the PD controller given by DMP is high enough, the virtual trajectory gets closer to the real trajectory

3.2 Via-Points Movement Primitive (VMP)

3.2.1 Basic Formulation

The formulation of VMP consists of two different parts: elementary trajectory $\boldsymbol{h}$ and shape modulation $\boldsymbol{f}$:

$$\boldsymbol{y}(x) = \boldsymbol{h}(x) + \boldsymbol{f}(x). \tag{3.3}$$

The elementary trajectory $h(x)$ connects directly the start and the goal of the motion. The shape modulation $f(x)$ describes the offset of the real trajectory to the elementrary one. This offset is parameterized by the parameter vector $\boldsymbol{w}$. As DMP and ProMP, we use a linear weighted regression model with *squared exponential kernels* to represent the shape modulation:

$$f(x) = \boldsymbol{\psi}(x)^T \boldsymbol{w}, \quad \boldsymbol{\psi}_i(x) = exp(-\frac{1}{2\sigma_i^2}(x - c_i)^2). \tag{3.4}$$

It is not difficult to find the similarity between a VMP and a DMP because a DMP uses the same structure to represent the virtual trajectory tracked by a PD controller. Further, the elementary trajectory $h(x)$ in a DMP is a line trajectory with a constant velocity profile. Instead of encoding the virtual trajectory, however, a VMP represents directly the real trajectory. The virtual trajectory

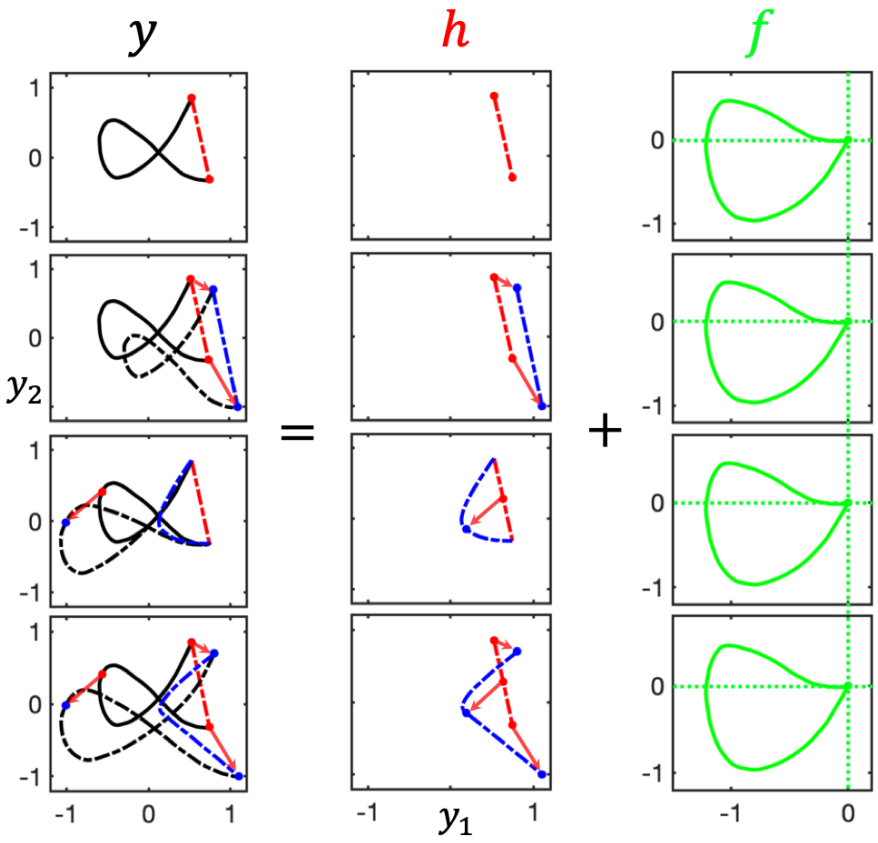

Figure 3.3: Via-point modulation by changing the elementary trajectory h: The motion for drawing "α" in a 2D plane is divided into two parts: elementary trajectory h and shape modulation f. **Left:** The VMP is learned from only one demonstration. **Middle:** The via-points modulation is realized by manipulating the elementary trajectory h. **Right:** In this process, the learned force modulation is not changed.

is similar to the real one only when the stiffness K of the DMP is very high (Fig. 3.2).

Because of directly encoding the real trajectory, a VMP allows direct trajectory manipulations such as via-points integration. As an example depicted in Fig. 3.3, the motion that draws the letter "α" is separated into the linear trajectory h, and the shape modulation f. The parametric shape modulation contains the features of the motion. In Fig. 3.3, we also show how to manipulate the trajectory by only changing the elementary part of a VMP. The result of the start and goal adaptation of a VMP shown in the second line is the same as a result given by a DMP. However, a VMP allows the intermediate via-points adaptation, which is challenging with a DMP.

Unlike a DMP with a deterministic parameter vector $\boldsymbol{w}$, a VMP assumes that $\boldsymbol{w}$

follows a Gaussian distribution:

$$\boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}}\right).$$

With this assumption, a VMP allows the calculation of the conditional probability as a ProMP does. Hence, a VMP is a combination of DMP and ProMP. The probabilistic formulation of a VMP is as follows:

$$\boldsymbol{y}(x) \sim \mathcal{N}\left(\boldsymbol{h}(x) + \boldsymbol{\psi}(x)^T \boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\psi}(x)^T \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\psi}(x) + \boldsymbol{\Sigma}_f\right). \tag{3.5}$$

Learning a VMP is to infer the parameters of the Gaussian distribution. The *maximum likelihood estimation* (MLE) is the emprical mean and variance:

$$\boldsymbol{\mu}_{\boldsymbol{w}} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{w}_i, \quad \boldsymbol{\Sigma}_{\boldsymbol{w}} = \frac{1}{M}\sum_{i=1}^{M} (\boldsymbol{w}_i - \boldsymbol{\mu}_{\boldsymbol{w}})(\boldsymbol{w}_i - \boldsymbol{\mu}_{\boldsymbol{w}})^T, \tag{3.6}$$

where M is the number of demonstrations and the i-th parameter vector is obtained by solving a least square problem such that

$$\boldsymbol{w}_i = \left(\boldsymbol{\Psi}^T \boldsymbol{\Psi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Psi}^T (\boldsymbol{Y}_i - \boldsymbol{H}_i), \tag{3.7}$$

where $\boldsymbol{Y}_i$ represents the i-th demonstrated trajectory and $\boldsymbol{H}_i$ encodes its corresponding elementary trajectory and $\boldsymbol{\Psi}$ is given by

$$\boldsymbol{\Psi} = \begin{pmatrix} \boldsymbol{\psi}^T(x_0) \\ \boldsymbol{\psi}^T(x_1) \\ \vdots \\ \boldsymbol{\psi}^T(x_N) \end{pmatrix}, \tag{3.8}$$

with N as the number of the kernels.

As DMPs, VMPs assume that a goal-oriented basic structure always exists. As ProMPs, VMPs also suppose that the trajectories follow a Gaussian distribution. Based on these two assumptions, VMPs can adapt to the via-points in two different ways: either by manipulating the elementary trajectory $\boldsymbol{h}$ or by adjusting the shape modulation $\boldsymbol{f}$.

3.2.2 Elementary Trajectory

We use a triple $(x^*, \boldsymbol{y}^*, \boldsymbol{\Sigma}^*)$ to represent a via-point, where x^* is the canonical value that specifies when the via-points that should be incorporated into the

trajectory and $\boldsymbol{\Sigma}^*$ shows the confidence level of the via-point. If the via-point should be definitely met, $\boldsymbol{\Sigma}^* = 0$.

Here, we concentrate on how to use the elementary trajectory to realize via-points adaptation. Without loss of generality, we assume that we train MPs on only one demonstration. The learned shape modulation $f(x)$ of a VMP with zero variance is a deterministic trajectory. Note that ProMPs cannot adapt to any via-points because calculating the conditional probability based on the prior probability with zero variance $\boldsymbol{\Sigma}_{\boldsymbol{w}} = 0$ causes the numerical problem, especially when the via-point should be definitely met, namely $\boldsymbol{\Sigma}^* = 0$ in Eq. 2.32.

Inspired by the virtual trajectory of DMPs, the elementary trajectory of VMPs $\boldsymbol{h}$ directly connects the start and the goal with a straight line. It also makes sense to let the robot generate a line trajectory directing to the target when there is no demonstration. For the goal adaptation, the elementary trajectory is a new line from the start to the new goal.

The basic idea for the intermediate via-points adaptation is to split the elementary trajectory into segments, where each segment corresponds to a real trajectory that connects two adjacent via-points, including the start and the goal. This straight line connects two adjacent virtual via-points $(x^*, \boldsymbol{h}^*)$ with

$$\boldsymbol{h}^* = \boldsymbol{y}^* - \boldsymbol{f}(x^*). \tag{3.9}$$

Even though the elementary trajectory is a line trajectory connecting two points, it can be associated with different velocity profiles. Based on these profiles, they behave differently. Since we define all trajectories in the canonical variable domain, the shape of the trajectories changes significantly in the temporal domain if the canonical system is not linear.

In order to keep the same shapes in both the canonical variable domain and temporal domain, we use a linear decay canonical system instead of an exponential decay system, which is popular for DMPs. The canonical system is normalized and goes from 1 to 0. For each trajectory, once the start and the goal are defined, we have two predetermined via-points $(1, y_0)$ and $(0, g)$. Hence, the shape modulation $\boldsymbol{f}(1) = \boldsymbol{f}(0) = 0$. We can also learn the start and goal with the shape modulation and use a constant zero as the elementary trajectory during the training. In this case, learning a VMP is the same as learning a ProMP.

Without loss of generality, we let the shape modulation only model the shape of the trajectory but not learn the start and goal like what DMP assumes. For

simplicity, all the formulations mentioned below are given for one dimension. It is trivial to extend them for multiple dimensions.

Line Trajectory with a Constant Velocity

A simple line function connecting two points (x_0, h_0) and (x_1, h_1) is:

$$h(x) = (h_1 - h_0) \cdot \frac{x - x_0}{x_1 - x_0} + h_0, \tag{3.10}$$

where $1 = x_0 > x_1 = 0$ because of a linear decay canonical system. If no via-points exist, $h_0 = y_0$ and $h_1 = g$. Then, a VMP can be represented as

$$\begin{aligned} y(x) &= h(x) + f(x) \\ &= (g - y_0) \cdot \tfrac{x-1}{0-1} + y_0 + f(x) \\ &= (y_0 - g) \cdot x + g + f(x), \end{aligned}$$

which is exactly the formulation of the virtual trajectory of a DMP (see Eq. 3.1). In this case, for the goal adaptation, both VMP and DMP generate the same trajectory. When an intermediate via-point is required, a VMP splits the elementary trajectory into segments. These segments should meet three requirements:

$$h(0) = g, \quad h(1) = y_0, \quad h(x^*) = y^* - f(x^*) \tag{3.11}$$

With a constant velocity profile, a piece-wise linear function such as

$$h(x) = \begin{cases} -\frac{(h^* - y_0)x}{1 - x^*} + y_0 + \frac{(h^* - y_0)}{1 - x^*} & x \leq x^* \\ \\ -\frac{(g - h^*)x}{x^*} + g & x > x^*, \end{cases} \tag{3.12}$$

where $h^* = y^* - f(x^*)$ meets the requirements. It is straightforward to extend this method for more than one via-points. Theoretically, it allows as many via-points as required. In contrast, as mentioned before, a DMP cannot handle intermediate via-points because it encodes more complex virtual trajectories which are difficult to adjust in a meaningful way.

The problem of the linear trajectory with a constant velocity profile is that it introduces a turning point to the elementary trajectory and leads to a sudden acceleration around any via-points. If no training data exists, a VMP automatically generates a line trajectory with a constant velocity profile towards the target, which has a sudden acceleration at both start and goal points. Although

we can resolve this sudden acceleration with some lower-level controllers, we still prefer generating a smooth trajectory in many robot applications.

Optimized Trajectory

As mentioned in Chapter. 2, in Dragan et al. (2015), the authors proved that DMP minimizes a Hilbert space norm to adapt the trajectory to the new goal

$$\begin{aligned} \underset{\boldsymbol{\xi}}{\text{minimize}} \quad & ||\boldsymbol{\xi} - \boldsymbol{\xi}_D||_A^2 \\ s.t. \quad & \boldsymbol{\xi}(1) = y_0, \boldsymbol{\xi}(T) = g_{new}, \end{aligned} \tag{3.13}$$

where $\boldsymbol{\xi}$ refers to a trajectory vector, and $\boldsymbol{\xi}_D$ is the demonstrated one. A is a Hilbert space norm $A = K^T K$ with K being the finite differencing matrix. T is the length of the trajectory that corresponds to the time duration of the motion. The proof consists of two parts. First, it is not difficult to see that the gradient of the objective function is the acceleration profile of the trajectory. The optimization requires that the gradient equals to zero, which means that the trajectory has zero acceleration, namely constant speed. If the force term is equal to zero, a straight line with a constant speed is the adaptation result of the optimization. Second, it is provable that the force derivative should be the same if it is not zero in the demonstrated trajectory in order to minimize the objective function. In Dragan et al. (2015), the authors prove the claim for the virtual trajectory of DMP. The proof is the same for the goal and intermediate via-points adaptation for a VMP.

For one intermediate via-point, the following optimization problem is to be solved:

$$\begin{aligned} \underset{\boldsymbol{\xi}}{\text{minimize}} \quad & ||\boldsymbol{\xi} - \boldsymbol{\xi}_D||_A^2 \\ s.t. \quad & \boldsymbol{\xi}(1) = y_0, \quad \boldsymbol{\xi}(T) = g_{new}, \\ & \boldsymbol{\xi}(i^*) = y^*, \end{aligned} \tag{3.14}$$

where i^* is the index for the via-point and corresponds to the via-point canonical value x^*, because we split the trajectory into the elementary trajectory and the shape modulation. Once it is learned, the shape modulation will not change. This common part of the trajectory can be eliminated. Instead of the index, the canonical value is used and we get:

$$\begin{aligned} \underset{h}{\text{minimize}} \quad & \tfrac{1}{2}(\boldsymbol{H}_D - \boldsymbol{H})^T A(\boldsymbol{H}_D - \boldsymbol{H}) \\ s.t. \quad & h(1) = y_0, \quad h(0) = g, \\ & h(x^*) = y^* - f(x^*), \end{aligned} \tag{3.15}$$

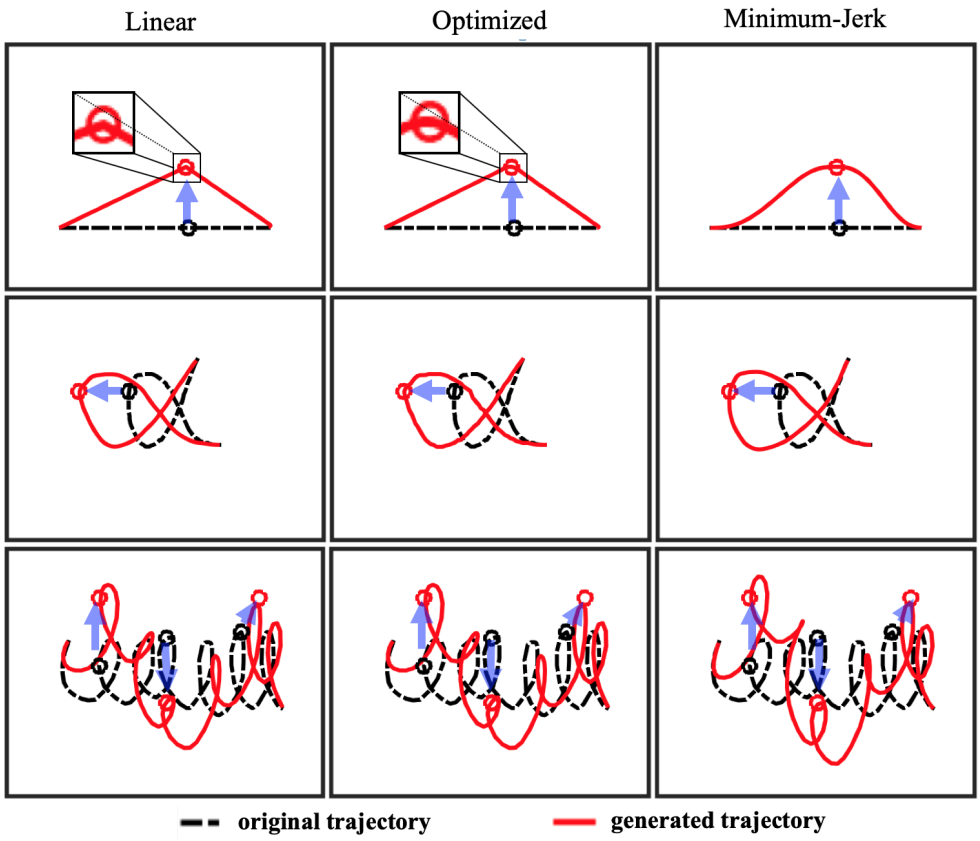

Figure 3.4: The black dotted lines are the original trajectories, and the red lines are the trajectories which adapt to the via-points (red circles) that have the same canonical values as the original points (black circles) in the demonstrations. The linear trajectory segments with a constant velocity generate a sudden acceleration at the via-point. This problem can be solved by the optimization defined in Eq. 3.14 or the minimum-jerk trajectory segments defined in Eq. 3.17 solves the problem. The minimum-jerk trajectories generate much smoother trajectory because the velocity at the via-point is matched to the one on the original trajectory.

where $\boldsymbol{H}_D$ and $\boldsymbol{H}$ are the elementary trajectories in human demonstrations and during the execution. By solving this optimization problem, we get a trajectory which is similar to the one with a piece-wise linear function. The only difference is that the turning point has a smooth transition due to the fact that a discontinuous velocity profile leads to big costs regarding the norm A cost function (see Fig. 3.4).

Minimum-Jerk Trajectory

In neuro-psychology, researchers conducted experiments to explore the patterns followed by animals and human during motion execution. In Hogans (1984) and Flash and Hogan (1985), it has been shown that human reaching movements minimize the third time derivative of the position trajectory. In many robot applications, the motion is generated based on this optimization problem:

$$\underset{\boldsymbol{Y}}{\text{minimize}} \int_{t=0}^{T} \dddot{y}(t)^2 dt, \tag{3.16}$$

This kind of trajectories is called minimum-jerk trajectories because they minimize the third derivative of the location called jerk. With the calculus of variations (see Hogans (1984)), it has been shown that these minimum-jerk trajectories have zero sixth derivative. This fact gives a possibility to model the minimum-jerk trajectory with a fifth-order polynomial

$$h(x) = \sum_{k=0}^{5} a_k x^k, \tag{3.17}$$

whose six unknown parameters a_k can be obtained by the boundary conditions at both adjacent via-points (x_0, y_0) and (x_1, y_1):

$$\begin{array}{lll} h(x_0) = y_0 - f(x_0) & \dot{h}(x_0) = \dot{y}_0 - \dot{f}(x_0) & \ddot{h}(x_0) = \ddot{y}_0 - \ddot{f}(x_0) \\ h(x_1) = y_1 - f(x_1) & \dot{h}(x_1) = \dot{y}_1 - \dot{f}(x_1) & \ddot{h}(x_1) = \ddot{y}_1 - \ddot{f}(x_1) \end{array} \tag{3.18}$$

Because the shape modulation is a differentiable function, we can also specify the velocity and acceleration at the via-points with this elementary trajectory segment modeled by a fifth order polynomial. By matching the velocity at the via-point with the one on the original trajectory, the fifth order polynomial generates smoother and more similar trajectories than the constant speed line trajectory.

In Fig. 3.4, three different elementary trajectories are compared. The optimized elementary trajectory based on the norm A is very similar to the constant speed line trajectory, with only a smoother transition around the via-points. The minimum-jerk elementary trajectory generates even smoother trajectories. In many robot applications, we prefer the minimum-jerk elementary trajectory. Because it always generates smooth trajectories even when no training data exists.

3.2.3 Via-Points Modulation

Manipulating the elementary trajectory by segmenting it to parts and connecting each pair of via-points with an individual segment is not the only way for the via-points adaptation with VMPs. We can still calculate the conditional probability based on the probability distribution of the shape modulation like what a ProMP does. In this case, the elementary trajectory is not changed, but the shape modulation is adjusted and adapted to the new via-points. For a via-point $(x^*, \boldsymbol{y}^*, \boldsymbol{\Sigma}^*)$, the equations to adjust the parameters of the shape modulation distribution are similar to the ones used for ProMP (see Eq. 2.32):

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{w}}^* &= \boldsymbol{\mu}_{\boldsymbol{w}} + L\left(\boldsymbol{y}^* - \boldsymbol{h}(x^*) - \boldsymbol{\psi}(x_{via})^T \boldsymbol{\mu}_{\boldsymbol{w}}\right), \\ \boldsymbol{\Sigma}_{\boldsymbol{w}}^* &= \boldsymbol{\Sigma}_{\boldsymbol{w}} - L\boldsymbol{\psi}(x^*)^T \boldsymbol{\Sigma}_{\boldsymbol{w}}, \\ L &= \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\psi}(x^*)\left(\boldsymbol{\Sigma}^* + \boldsymbol{\psi}(x^*)^T \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\psi}(x^*)\right)^{-1}. \end{aligned} \tag{3.19}$$

To select one of the two ways to insert the via-points: changing the elementary trajectory or the shape modulation, we calculate the probability of the via-point conditioning on the learned distribution of the parameter vector:

$$p\left((x^*, \boldsymbol{y}^*, \boldsymbol{\Sigma}^*) | \boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}}\right) = \mathcal{N}\left(\boldsymbol{y}^*; \boldsymbol{h}(x^*) + \boldsymbol{\psi}(x^*)^T \boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\psi}(x^*)^T \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\psi}(x^*) + \boldsymbol{\Sigma}^*\right). \tag{3.20}$$

If this probability is bigger than a given threshold η, the via-point is considered inside the learned distribution and the trajectory passes through it by manipulating the shape modulation distribution. If the via-point probability is smaller than the threshold, it is considered outside the region of demonstrations and is passed through by adjusting the elementary trajectory as mentioned in the previous section. By doing this, VMPs combine the strategies of DMPs and ProMPs and thus is more robust for the via-points adaptation.

3.2.4 Orientation VMP

In order to use VMPs to represent task space trajectories in robot applications, encoding the position trajectories is not enough. In the next section, we propose a method which also encodes the orientation trajectory.

The elementary trajectory of a VMP serves as a local coordinate system where the shape modulation generates the offset trajectory. The superposition of these

two parts is a trajectory defined in the global coordinate system. However, the superposition of two orientation trajectories is less meaningful and usually makes no sense in robot applications. Hence, in order to be able to encode the orientation trajectory, we use quaternions to represent the orientation and the quaternion multiplication to replace the superposition for the position trajectories. The formulation of VMP for the orientation trajectory is as follows:

$$y(x) = h(x) * f(x), \tag{3.21}$$

where y, h and f are three quaternion functions defined on the canonical value domain. $h(x)$ denotes the orientation elementary trajectory that consists of segments connecting two adjacent orientation via-points. For the position trajectory, the elementary trajectory segment is a straight line. For the orientation trajectory, this segment is the shortest curve connecting two quaternion. The spherical linear interpolation (SLERP) provides the curve:

$$h(x) = \frac{\sin((1-\alpha(x))\beta)}{\sin(\beta)} q_0 + \frac{\sin(\alpha(x)\beta)}{\sin(\beta)} q_1, \tag{3.22}$$

where $cos(\beta) = q_0 \cdot q_1$ is the dot product of two quaternions. q_0 and q_1 are calculated by

$$q_0 = y_0 * f(x_0)^{-1}, \quad q_1 = y_1 * f(x_1)^{-1}. \tag{3.23}$$

(x_0, y_0) and (x_1, y_1) are two adjacent orientation via-points and $\alpha(x)$ is the parameter of the SLERP function which determines the velocity profile of the generated orientation trajecory. For a constant speed, we have

$$\alpha(x) = \frac{x - x_0}{x_1 - x_0}.$$

For a bell-shaped velocity profile which is like the one of a minimum-jerk position trajectory, we can also use a fifth-order polynomial such that

$$\alpha(x) = \sum_{k=0}^{5} a_k \left(\frac{x - x_0}{x_1 - x_0} \right)^k,$$

where the parameters a_k are obtained by the boundary conditions:

$$\begin{array}{lll} \alpha(x_0) = 0 & \dot{\alpha}(x_0) = \dot{h}_0 / c_0 & \ddot{\alpha}(x_0) = (\ddot{h}_0 - l_0) * \dot{\alpha}(x_0) / \dot{h}_0 \\ \alpha(x_1) = 1 & \dot{\alpha}(x_1) = \dot{h}_1 / c_1 & \ddot{\alpha}(x_1) = (\ddot{h}_1 - l_1) * \dot{\alpha}(x_1) / \dot{h}_1 \end{array} \tag{3.24}$$

with

$$c_0 = -\beta\cot(\beta)q_0 + \beta\csc(\beta)q_1 \quad c_1 = -\beta\csc(\beta)q_0 + \beta\cot(\beta)q_1$$

and

$$l_0 = -\beta^2 q_0 \quad l_1 = -\beta^2 q_1.$$

$\dot{h}$ and $\ddot{h}$ are the velocity and acceleration of the elementary trajectory at the orientation via-point, which can be calculated based on the required velocity and acceleration of the real trajectory and the shape modulation. In robot applications, the acceleration of the minimum-jerk orientation trajectory is usually set zero to simplify the calculation.

For the probabilistic formulation of the quaternion trajectory, one simple way is to use three linear model to represent the trajectories for q_x, q_y, and q_z separately. We calculate the component q_w with the fact that a unit quaternion is required. The start quaternion determines the initial value of q_w, and its value is continuously tracked to guarantee that there is no jump because of the antipodally symmetric property. Based on the assumption that the components are independent and identically distributed (i.i.d.), we end up with the following quaternion distribution:

$$y(x) \sim \mathcal{N}\Big(h(x)*\psi(x)^T\left[q_w, \boldsymbol{\mu}_{q_x}, \boldsymbol{\mu}_{q_y}, \boldsymbol{\mu}_{q_z}\right], \boldsymbol{\Psi}(x)^T diag(\boldsymbol{\Sigma}_{q_w}, \boldsymbol{\Sigma}_{q_x}, \boldsymbol{\Sigma}_{q_y}, \boldsymbol{\Sigma}_{q_z})\boldsymbol{\Psi}(x) + \boldsymbol{\Sigma}_f\Big), \tag{3.25}$$

where $diag(.)$ and $\boldsymbol{\Psi}(x)$ are block diagonal matrices. This is a multivariate normal distribution with independent components, and the covariance matrix is diagonal. With a multivariate Gaussian distribution, we can use the same strategy as mentioned before for the position trajectory to calculate the conditional probability, and orientation via-points are added by manipulating the distribution (see Eq. 3.19). In reality, it is not necessary to manipulate the shape modulation for the orientation via-points adaptation, because it is usually not accurate and more complicated than directly manipulating the elementary trajectory.

3.2.5 Task Space VMP

By combining the position and orientation trajectory representation, a complete VMP is used to represent a task space trajectory or its distribution. As mentioned in Chapter. 2, MP usually serves as a kinematic planner. The robot needs some controllers to follow the MP trajectory to accomplish the tasks. As

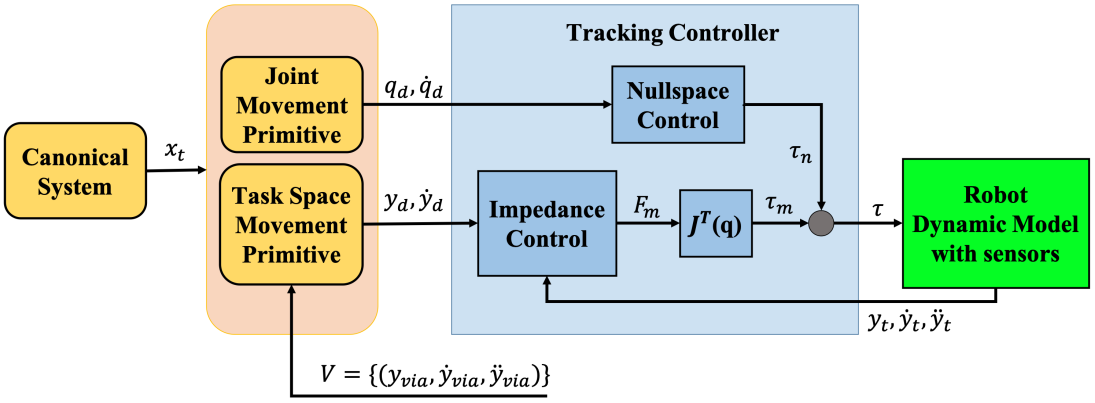

Figure 3.5: The control framework for VMP is shown with the block diagrams. The joint space and task space trajectories are separately learned from the kinesthetic demonstration.

mentioned before, we follow the basic idea and separate the trajectory representation from the design of the tracking controller that is dependent on the individual robot model. For example, an MP can be used to represent either the TCP trajectories for a robot arm or the trajectories of the platform of a mobile robot. These robots have different models and hence require different control strategies.

The advanced control methods such as iterative learning control or reinforcement learning control can also be applied to allow the robot to follow the given trajectories with compliant behaviors.

The simplest way to control a robot to follow a trajectory is a PD controller. We consider first to draw a sample from the trajectory distribution and let the PD controller track the sample trajectory. For the via-points added before the execution, it is straightforward. For the online via-points, we re-calculate the trajectory distribution conditioning on the required via-points and draw another sample from the new distribution. In order to handle perturbations, we consider the current position and velocity as an online via-point and generate the rest part of the trajectory based on it.

The Fig. 3.5 depicts a VMP control framework for tracking task space trajectories. The output of the joint space VMP provides the target for the null space control to guarantee that the robot does not generate any counter-intuitive joint motions when tracking the required task space trajectory.

For the control targets, since VMP is differentiable, it can provide both target positions y_d and velocities $\dot{y}_d$. Like DMPs, VMPs can also adapt to different

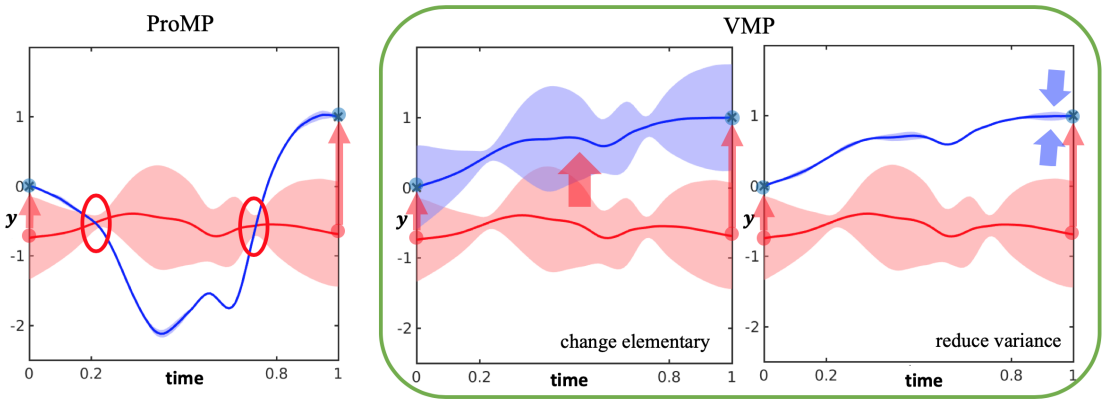

Figure 3.6: **Left:** ProMP generates trajectories going through low variance region (red circle). **Right:** VMP adapts to the goal with two steps. Firstly it changes elementary trajectories (see the middle plot) and then reduces the variance (see the most right plot).

speeds by adjusting the temporal factor in the canonical system such as a linear decay canonical system:

$$\tau \dot{x} = -\alpha_x.$$

With changed temporal factors, the target velocities should be divided by the temporal factor such as $\frac{\dot{y}_d}{\tau}$.

3.2.6 Comparison of VMPs and ProMPs

Because DMPs cannot adapt to the intermediate via-points, we concentrate on the comparison between VMPs and ProMPs.

We first consider a simple example where we want to adapt an MP to a new goal, which is out of the range of demonstrations, which means that the conditional probability (see Eq. 3.20) is less than the threshold η (see Fig. 3.6). In this case, if we use a constant speed line trajectory, DMPs give the same result as the mean trajectory of VMPs.

A VMP accomplish this task with two steps. First, it adapts the elementary trajectory to the target because the value obtained by Eq. 3.20 is less than the threshold. Second, the VMP reduces the variance by adjusting the variance of the shape modulation using Eq. 3.19. The second step is to guarantee that the generated trajectory will hit the target. These two steps also work for the intermediate via-points that are beyond the range of demonstrations. In contrast, ProMPs stick to the low variance region and generates a trajectory that passes

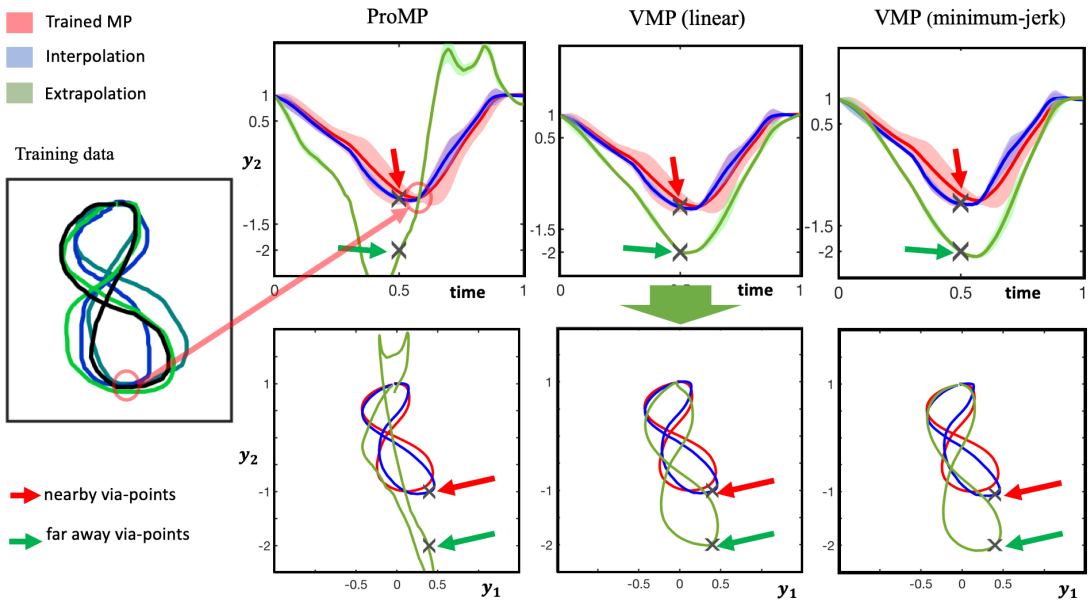

Figure 3.7: Comparison between VMP (with $\eta = 0.8$) and ProMP for drawing a figure eight. We consider three cases: without via-points (red curves), with a via-point within the observation range (blue curves) and with a via-point outside the range (green curves). The **top-right** plots show the vertical axis values along the time line. The **bottom-right** plots show the result motions. Different elementary trajectory segments generate slightly different trajectories.

through these regions. This leads to infeasible motions and a large change in the demonstrated trajectories.

In Fig. 3.7, we compare further the behavior of VMPs with ProMPs and train them to draw a figure "8". Two different intermediate via-points are asked during two motion executions. For the via-point that is inside the learned trajectory distribution where $p(y_{via}|\mu_w, \Sigma_w) > \eta$, they have similar behaviour (blue curves). For the via-point that is outside the learned trajectory distribution, namely $p(y_{via}|\mu_w, \Sigma_w) < \eta$, VMP manipulates the elementary trajectory $h(x)$ and generates a scaled figure eight that goes through the via-point (green curves). In contrast, ProMP fails to generate any reasonable motions (most left-bottom figure).

In the top row of the Fig. 3.7, we depict the trajectories for the vertical dimension in the time domain. It can be seen that the ProMP sticks to the region with low variance. When carefully inspecting the training data, it is easy to find that the region of low variance does not have any special meaning. It corresponds to a point at the bottom of figure "8" that is accidentally passed through

by all demonstrations. This accident leads to the erroneous prior probability learned in the ProMP. It is inevitable to produce an erroneous prior probability because the number of demonstrations for real applications is always limited, and the human demonstrations cannot cover all possible situations. In this case, VMPs assume that there is a goal-directed trajectory that stabilizes the generated motion as DMPs do. Although it is still difficult to guarantee the correctness, VMPs reduce the risk of generating infeasible motions when compared to ProMPs.

Theoretically, the low variance region corresponds to task constraints (such as obstacles in Fig. 2.3) during the human demonstrations. However, as mentioned before, it is usually difficult for a human to give multiple demonstrations in a way that is appropriate for training a "correct" probabilistic model. Furthermore, the human demonstrations are sometimes similar, which results in a set of dummy regions with low variance (as shown in Fig. 3.7).

For hard task constraints, it is better to set via-points which avoid the conflicts to the constraints than to let the robot learn these constraints from the demonstrations. In the case of multiple task constraints, a ProMP probably generates an even more infeasible motion to meet all of them.

The threshold of the conditional probability to determine whether to manipulate the elementary trajectory or the shape modulation is not easy to decide, because it is a probability density function and theoretically ranges from 0 to infinite. This function is dependent on the variance of the learned prior probability. If this variance is high, manipulating the shape modulation does not cause infeasible motions. In contrast, if the variance is low, manipulating the shape modulation is probably not a good idea because it shows the same behavior of ProMP. In this case, the threshold should be high enough to eliminate the possibility of generating unreasonable motions. The more stable way is to test whether the via-point lands out of the range $(\mu - n * \sigma, \mu + n * \sigma)$ where n is a chosen integer. For each dimension, it can also be different, especially when we learn the dimension of the trajectory independently.

3.3 Robot Applications

The via-points integration in the VMP formulation is an advantage over other methods such as DS, DMP, or ProMP. The ability to integrate via-points makes many robotic applications much simpler. In Stulp et al. (2011) and Rueckert and D'Avella (2013), researchers extended the learning process of a DMP. They

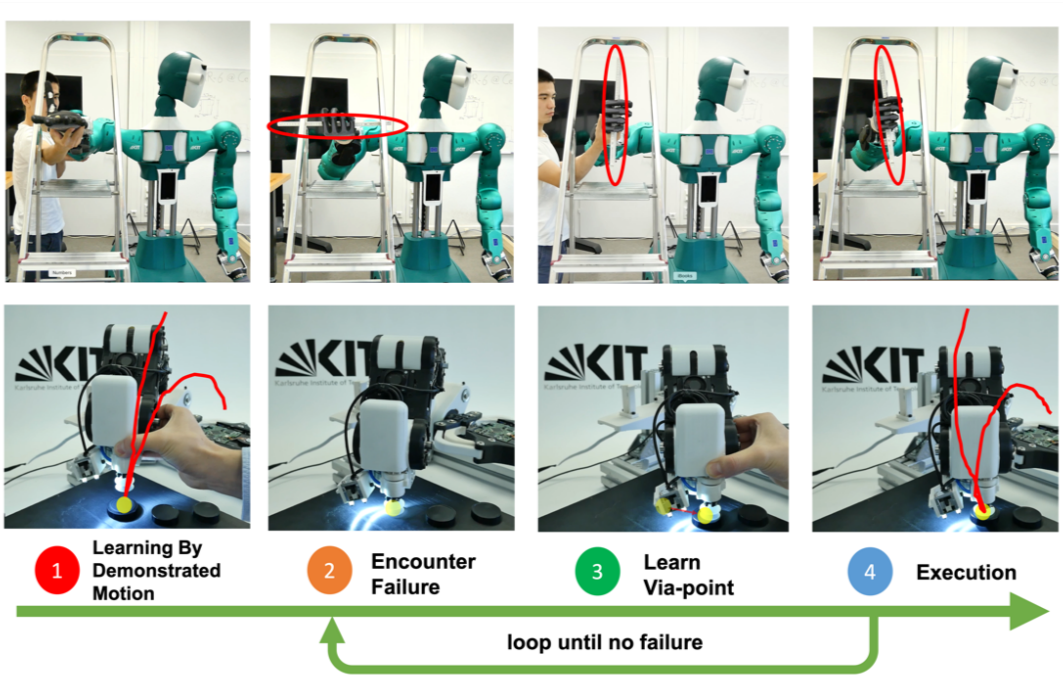

Figure 3.8: In the **upper diagram**, ARMAR-6 learns how to pass a ruler through the narrow area. After that, it fails to pass a longer object through. With only one via-point, the robot accomplishes the task. In the **lower diagram**, the sucker gripper learns how to transport the object. The whole motion is encoded with one VMP. The via-point is used to specify the location of the object.

tested their methods with via-points tasks, for which they designed a cost function and used either optimization methods or reinforcement learning to minimize it. However, these methods require several iterations, and their results are usually dependent on the cost function and learning methods. Instead, VMP with a constant speed linear elementary trajectory can directly generate trajectories that coincide with the results of an optimization method (see Eq. 3.15).

3.3.1 Robot Learning Framework

A major problem of an MP used in the robotic area is its weak adaptability. Although VMPs improve the adaptability regarding the via-points integration, it still cannot guarantee that the generated motion can fulfill the task requirements. DMPs assumes that the goal adapted motion is generated by a changed goal of its underlying damped spring system, while ProMPs assume that the skill-associated trajectories follow a Gaussian distribution, and the conditional probability realizes the adaptation. These assumptions might be wrong, and

the generated motions might not be able to accomplish the task. The fundamental reason is that we do not provide any task specified cost functions like in reinforcement learning but only try to mimic the human motions when using MPs for learning from demonstrations. This weakness can be solved by other learning methods built upon the MP system. In Chapter. 4, several methods to generalize MP for different task constraints are discussed in detail. In this section, based on VMPs, a robot learning framework is developed to allow the robot to learn from a human when encountering failures.

The basic idea is to integrate the specific via-points in case of failures in executing the motion generated by the current VMP. This process benefits from the property of VMP that allows integrating as many via-points as necessary. This supports a natural human-robot interaction. In reality, when we learn a specific skill, the teachers usually prefer adjusting our pose during the motion execution instead of showing us the motion again and again. By teaching based on pose adjustment, the teacher considers the individual difference. By remembering the intermediate pose, we speed up the skill acquisition of the robot. In many cases, remembering the pose might be more efficient than memorizing the whole motion trajectory. In learning by demonstration, it is much easier to use intermediate via-points to either correct the error during the motion reproduction or to help the robot to adapt the learned motion to new task constraints than to repeat the demonstrations multiple times.

In Fig. 3.8, we show one example with the humanoid robot ARMAR-6 described in Asfour et al. (2018), where the learned motion is to pass a short ruler through the narrow area of the ladder. However, the robot cannot pass the long ruler through, hence, encounters a failure. In order to solve the problem with DMP or ProMP, the human needs to show the robot a new motion with a suitable orientation trajectory. With VMP, however, the human just stop the robot when it collides with the ladder and adjusts the pose of its hand by rotating 90 deg. With VMP and this orientation via-point, the robot can simply pass the long ruler through the restricted area without learning a new motion. In reality, the robot might encounter many tasks with slightly different task constraints. Memorizing via-points is much more efficient than memorizing entire trajectories.

In Fig. 3.8, another example illustrates that the usage of via-points is not only the obstacle avoidance. The sucker gripper (Borràs et al. (2018)) learns how to reach and transport the object with only one VMP. When it tries to move another object in a different location, it fails. By memorizing a via-point, the sucker gripper can transport the new object. With DMP, however, we need

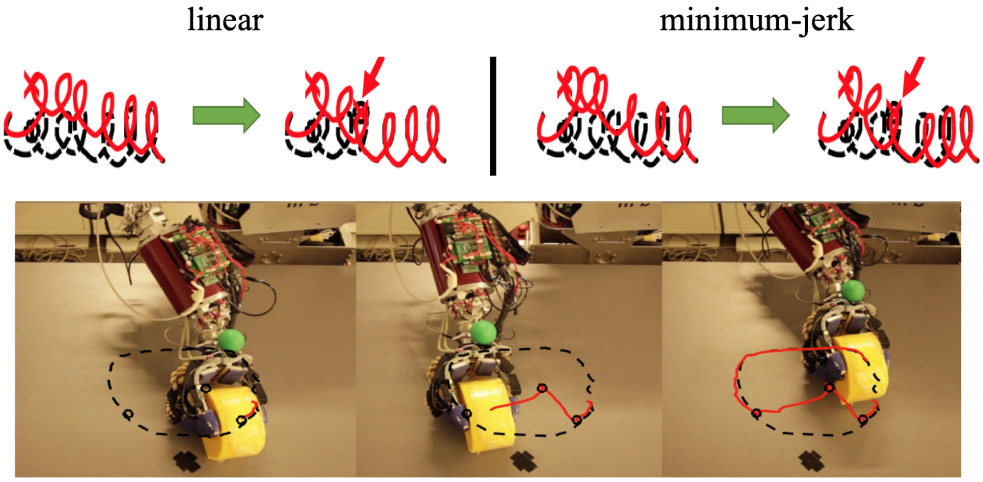

Figure 3.9: A via-point on the original demonstrated trajectory guarantees that the rest trajectory is similar to the demonstrated one. By doing this, ARMAR-III avoids obstacles during wiping, but does not change the wiping pattern afterwards.

either to have two separate DMPs, one of which approaches the object, and the other one transports it, or to train an another DMP for the new object at the new location. With ProMP, we need to demonstrate multiple times to cover the space surrounding the possible object locations. Otherwise, it might result in an unexpected infeasible motion.

The robot learning framework is built based on VMPs. Memorizing the via-point can solve many problems and is more efficient than requiring new demonstrations. With only one learned parameters vector, a VMP is more compact representations than DMP and ProMP, because it can generate different trajectories for many different task constraints.

3.3.2 Return Property

Another property of VMPs is its return property that is useful in many robot applications. The "return" property refers to the fact that a via-point on the original trajectory drags the generated motion back to the original demonstrated one. Because we do not have any task-specific cost functions when we use MP for learning by demonstration, any MP cannot guarantee the absolute correctness of generated motions for new task constraints. Hence, it is much safer to keep the executed motion near to the demonstrated one when no via-points are required. In Fig. 3.9, we show that a via-point on the demonstrated trajec-

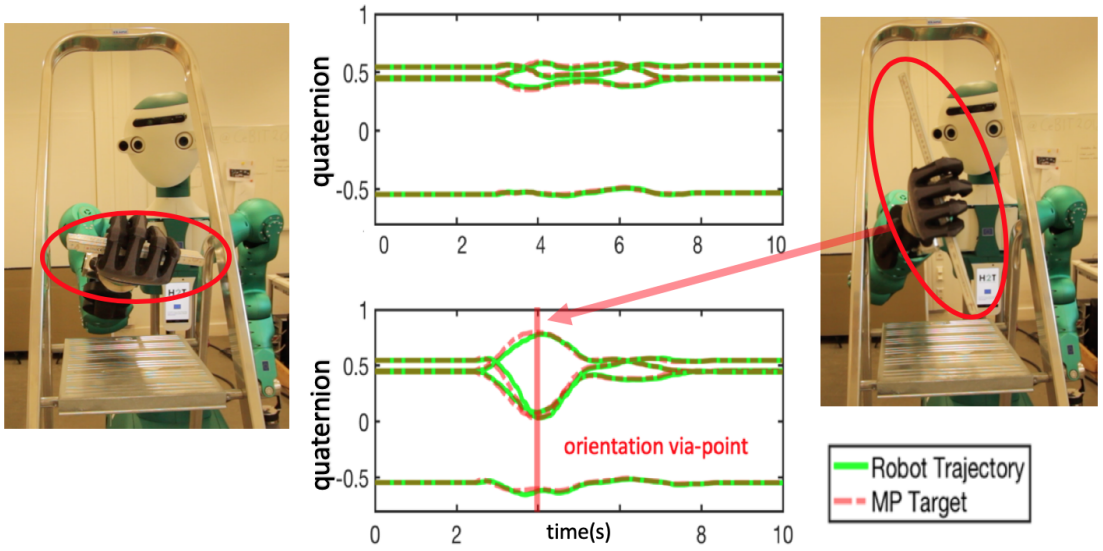

Figure 3.10: An orientation via-point is integrated to pass a longer ruler through the narrow area. The red bar indicates the via-point.

tory guarantees that the remaining part of the generated trajectory coincides with the demonstrated one. We can apply this property, for example, for obstacle avoidance in the wiping task, where the robot needs to avoid some obstacles during wiping, but does not change the original wiping pattern a lot (see Fig. 3.9).

Imagine that a robot is standing in front of the table and retracts its hand from above the table. In order to avoid a collision while retracting hand from anywhere on the table, the robot can remember a via-point. In such situations, we need two DMPs to accomplish this task or multiple demonstrations to train ProMP.

3.3.3 Obstacle Avoidance

Integrating via-points to the generated trajectories is one of the most intuitive ways for obstacle avoidance. The standard way to solve the obstacle avoidance problem is either to apply the path planning algorithms such as rapidly-exploring random tree (RRT) (Lavalle et al. (2000)) or to create a repulse field (force or velocity field) (Khatib (1990)) to avoid the collision. The former methods are usually time-consuming and can result in counter-intuitive joint motions. The latter methods require full knowledge about the obstacles and can possibly be stuck into local optimum. Both methods are unlikely to generate motion similar to humans.

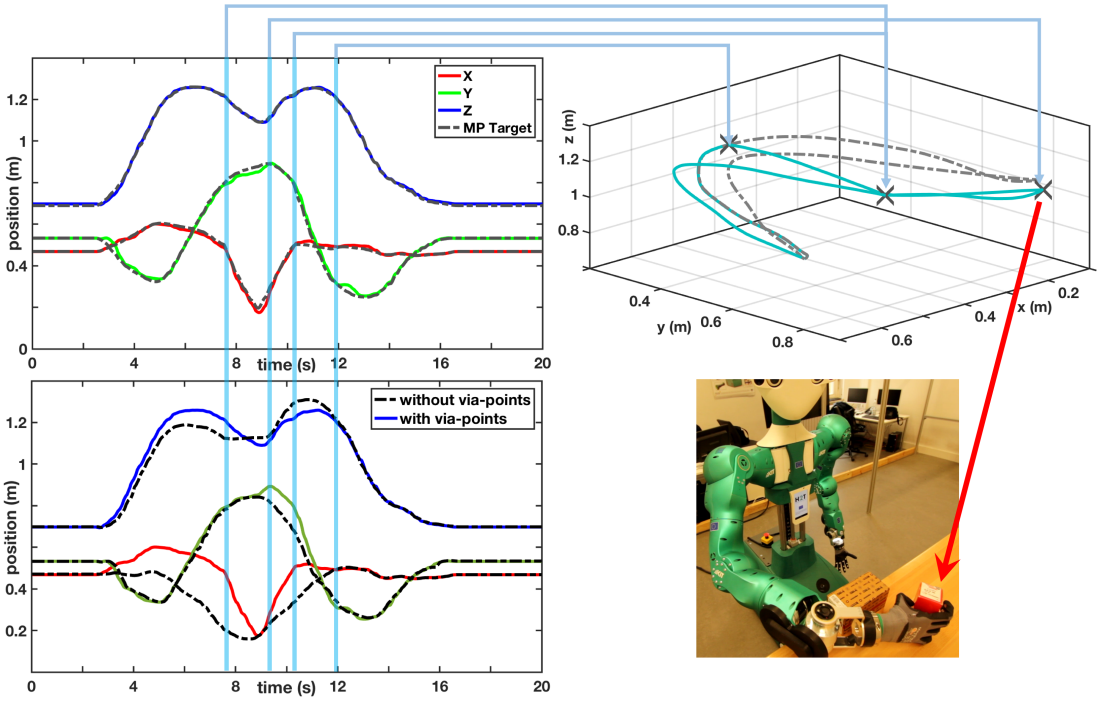

Figure 3.11: Four via-points are integrated to realize a collision free grasping. Two via-points are set at the same location with a distance to the obstacle to avoid it. One via-point takes the robot's manipulability into the consideration for grasping (see the red arrow). The other one via-point utilizes the return property of VMP to guarantee no collision during hand retraction.

VMPs provide another way for obstacle avoidance by combining the motion planner with learning by demonstration. In this case, a local motion planner that works in the task space is required. It generates a sequence of via-points with which the robot can avoid the obstacles. For the arm motion, however, this task space local motion planner should also take the joint space into the considerations. Currently, we manually design this local motion planner for different obstacles and their locations. In many cases, it is not difficult to design the via-points for obstacle avoidance. In Fig. 3.10, we show an example where the robot easily obtained an orientation via-point when observing the boundary of the narrow area.

In Fig. 3.11, we show another obstacle avoidance example, where ARMAR-6 grasps an object on the table which is behind an obstacle. The robot uses four via-points to avoid the obstacle and successfully grasps the target object. In this case, the robot calculated these via-points by observing the relative positions of the obstacle to the target object and its manipulability when approaching

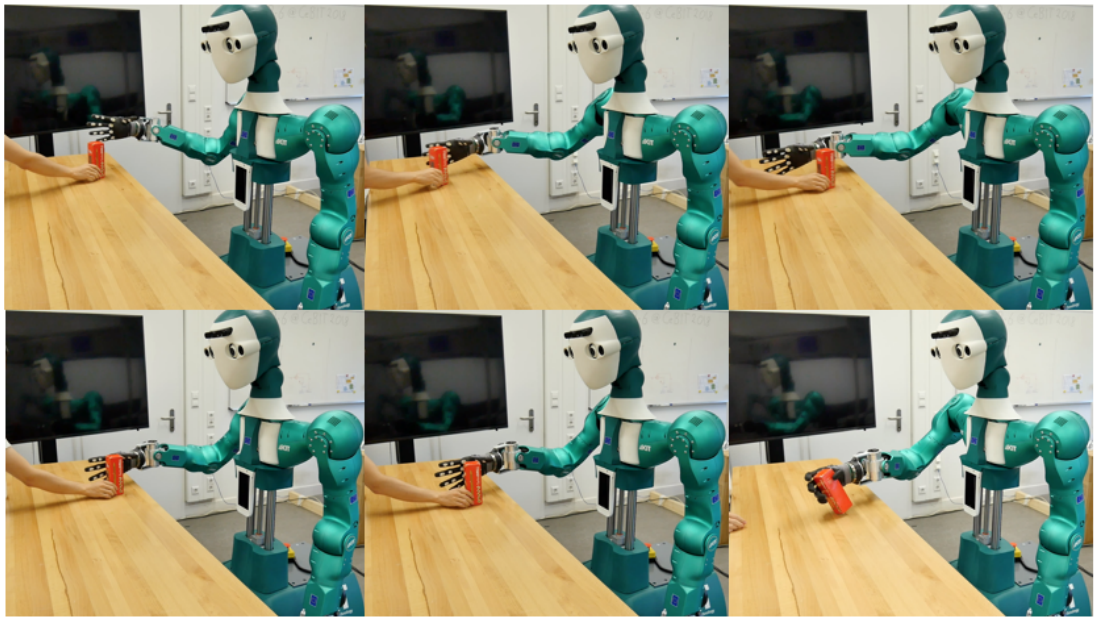

Figure 3.12: The via-points are online integrated to the motion for object grasping .

the target object. We can design a general local motion planner to fulfill the collision-free grasping task. Compared to direct motion planning, VMP generates more human-like motions because they are learned from humans and has been proved to be similar to the demonstrated trajectory with required via-points.

3.3.4 Online Via-Points Integration

During the execution of a VMP, additional via-points can be integrated into the trajectory as required. For this purpose, the VMP re-calculates the trajectory based on newly integrated via-points. In Fig. 3.12, we show how this online via-points integration works. In this case, ARMAR-6 is trying to grasp an object moved by a human. We represent the whole motion with one VMP. With the camera on the head, the robot can detect the position of the object. We associate the position of the object with a canonical value and form a via-point (x_{via}, y_{via}). We obtain the canonical value by considering an appropriate time offset for moving from the current position to the current object position. If the robot has not yet grasped the object, the total motion time duration will increase with a fixed amount of time to avoid the high speed of the motion.

3.4 Conclusion

In this chapter, a novel formulation for movement primitives, *Via-points Movement Primitive* (VMP), is introduced, which allows flexible via-points integration into the generated trajectory to accomplish tasks. Compared to the DMP formulation, the VMP is a probabilistic formulation, i.e. it can encode multiple demonstrations. Furthermore, a VMP allows intermediate via-points adaptation, which cannot be done by a DMP. Compared to a ProMP, a VMP can handle via-points extrapolation. a ProMP, however, generates infeasible motions when the via-points are out of the range of demonstrations.

Based on the via-points adaptation of VMPs, we have developed a robot learning framework. Instead of learning new motions, a robot accomplishes a particular task by memorizing a set of via-points. For the same task, however, we require multiple DMPs or multiple demonstrations to learn appropriate ProMPs.

4 Movement Primitive Generalization

The adaptation based only on the MP representation is not enough to generate motions for any task parameters. VMPs can only adapt to starts, goals, and intermediate via-points to change the shape of the trajectory. If we consider an arbitrary task parameter and a set of demonstrations for different queries, we need to be able to generalize the learned MP to different tasks with different task parameter queries.

In Chapter. 2, different methods to generalize MPs to different task parameters have been introduced and compared. They are classified into two categories: learning a direct mapping or learning a generative model. As mentioned, learning a generative model is more difficult than learning a direct mapping and usually requires a large number of demonstrations. In this chapter, we mainly focus on how to learn a direct mapping from the task parameter queries to the motion parameters.

In Section. 2.2.1, two different strategies have been discussed for learning a direct mapping. One type of approach learns two regression models, hence, called two-steps methods. One regression model $f(x)$ maps the canonical variable to the trajectory related variable, which, for example, is the force term of a DMP or the trajectory point in a ProMP. The other regression model $\boldsymbol{\omega}(\boldsymbol{q})$ maps the task parameter queries to the parameter vector $\boldsymbol{w}$ of the previous regression model $f(x)$. The other type of approaches combine both regression models to one single function $f(x, \boldsymbol{q})$ that takes the canonical variable and the task parameters queries as inputs and outputs the trajectory related variable. Both types of approaches have been compared and their advantages and disadvantages have been discussed. In this chapter, we focus on the development of two-steps methods for generalization of movement primitives.

Two approaches to learn the mapping from task parameter queries to movement primitive parameters are introduced. One approach is to use a mixture of experts model for the mapping. A *Leave-One-Out Expectation Maximization*

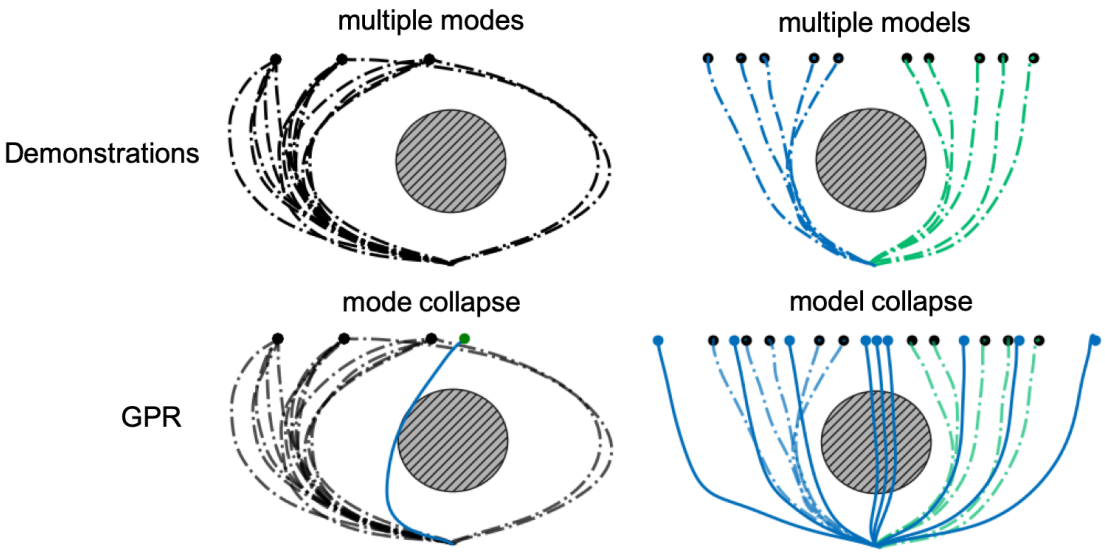

Figure 4.1: **Left:** multiple modes; **Right:** multiple models.

(LOO-EM) is developed to learn the mixture of experts model. LOO-EM outperforms the original EM algorithm, which is easily stuck in the local minima that results in a poor performance when learning the mixture of experts model. Another approach is to use a *Mixture Density Network* (MDN) to learn a mapping of task parameters to distribution of movement primitive parameters. In order to reduce the occurrence of mode and model collapse, a so-called *entropy cost* is introduced to distribute the demonstrations to different mixture components in a balanced way. To further improve training MDN, we develop a failure cost that keeps MDN from outputting MP parameters similar to those parameters that lead to failure. Several experiments show that an MDN trained with new cost functions outperforms previous methods.

The work for learning mixture of experts models with LOO-EM has been published in Zhou and Asfour (2017), and the work for MDN based MP generalization has been published in Zhou et al. (2020).

4.1 Multiple Modes and Models

In robot learning from human demonstration, humans usually have to demonstrate a task without instructions, especially when we require multiple demonstrations. These demonstrations can have multiple modes, which means that humans generate different types of motions to demonstrate the same task with the same task parameters. They can also have multiple models, which means

that humans generate motions for different task parameters with different hidden models.

For example, given the task of reaching a target while avoiding an obstacle (see the left column of Fig. 4.1), a specific goal as a task parameter query can be reached using two different modes, i.e., by trajectories passing the obstacle on the left or the right. Thus, learning a movement primitive for such task should take *multiple modes* in human demonstrations into account to increase the diversity of motions, which is beneficial if the task constraints change. Even though there are no multiple modes for each single task parameter query in human demonstrations, there might exist *multiple models* for different types of task parameters. As shown in the right column of Fig. 4.1, the goals on the left and right sides of the obstacle are reached separately by two types of trajectories, which are generated with two different models to allow passing the obstacle on the left or the right.

4.1.1 Mode and Model Collapse

Previous two-steps methods that learn a direct mapping from task parameters $\boldsymbol{q}$ to MP parameters $\boldsymbol{w}$ cannot handle multiple modes and models. Hence, they can lead to *mode collapse* and *model collapse* as shown in the bottom row of Fig. 4.1. The reason is that the traditional regression models, such as GPR, do not take multiple modes and models into consideration. Hence, this happens also for one-step methods. As mentioned before, in this chapter, we mainly consider how to solve these problems for the two-steps methods.

Overall, for two-steps methods, the purpose is to learn a mapping $\boldsymbol{\omega}$ from task parameters $\boldsymbol{q}$ to MP parameters $\boldsymbol{w}$. With M demonstrations, a training dataset $\{(\boldsymbol{q}_i, \boldsymbol{w}_i)\}_{i=1}^{M}$ is collected. The i-th MP parameter $\boldsymbol{w}_i$ is learned from the i-th demonstration. In DMPs and ProMPs, we obtain the MP parameters by solving a least square problem.

A deterministic regression model $\boldsymbol{\omega}$ that always maps one task parameter query to one specific MP parameter cannot avoid *mode collapse*. Hence, a probabilistic model is required to keep the motion diversity that exists in human demonstrations. Motion diversity is crucial because it is beneficial when the change of the task constraints leads to the failure of some types of motions. In these cases, a deterministic model would not work. If the task constraints do not change after the motion generation, it is not necessary to consider multiple modes, because the robot can only execute one motion at one time. As far as the generated motion is valid and reasonable for the task, the task execution will succeed.

Model collapse is more severe for a deterministic model than the mode collapse because it leads to failure of task execution, as shown in the bottom right of Fig. 4.1. The reason for the failure with traditional regression models is that they usually consider only one model. However, human demonstrations can include several different models. In this case, the single learned models perform worse, especially for task parameter queries on the boundary between the two models. They tend to average the results from both models, which lead to trajectories going directly through the obstacle in Fig. 4.1.

In this chapter, two different approaches are introduced. One of them is to solve the model collapse for the deterministic models. The other one is to create a probabilistic model based on *Mixture Density Networks* (MDN) to resolve both mode and model collapse.

4.2 Mixture of Experts for Movement Primitve Generalization

One way to solve the model collapse problem for deterministic models is to combine multiple regression models and create a mixture of experts, as described in Jacobs et al. (1991). Then, we divide the space of the task parameter queries into different regions. We learn each expert of the mixture model from human demonstrations which correspond to the task parameters in one of the regions. During task execution, we assign a new task parameter query to one of the models according to its region. The model takes the query as input and outputs the MP parameter. In Fig. 4.1, we divide the space of the trajectory goals, as the task parameters, from the middle. If we train two GPRs for two separate groups of the demonstrations, the model collapse will not occur.

Assuming that the number of experts is known, to use the mixture of experts, we need to solve two problems. One is how to assign each demonstration to the models during training of the mixture model. The other is which model is to select for a newly encountered task parameter query.

A simple solution to the latter problem is to find the training task parameter query that is the closest to the new query and use its model to determine the MP parameter. In Da Silva et al. (2012), the authors presented an alternative, where they trained a classifier that takes the task parameter query as input and outputs a label indicating which model to use.

4.2.1 Training Mixture of Experts

For training the mixture of experts model, if we know how to assign demonstrations to models, it is trivial to train each regression model (or expert) separately based on their associated demonstrations. However, it is often not trivial to determine such training data association. If the number of experts K is known, the training purpose is to infer the parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_k\}_{k=1}^K$. For the i-th pair of training data $(\boldsymbol{q}_i, \boldsymbol{w}_i)$, the posterior probability of the association index k_i is given by

$$p(k_i|\boldsymbol{q}_i, \boldsymbol{w}_i) = \frac{1}{Z} p(\boldsymbol{w}_i|k_i, \boldsymbol{q}_i) p(k_i|\boldsymbol{q}_i), \tag{4.1}$$

where Z is the normalizing coefficient.

We assume that

$$p(\boldsymbol{w}_i|k_i, \boldsymbol{q}_i) \propto exp\left(-h||\boldsymbol{w}_i - \boldsymbol{\omega}(\boldsymbol{q}_i; \boldsymbol{\theta}_{k_i})||\right) \tag{4.2}$$

with h as a constant and $\boldsymbol{\omega}(\cdot; \boldsymbol{\theta})$ as a parametric function with the parameter $\boldsymbol{\theta}$ that corresponds to one expert model. $p(\boldsymbol{w}_i|k_i, \boldsymbol{q}_i)$ reaches its maximum if $\boldsymbol{w}_i = \boldsymbol{\omega}(\boldsymbol{q}_i; \boldsymbol{\theta}_{k_i})$ and decreases when $\boldsymbol{w}_i$ and $\boldsymbol{\omega}(\boldsymbol{q}_i; \boldsymbol{\theta}_{k_i})$ depart from each other. Based on this assumption, the posterior probability satisfies

$$p(k_i|\boldsymbol{w}_i, \boldsymbol{q}_i) \propto exp\left(-h||\boldsymbol{w}_i - \boldsymbol{\omega}(\boldsymbol{q}_i; \boldsymbol{\theta}_{k_i})||\right). \tag{4.3}$$

Expectation Maximization

To infer the parameters $\boldsymbol{\Theta}$, we consider $\{k_i\}$ as hidden variables and use the *Expectation Maximization* (EM) algorithm. Before the E- and M-step, *K-means clustering* is conducted to separate the training MP parameters $\{\boldsymbol{w}_i\}_{i=1}^N$ into K clusters and K models are trained for the initial parameters set $\{\boldsymbol{\theta}_k\}_{k=1}^K$. In the E-step, for each pair $(\boldsymbol{q}_i, \boldsymbol{w}_i)$, the current optimal association index is calculated based on Eq. 4.3:

$$k_i^* = \underset{k}{\operatorname{argmax}}\, p(k|\boldsymbol{q}_i, \boldsymbol{w}_i). \tag{4.4}$$

In the M-step, K models are trained again on the new clusters based on the current association indices. By iterating between E- and M-steps, the algorithm finds a local minimum.

If the regression model $\boldsymbol{\omega}$ has a strong representation capability, i.e. it can represent complex functions, or the training dataset is small, EM algorithm could converge to the local minima, where a subset of K models overfit the training

data. These local minima lead to the *model collapse*, as mentioned before. In the extreme case, the EM algorithm stops at the first iteration and keeps the initial training data assignment to different models because each model overfits its associated training data.

Leave-One-Out Expectation Maximization (LOO-EM)

To solve the problem, instead of limiting the capability of the regression models such as using a linear model, we consider an alternative based on the idea of the *leave-one-out* (LOO) cross-validation.

In the M-step, instead of only training K models, we train $K + N$ models. For each data pair $(\boldsymbol{q}_i, \boldsymbol{w}_i)$, we train a so-called LOO model. We obtain the training dataset for the model by dropping the data $(\boldsymbol{q}_i, \boldsymbol{w}_i)$ from the current training dataset of the k_i-th model. With the performance of these leave-one-out models, we check how significantly this training data affects the k_i-th model by evaluating its fitting error with the LOO model. The more significant it affects its current model, the more possible its current model overfits it, and the less likely it belongs to this model. We use $\boldsymbol{\theta}_i^{loo}$ to denote the parameters of the leave-one-out model for the i-th training data.

In the E-step, the model association indices are updated with Eq. 4.4 based on the following posterior probability:

$$p(k|\boldsymbol{q}_i, \boldsymbol{w}_i) \propto \begin{cases} exp(-h||\boldsymbol{w}_i - \boldsymbol{\omega}(\boldsymbol{q}_i; \boldsymbol{\theta}_k)||), & \text{if } (\boldsymbol{q}_i, \boldsymbol{w}_i) \notin M_k \\ exp(-h||\boldsymbol{w}_i - \boldsymbol{\omega}(\boldsymbol{q}_i; \boldsymbol{\theta}_i^{loo})||), & \text{if } (\boldsymbol{q}_i, \boldsymbol{w}_i) \in M_k, \end{cases} \tag{4.5}$$

where M_k is the training dataset of the current k-th model.

Except the data association indices $\{k_i\}_{i=1}^N$, $\{\boldsymbol{\theta}_i^{loo}\}_{i=1}^N$ is also regarded as the hidden variables in the *Leave-One-Out Expectation Maximization* (LOO-EM) algorithm.

Let's consider a multi-modal mapping as shown by the black dashed curves on the left side of the Fig. 4.2 and is described by:

$$f(x) = \begin{cases} \frac{1}{x} + 1, & \text{if } x \leq 0 \\ \frac{1}{x} - 1, & \text{if } x > 0. \end{cases} \tag{4.6}$$

We compare the use of EM and LOO-EM for learning a mixture of experts. Each expert is a two-layers neural network and each layer contains 40 neurons. The neural network is trained with a gradient descent method. As shown

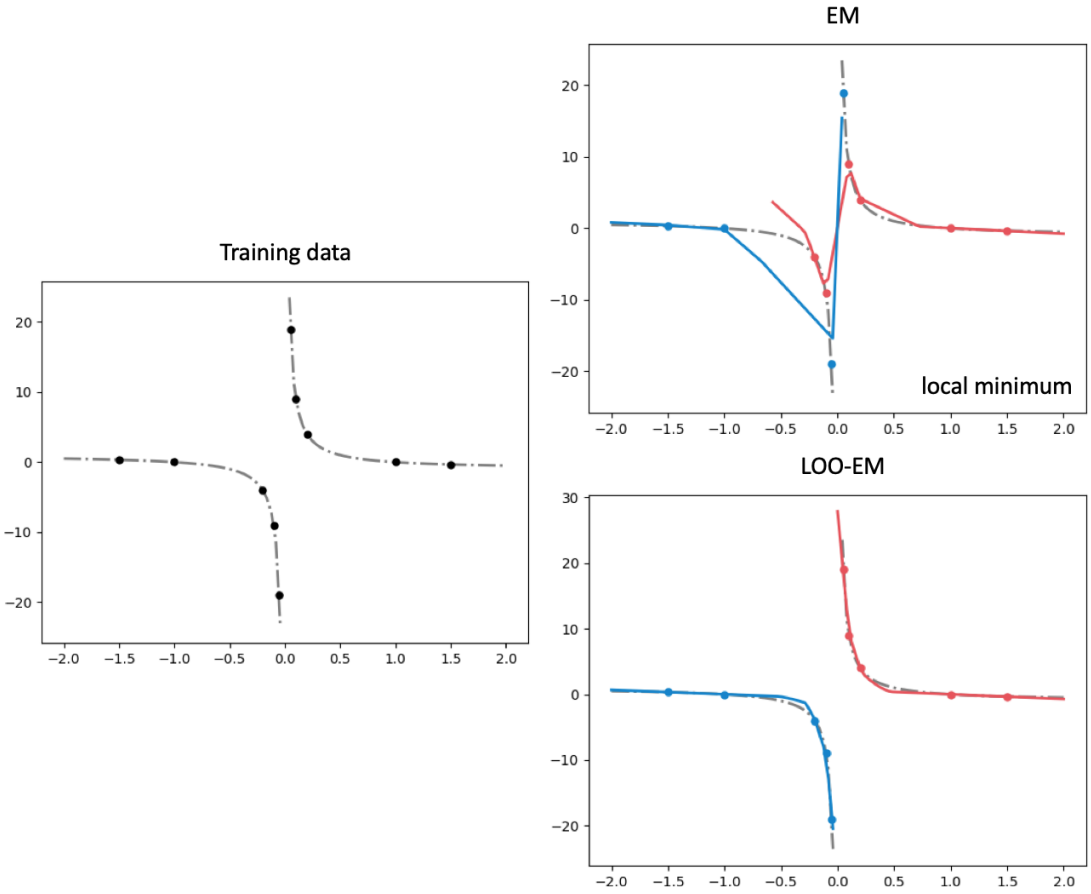

Figure 4.2: Comparison between EM and LOO-EM for a multi-modal mapping.

in Fig. 4.2, the training data points are indicated by the dots. Since a neural network is powerful enough to approximately go through all the training data points, EM is simply stuck in a local minimum (Fig. 4.2, right-top). Although almost all training data points are met by the learned regression model, the result regression models cannot predict other points on the desired mapping. In contrast, LOO-EM learns the correct data association (Fig. 4.2, right-bottom). In 30 experiments with random initialization of the data association, EM obtains an error of around 2.6 . While LOO-EM has an error of around 0.7.

The LOO-EM algorithm is, however, time-consuming, especially for a complex task, where we need a large dataset. Furthermore, the learned system is still deterministic. Hence, it cannot encode multiple modes.

To handle multiple modes, we need an MP generalization model which can output a probabilistic representation. A *Mixture Density Network* (MDN), described in Bishop (1994), outputs a mixture of Gaussian distributions which can encode multiple modes and models. Furthermore, an MDN takes the ar-

chitecture of neural networks, which allows it to handle complex tasks.

In the rest of this chapter, we suggest to use a *Mixture Density Network* (MDN) to model the mapping $\boldsymbol{\omega}(\cdot)$ in a two-steps method.

4.3 Mixture Density Networks for Movement Primitive Generalization

Using methods such as LWR (Ude et al. (2010)), SVR (Da Silva et al. (2012)) or GPR (Forte et al. (2012)) to model the mapping are good enough to accomplish simple tasks with a small number of demonstrations (less than 100). For complex tasks, however, they fail to scale to large datasets and result in poor performance.

Furthermore, single regression models cannot handle multiple modes and models and lead to the mode and model collapse. The LOO-EM algorithm solves the model collapse problem, however, with a high computational cost. In the M-step, the algorithm requires training of the regression model for each data point. If we use a neural network with a big set of parameters, the algorithm is intractable for a large dataset.

As described in the previous section, MDN is proposed. Combined with the *Via-points Movement Primitive* (VMP) introduced in the last chapter, it provides a probabilistic model and can scale from simple to complex tasks, as we will describe in the following.

4.3.1 Extended Via-points Movement Primitive

In last chapter, we introduced VMP to represent the motions. The formulation of VMP (see also Eq. 3.3) is given by

$$\boldsymbol{y}(x) = \boldsymbol{h}(x) + \boldsymbol{f}(x), \tag{4.7}$$

where the shape modulation $\boldsymbol{f}$ is a linear regression model such that

$$\boldsymbol{f}(x) = \boldsymbol{\psi}(x)^T \boldsymbol{w}, \tag{4.8}$$

whose parameter follows a Gaussian distribution, namely $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For M demonstrations, the parameters of the Gaussian distribution are determined by

the empirical mean and covariance, which correspond to the *maximum likelihood estimation* (MLE):

$$\boldsymbol{\mu} = \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{w}_i, \quad \boldsymbol{\Sigma} = \frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{w}_i - \boldsymbol{\mu})(\boldsymbol{w}_i - \boldsymbol{\mu})^T. \tag{4.9}$$

The i-th MP parameter $\boldsymbol{w}_i$ is obtained by solving a least square problem for the i-th demonstration $\boldsymbol{Y}_i$ and can be written as follows

$$\boldsymbol{w}_i = (\boldsymbol{\Psi}^T\boldsymbol{\Psi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Psi}^T(\boldsymbol{Y}_i - \boldsymbol{H}), \tag{4.10}$$

where $\boldsymbol{H}$ is a vector representing points in the elementary trajectory. Here, we assume that it is a line with a constant velocity profile (see Eq. 3.2.2).

We replace the Gaussian distribution with a *Gaussian Mixture Model* (GMM) and consider that the parameters of the GMM are the functions of the task parameters $\boldsymbol{q}$. Hence, we have

$$\boldsymbol{w} = \omega(\boldsymbol{q}) \sim \prod_{k=1}^{K}\mathcal{N}(\boldsymbol{\mu}_k(\boldsymbol{q}), \boldsymbol{\Sigma}_k(\boldsymbol{q}))^{z_k}, \tag{4.11}$$

where z_k is an element of a K-dimensional binary variable $\boldsymbol{z} = (z_1, z_2, ..., z_K)^T$ where only one particular element is equal to 1 and all other elements are zero. The probability of the k-th element of $\boldsymbol{z}$ being equal to 1 is

$$p(z_k = 1) = \pi_k(\boldsymbol{q}).$$

The probability of the result $\boldsymbol{w}$ given the task parameter $\boldsymbol{q}$ is

$$p(\boldsymbol{w}|\boldsymbol{q}) = \sum_{k=1}^{K}\pi_k(\boldsymbol{q})\mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_k(\boldsymbol{q}), \boldsymbol{\Sigma}_k(\boldsymbol{q})). \tag{4.12}$$

4.3.2 Mixture Density Network (MDN)

Structure of MDN

A *Mixture Density Network* (MDN) for movement primitive generalization is shown in Eq. 4.11. An MDN models $\pi_k(\cdot)$, $\boldsymbol{\mu}_k(\cdot)$ and $\boldsymbol{\Sigma}_k(\cdot)$ by three neural networks to determine the MP parameters. For a detail introduction to MDN, the readers are referred to Bishop (1994).

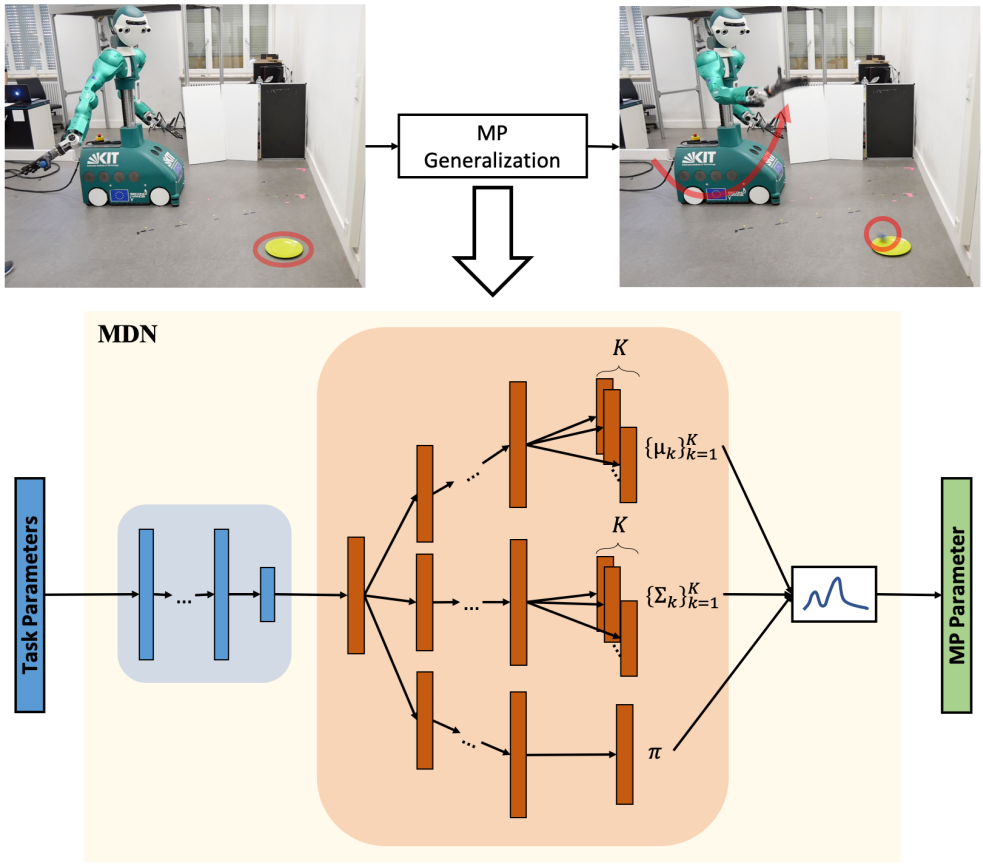

Figure 4.3: Mixture Density Network (MDN) for the MP generalization

The three neural networks share a common network part (blue region in Fig. 4.3) that extracts the latent features and is exchangeable for different tasks. The output dimension of the mixture coefficients functions $\pi(\boldsymbol{q})$ is the number of the mixture components K. The output dimension of the mean network $\boldsymbol{\mu}(\boldsymbol{q})$ is the dimension of the MP parameter $\boldsymbol{w}$ or the number of kernels, which determines the reproduction accuracy of the MP. In many applications, 10 kernels are enough for one *degree of freedom* (DoF). Hence, for a three-dimensional trajectory, the output of the mean network has 30 dimensions. According to Mclachlan and Basford (1988), Eq. 4.12 can approximate any given density function to arbitrary accuracy with a diagonal covariance matrix. The output dimension of the covariance network $\boldsymbol{\Sigma}(\boldsymbol{q})$ coincides with the mean network. Hence, for a three-dimensional trajectory, an MDN has a total of $K + K \times 30 \times 2$ output values.

To train an MDN, we consider the *negative-log-likelihood* (NLL) cost such that

$$l_{NLL}(\boldsymbol{\Theta}) = -\sum_{i=1}^{M} log\Big(\sum_{k=1}^{K} \pi_k(\boldsymbol{q}_i;\boldsymbol{\Theta})\mathcal{N}\Big(\boldsymbol{w}_i|\boldsymbol{\mu}_k(\boldsymbol{q}_i;\boldsymbol{\Theta}),\boldsymbol{\Sigma}_k(\boldsymbol{q}_i;\boldsymbol{\Theta})\Big)\Big), \tag{4.13}$$

where $\boldsymbol{\Theta}$ denotes all the parameters of the MDN. The normal distribution with a diagonal covariance matrix can be written as

$$\mathcal{N}\left(\boldsymbol{w}_i|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i\right) = \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}_i|^{1/2}} exp\left(-\frac{1}{2}\sum_{j=1}^{N}\frac{(w_{i,j}-\mu_{i,j})^2}{\sigma_{i,j}^2}\right), \tag{4.14}$$

where $\boldsymbol{\mu}_i = \boldsymbol{\mu}_k(\boldsymbol{q}_i;\boldsymbol{\Theta})$ and $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_k(\boldsymbol{q}_i;\boldsymbol{\Theta})$ and N is the dimension of the MP parameter or the number of kernels. A stochastic gradient descent algorithm such as Adam (Kingma and Ba (2014)) is used to minimize the NLL cost.

With a gradient descent method, we can only find local minima and expect that they are good enough to accomplish the tasks. However, this is sometimes not the truth.

Let us take a look at the simple previous example where a fixed obstacle should be avoided in Fig. 4.1. The black dashed curves are the demonstrations. As mentioned before, the left column shows multiple modes, and the right column shows multiple models.

Mode and Model Collapse with Previous Methods

For multiple modes, GPR is a deterministic method and captures only one mode. In the left plot of the second row of Fig. 4.1, GPR generates a trajectory that approaches the target from the left side. For a particular task parameter query, a deterministic model always outputs the same trajectory after it is trained.

GPR has another problem in this case. Since several demonstrations are from the right side, the result of GPR is affected and shifted towards right side. Simple regression models such as GPR and SVR tends to get the average of the training outputs for the queries in the middle of the training queries. This problem is more serious when we consier multiple models in the right plot of the second row of Fig. 4.1. It leads to the model collapse and results in failure of task execution.

The mode and model collapse also happens for some other regression models such as SVR because of the same reasons mentioned before.

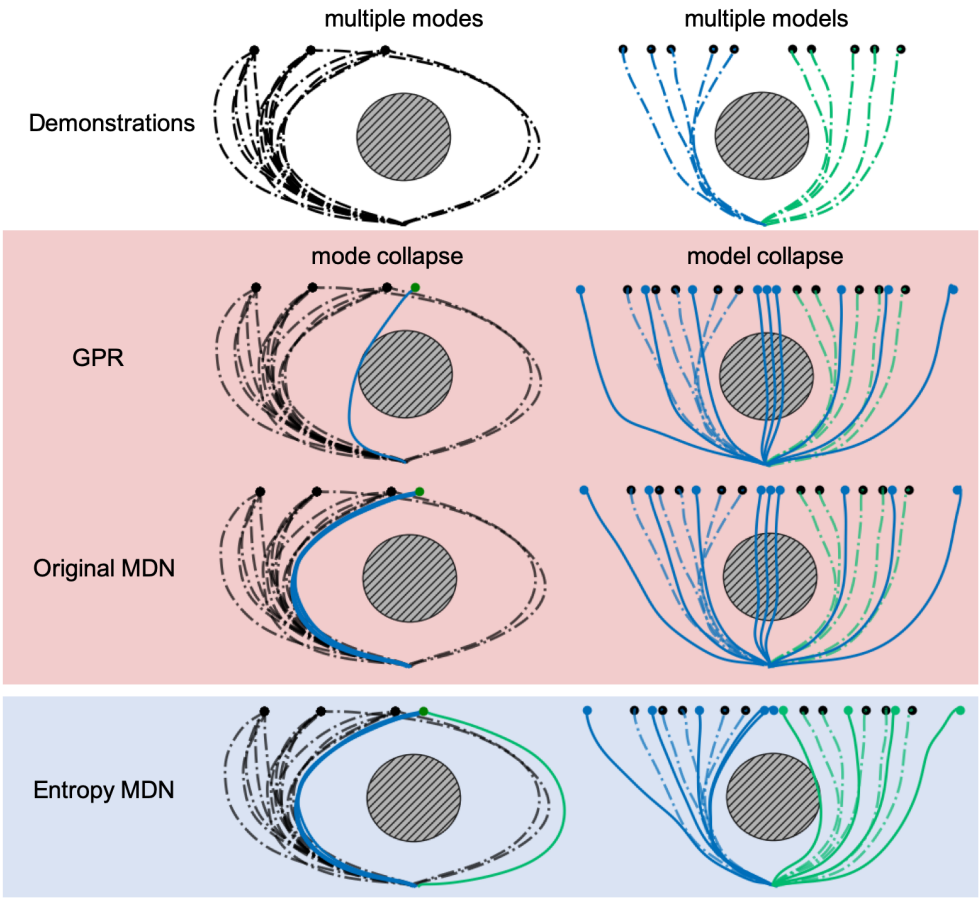

Figure 4.4: Obstacle avoidance with MDN.

Mode and Model Collapse with NLL cost

Unlike GPR or SVR, an MDN is a probabilistic model and outputs a distribution, from which we can draw multiple samples. Hence, we have opportunities to get different types of motions for one particular task parameter query. However, in the experiment (left plot in the third row of Fig. 4.1), the MDN with only the NLL cost (denoted as the original MDN in Fig. 4.4) still loses one mode if we draw 5 samples from its output for the task parameter query denoted with the green dot.

Similar to GPR or SVR, the original MDN with only the NLL cost still loses one model and fails to generate valid motions for the task parameter queries in the middle.

The mode collapse during training MDN is a known problem. In Hjorth and Nabney (1999), the authors solve this problem by reducing the learning complexity. They fix the means and the variance, and train the mixing coefficient

function $\pi(\boldsymbol{q})$ to reduce the NLL cost. If there are enough fixed mixture components regularly distributed in the output space, the learned MDN does not lose its representation capability. However, for the MP parameter as high dimensional output, an MDN requires a large grid that might be intractable.

In Makansi et al. (2019), the authors predict the next car location based on the current one in a 2D map. In order to avoid the mode collapse, they suggest separating the MDN into two parts: a sampling network and an inference network. The sampling network takes the current car location as the input and outputs a number of the hypotheses of the next car locations. To keep the modes, they trained the network to diversely place these hypotheses and increase the likelihood at the same time. After training the sampling network, they trained an inference network to infer the GMM parameters based on the hypotheses. For using this strategy for the MP generalization, a large dimensional output of the sampling network is required, which can also cause problems during the training.

In summary, previous methods in literature cannot be applied for the MP generalization.

4.3.3 MDN with Entropy Costs

Before solving the mode and model collapse, we first discuss in more details the reason of their occurrence.

The mode collapse occurs if demonstrations associated with different modes for one task parameter query are imbalanced. As an example, in Fig. 4.4 (left column), only a small number of demonstrations take the path on the right side. By maximizing the likelihood of all demonstrations, an MDN tends to output a small mixing coefficient for the mixture component, which corresponds to the mode with less associated training data. In theory, it is correct to associate a small probability to events that rarely happen in the observations. However, the imbalance of the demonstrations in different modes can be caused due to the habit of the demonstrator. It is often the case that we cannot collect enough demonstrations to cover all modes. Even if there are only a few demonstrations for a specific way for task execution, these demonstrations should be taken into account to increase the motion diversity.

The model collapse occurs in the case of small number of demonstrations. Several mixture components of the MDN, which are represented by neural networks, are powerful enough to overfit all the demonstrations. After training of

the MDN, instead of all K mixture components, we can use a subset of them to approximately fit all training data. This corresponds to certain local minima of the NLL cost, which results in a poor performance of the MDN for certain task parameter queries. As shown in Fig. 4.4 (right column), one of the two models disappear with the original MDN, and the MDN performs similar to GPR. Compared to the mode collapse, the model collapse is more severe because it can lead to the failure of the task execution.

In order to reduce the occurrence of the mode and model collapses, we introduce a negative model entropy cost function over demonstration set $\boldsymbol{D}$ is as follows,

$$l_{model}(\boldsymbol{\Theta}) = \sum_{k=1}^{K} p(m = k|\boldsymbol{D}; \boldsymbol{\Theta}) \log p(m = k|\boldsymbol{D}; \boldsymbol{\Theta}), \tag{4.15}$$

where

$$p(m = k|\boldsymbol{D}; \boldsymbol{\Theta}) = \sum_{i=1}^{M} \pi_k(\boldsymbol{q}_i; \boldsymbol{\Theta}) p(\boldsymbol{q}_i), \tag{4.16}$$

and m is the component index and $p(\boldsymbol{q}_i) \propto M^{-1}$. By minimizing the cost, we increase the uncertainty of the model labels when considering the entire demonstration set $\boldsymbol{D}$. A high uncertainty of the model labels is equivalent to either equally distributed mixing coefficients for each task parameter query or equally distributed demonstrations to different models. In the former case, if all mixing coefficients for one specific task parameter query are nearly equal and close to $1/K$, each mode has the same probability of being selected to generate motions. Hence, the mode collapse does not occur. In the latter case, if each model is associated with a subset of the demonstrations, the corresponding mixture component, i.e. the network branch, is well trained. Hence, the model collapse does not occur.

The model entropy cost is related to the identifiability of the output GMM. A GMM is identifiable except the relabeling if (see Frühwirth-Schnatter (2006)):

- the components of GMM are different from each other (multiple modes);
- all components are used to explain the data distribution (multiple models).

In the following experiments, the final cost is a weighted sum of the NLL cost (l_{NLL}) and entropy cost (l_{model}) such that

$$l = w_{NLL} l_{NLL} + w_{model} l_{model}, \tag{4.17}$$

where $w_{NLL} = 1$ and $w_{model} = 50$.

As shown in Fig. 4.4, the MDN with both NLL and the entropy cost (denoted as entropy MDN) solve the mode and model collapses.

4.3.4 MDN with Failure Costs

In many applications, samples with failure can be easily collected with an underfitted MDN model. To reduce the occurrence of MP parameters that correspond to failures for similar task parameter queries, we introduce the failure cost function

$$l_{neg}(\boldsymbol{\Theta}) = \sum_{i=1}^{M^*} log\Bigg(\sum_{k=1}^{K} \pi_k(\boldsymbol{q}_i; \boldsymbol{\Theta}) \mathcal{N}\Big(\boldsymbol{w}_i^* | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_{neg}\Big)\Bigg), \tag{4.18}$$

where the normal distribution has the same form as Eq. 4.14 but $\boldsymbol{\Sigma}_{neg} = \sigma_{neg}\boldsymbol{I}$. By minimizing this cost function, the output mean vector $\boldsymbol{\mu}_i$ for a specific task parameter $\boldsymbol{q}_i$ is kept away from the MP parameters $\{\boldsymbol{w}_i^*\}_{i=1}^{M^*}$ that lead to failure.

If σ_{neg} is too small, the failure cost will not affect the results. On the other hand, a too big σ_{neg} can result in trajectories, which are significantly different from demonstrations. Here, we determined σ_{neg} empirically with the smallest variance of all MP parameter components.

For training an MDN with the failure cost, we prepare an evaluation dataset. After a certain number of training steps, we run the MDN on this evaluation dataset and collect the failed samples in a failure dataset $\{\boldsymbol{w}_i^*\}_{i=1}^{M^*}$. In the next training steps, we calculate the failure cost function based on the failures dataset. To keep the computational cost tractable, we use a fixed dataset size M^* and remove the earliest failed samples from it when new samples are collected.

The final cost with the failure cost involved can then be given as

$$l = w_{NLL} l_{NLL} + w_{model} l_{model} + w_{neg} l_{neg}, \tag{4.19}$$

where $w_{NLL} = 1$, $w_{model} = 50$ and $w_{neg} = 1$.

4.3.5 Generating Motion with MDN

In learning from demonstration, a successful task execution means that the generated motions are similar to the demonstrations, and the generated motions

accomplish the task with specific task parameters. In the proposed method, we meet the former requirement by training MDN with the NLL, which is related to the similarity between the collected and generated MP parameters. To check whether the latter requirement is met, we evaluate the trained MDN with the success rate of the task execution.

For the task execution, we can only execute one motion after another. Hence, the MP parameter for the task must be determined based on the MDN output distribution in a subsequent step.

The purpose is to generate single motions for given task parameter queries. In the following experiments, we consider two strategies: selecting the most probable mode or selecting the best one from multiple samples.

The most probable mode is the output mean vector of the mixture component that has the most significant mixing coefficient. If the learned MDN outputs a GMM with separate components, the most probable mode is the mode of the output GMM. For one specific task parameter query $\boldsymbol{q}$, MDN outputs K Gaussian mixture components with their mixing coefficients $\{\pi_k\}_{k=1}^{K}$. The K modes of these Gaussian mixture components correspond to K most probable motions of different types. However, not all these K modes can be used to accomplish the task. The most significant mixing coefficient indicates the mode that most likely succeeds. With this strategy, the MDN serves as a deterministic model. Hence, we can compare it with previous deterministic methods. Selecting the most probable MP parameter is the simplest way to generate motion from the MDN output. This strategy works quite well in many tasks.

However, with the most probable MP parameter, we ignore the information provided by the output variance $\boldsymbol{\Sigma}$ of the MDN. Each of its diagonal elements indicates how various the generated trajectories can be at the corresponding period for successful task execution. When we draw samples from the output GMM for a specific task parameter query, the variance matrix ensures that the samples have a high probability of success. In some tasks, the most probable MP parameter does not work very well because these tasks require a relatively accurate trajectory. A small deviation of the generated trajectory from the desired one leads to failure. To improve the performance, we draw several MP parameters from the MDN output distribution and execute one after another on the robot for the task until success. In this case, a successful task execution means that there is at least one example from the output distribution that leads to success. Hence, the success rate is also dependent on the number of samples drawn from the distribution.

Moreover, with the former strategy, an MDN always generates the same motion for a given specific task parameter query. In order to demonstrate the motion diversity and the fact that multiple modes can be learned by an MDN, we also need to draw multiple samples from the output distribution (see Section. 4.4.5).

We can evaluate The quality of learning from demonstration in two different ways. One way is to assess the similarity between the generated and demonstrated motions. In the MDN case, this similarity is associated with a low NLL cost. The other way is to evaluate task performance or calculate the success rate for different task parameters. The former is highly dependent on the effectiveness of the training algorithms, while the latter is more important for the task. Hence, we use the latter to evaluate the method described before.

4.4 Evaluation

4.4.1 Approximation of Polynomials

In Section. 2.2.1, we compared one-step with two-steps methods using a polynomial fitting test, where a 5-th order polynomial $y(x) = \sum_{k=1}^{5} a_k x^k$ has to be presented. The purpose is to learn a mapping from the coefficients a_k to the MP parameter $\boldsymbol{w}$. The error is the distance between the true 5-th order polynomials and the generated trajectories by the output VMP parameters. As shown in Fig. 4.5, we add MDN into the comparison. In total, we conducted 60 experiments. In each experiment, we sampled 30 random coefficients for training, and 20 were for testing.

For MDN, we select the most probable MP parameter as the output. Except for the GPR with DPKs, MDN outperforms all other methods.

4.4.2 Random Obstacles Avoidance

A 2D obstacle avoidance experiment showed in Fig. 4.4 has also been conducted in Zhou and Asfour (2017) as well as in Ewerton et al. (2015). In order to show whether the methods can scale to more complex tasks, we placed three obstacles randomly in the 2D space and searched for the collision-free trajectories with random starts and goals. To collect demonstrations, a person drew curves connecting random starts and goals without collisions with three

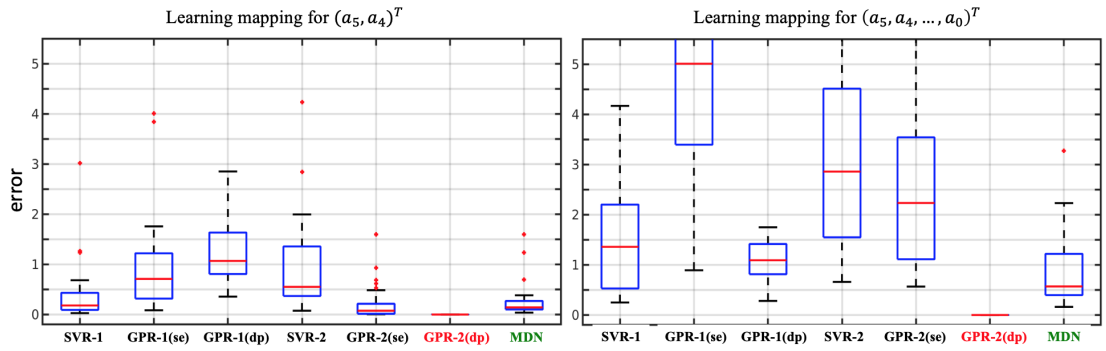

Figure 4.5: Polynomial fitting test with MDN .

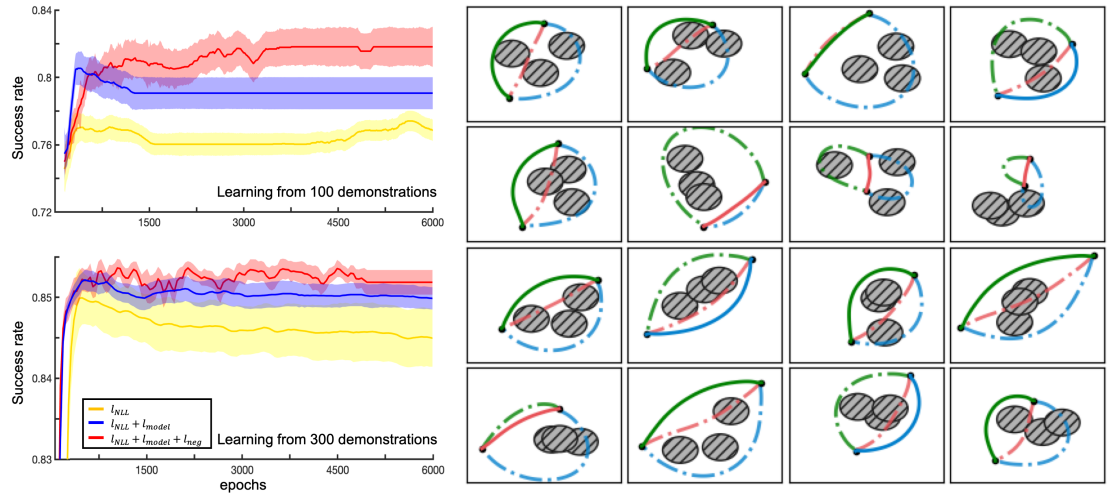

Figure 4.6: **Left-Top:** The MDN is trained on a dataset with 100 demonstrations. The best performance is approx. 82% success rate. **Left-Bottom:** The MDN is trained on a 300 dataset. The best performance is approx. 85% success rate. **Right:** 16 examples with MDN generated trajectories show how MDN works.

randomly generated 2D balls on a tablet. Without any instructions, the human demonstrations show multiple modes and models.

The success of the task execution requires that the generated trajectory connects the start and goal without any collision with randomly placed obstacles.

Due to the task complexity and existence of the multiple modes and models, previous approaches could not achieve acceptable results. With a 100 dataset, TP-GMM has only a success rate of about 45% with five local frames (three for the obstacles, two for the start and the goal) and five mixture components for the GMM. Both one-step and two-steps methods with SVR and GPR perform worse with a success rate that is less than 30%.

For MDN, we assume that there are three mixture components $K = 3$. To extract the latent feature (blue box in Fig. 4.3), we introduced three separate network branches for three obstacles' locations. Each branch takes the position of one obstacle, the start and the goal of the trajectory as the input and outputs a hidden feature vector. Three hidden feature vectors are then concatenated to one vector, which is used as input by the MDN. The whole MDN is trained in an end-to-end manner.

During testing, we select the most probable MP from the output distribution. For each number of training data, 30 experiments are conducted for 30 different training datasets randomly chosen from the collected demonstrations. In order to utilize the failure cost function, after each 100 training steps, the MDN is evaluated on an evaluation dataset to produce failed samples. To avoid increasing data, we considered only the recent 3000 failed samples.

The results are shown on the left side of Fig. 4.6. With 100 demonstrations, the MDN with both the entropy and failure cost functions ($l_{NLL} + l_{model} + l_{neg}$) achieves about 82% success rate. The performance is improved further to 85% with 300 demonstrations. With 100 demonstrations, the entropy MDN with the failure cost function ($l_{NLL} + l_{model} + l_{neg}$) achieves the best result and the entropy MDN ($l_{NLL} + l_{model}$) is better than the original MDN (l_{NLL}). Their difference is decreasing with the increasing number of demonstrations.

On the right side of Fig. 4.6, 16 testing samples are shown. The colorized solid curves are generated by the most probable mode (MP parameter) given by the MDN, and the transparent dashed curves correspond to the other two modes, which are not selected by the MDN with relatively small output mixing coefficients. Three different modes are shown with three different colors: The green curves bend towards the top; The blue curves bend towards the bottom; The red curves connect the start and the goal directly. The MDN accomplishes the task with two steps. One step is to separately update each mixture component branch to increase the success rate of their modes. The other step is to adjust the mixing coefficients output to select the mode, which has the highest chance of accomplishing the task.

4.4.3 Docking Problem

In this experiment, the MDN should generate a curve connecting two docking stations without colliding with them. This experiment has been used often by Calinon et al. (see Calinon et al. (2013); Calinon (2016)) to show the strength

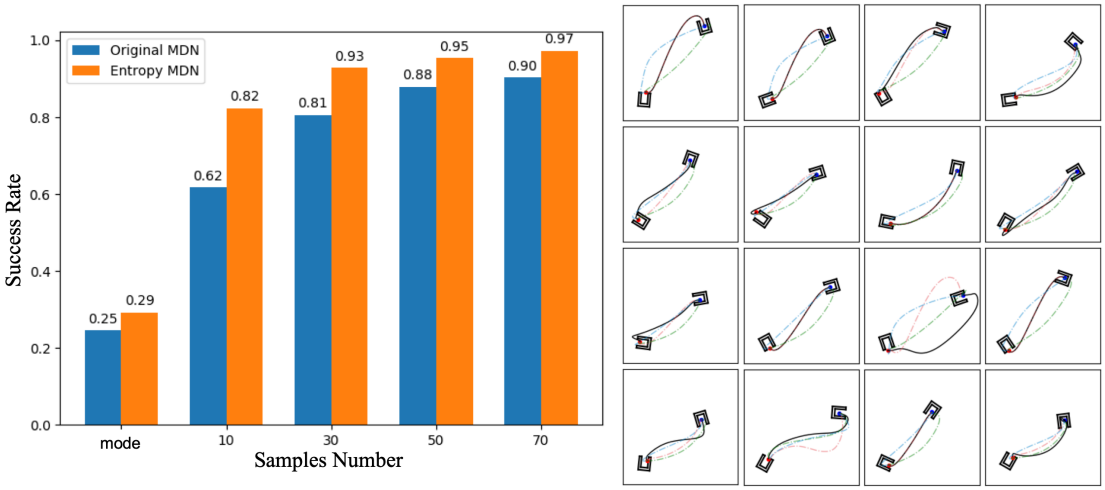

Figure 4.7: **Left:** The entropy MDN outperforms the original one for any number of samples. **Right:** The black solid curves are the sample drawn from the MDN output distribution. The other dashed curves denote the modes of three Gaussians.

of TP-GMM. We extend the experiment to allow the random orientation of the start and random position and orientation of the goal. The successful task execution requires that the generated trajectory has no collision with the docking stations and connects the start and the goal. In comparison, both TP-GMM and MDN are trained on 100 training data and tested on another 100 dataset. For TP-GMM, we associated two frames to the start and the goal separately. The number of components is determined based on the performance. TP-GMM shows a success rate smaller than 10%. For MDN, the number of components was 3. If the mode is selected, where the MDN serves as a deterministic model, the entropy MDN achieves a 29% success rate.

However, in this experiment, we need an accurate trajectory to guarantee that there is no collision when leaving the start or approaching the goal. Hence, the mode of the output GMM does not work very well. In order to improve the performance, we drew multiple samples and selected the best one. From the left diagram in Fig. 4.7, we see that the performance is improved with the increasing number of samples. In addition, the results show that the entropy MDN always has a better performance than the original MDN. On the right side of the Fig. 4.7, 16 samples are shown. The black curves indicate the result given by MDN. The dashed curves denote the mode of three Gaussian components. In several cases, the selected samples coincide with the mode. In most of the cases, the successful samples are obtained by taking a small change of one

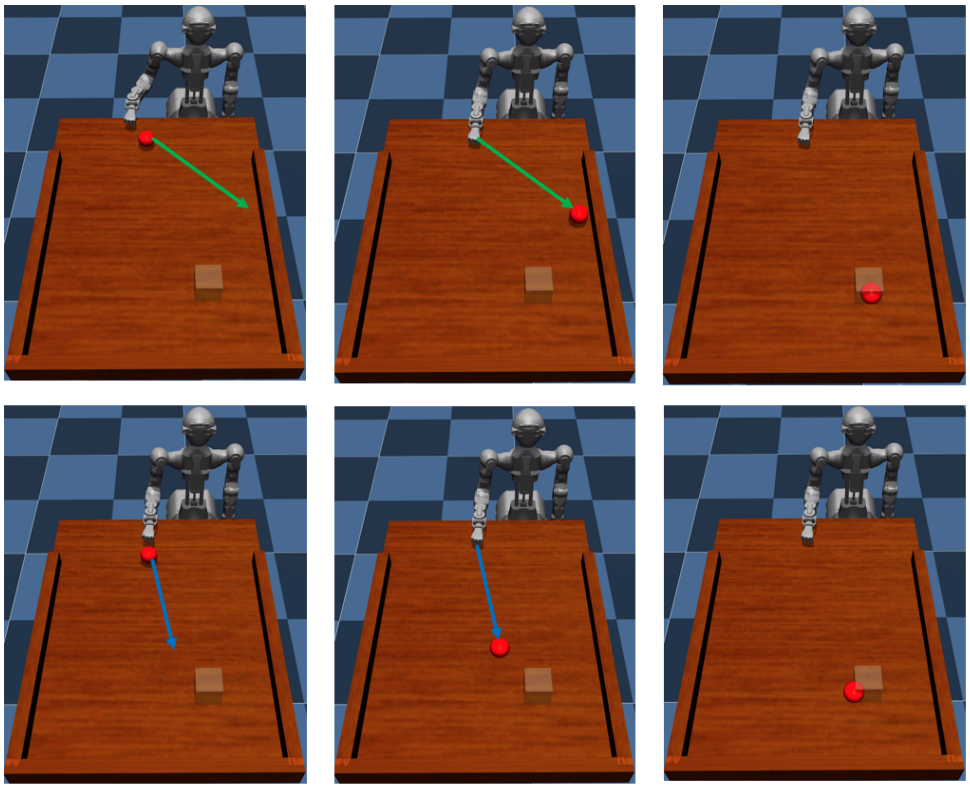

Figure 4.8: In this experiment, the desired final ball location is the input of the MDN, which is denoted by a transparent box. **Top**: the robot hits the ball from its right side, and the ball bounces off the border and stops at the target. **Bottom**: the robot hits the ball directly towards the target.

of the three modes. However, there are still a few cases, in which the generated trajectories are different from the three modes given by the MDN.

4.4.4 Hit Ball Experiment in Simulation with ARMAR-6

In this experiment, the robot hits the ball with its fist. After being hit, the ball slides on the table and stops at some locations (see Fig. 4.8). The final location of the ball on the table is the task parameter query. The purpose is to generate an appropriate robot motion to hit the ball and let the ball stop at a specific location. This experiment is conducted in the MuJoCo simulator (Todorov et al. (2012)) with the model of the humanoid robot ARMAR-6 (Asfour et al. (2018)).

For the demonstrations, we use a random trajectory generator based on a 5-th order polynomial, with which the position and velocity at the end of the

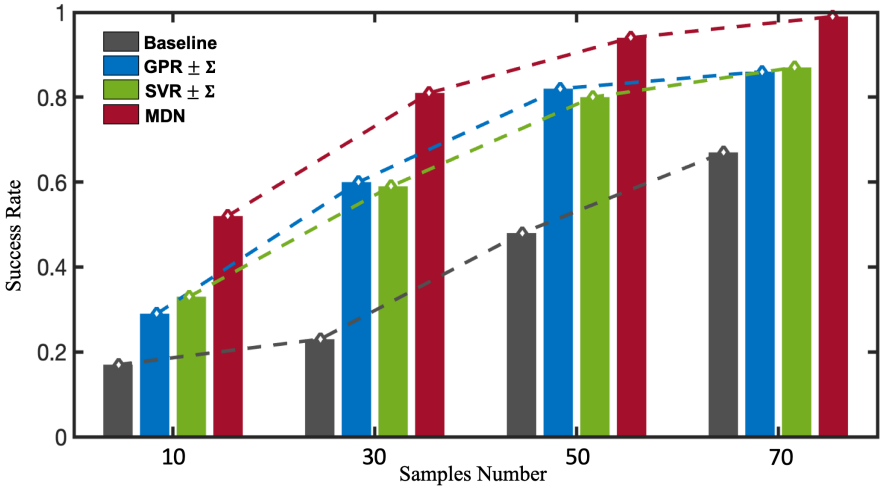

Figure 4.9: The result of the hit ball experiment shows that the MDN (red) outperforms the baseline (gray), GPR$\pm\Sigma$ (blue) and SVR$\pm\Sigma$ (green)

trajectory can be specified. The initial ball location on the table is fixed. The velocities, when hitting the ball, are randomly sampled from a uniform distribution. With different hitting velocities, the ball stops at different locations. The final locations of the ball $\boldsymbol{q}$ and the MP parameters $\boldsymbol{w}$ are collected in a training dataset.

As shown in Fig. 4.8, the ball can bounce off the borders of the table, which realizes multiple modes for one specific target location in the collected demonstrations. The table is $260\,\mathrm{cm} \times 200\,\mathrm{cm}$ big and the ball has radius $5\,\mathrm{cm}$. Successful task execution is that the ball is no more than $10\,\mathrm{cm}$ far from the target location.

We use $K = 3$ mixture components and train the MDN with 50 random demonstrations and test it on 100 new ball locations. If the most probable MP parameters are selected, the average success rate is around 15%. This poor performance might be because the task requires an accurate trajectory. With a random small perturbation of the MP parameters that correspond to the initial 50 demonstrations, the success rate drops rapidly. To improve the performance of this task, we drew several samples from the output distribution. In this case, a successful MP generalization means that there is at least one sample that leads to successful task execution. As shown in Fig. 4.9, increasing the number of samples improves the success rate.

In this task, the increasing number of samples helps because of two reasons. One trivial reason is the setup of the task, which allows successful task executions by chance: VMP guarantees that the robot hits the ball, and the table

borders limit the final ball locations. The other reason is that the MDN learns the correct distribution, which gives a high probability to the correct MP parameter, which is unfortunately not precisely the mode. Sampling from the correct distribution has a bigger chance of finding the correct solution than directly selecting the most probable mode. In order to prove that the latter reason exists with MDN for this task, we consider a uniform distribution of the MP parameters as the baseline, whose interval is determined by the minimal and maximal components of the MP parameters, which correspond to the 50 demonstrations. Besides the baseline, we construct the Gaussian distributions by considering the GPR and SVR outputs as mean vectors and with a fixed variance matrix ($\Sigma = 0.01\boldsymbol{I}$).

As shown in Fig. 4.9, MDN outperforms other methods for all the sample numbers. For the baseline, it coincides with the intuition that its success rate is almost proportional to the number of samples because it does not learn from the demonstrations. GPR and SVR have better performances than the baseline because they draw the samples close to their output MP parameters. However, the samples drawn around their outputs are totally by chance because of the fixed variance matrix. In contrast, the MDN learns a relatively correct distribution output and achieves already a high success rate with a smaller sample number.

4.4.5 Throw Ball Experiment with ARMAR-6

To further evaluate our method, we let the robot throw a ball on a specific target. The arms of ARMAR-6 have 8 degrees of freedom (DoF) each. In order to simplify the task, only 4 DoFs are used, 2 of the shoulder joints, the elbow and the wrist joint. Other joints are excluded for this task without loss of generality (see the 4 DoF in Fig. 4.10). The demonstrations were conducted by the human using kinesthetic teaching. After learning the corresponding MP parameter for each demonstration, we speed up the motion to 1 second and set a fixed joint goal. The robot hand is controlled to open always at 0.55 second. Then we record the location of the ball when it drops on the ground. By fixing the goals and speed of the motions, the locations where the ball hits the ground are only dependent on the shape of the motions. In the experiment, we let the robot face the wall so that it can bounce the ball off the wall to the target.

We let the robot throw 50 times with different human demonstrations and randomly split the collected data into 30 for training and 20 for testing. We train an MDN ($K = 2$) on 30 demonstrations. For the testing, we only use the hit ground

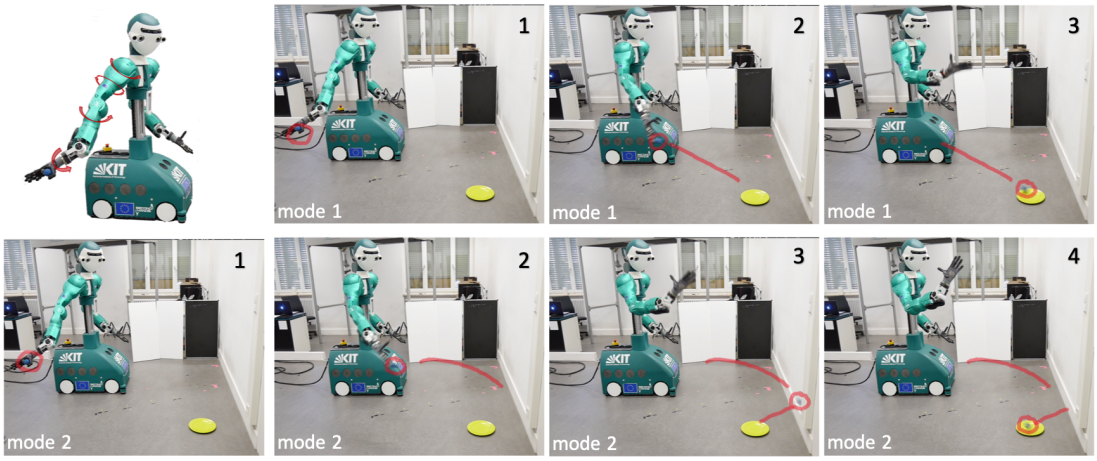

Figure 4.10: **Top-left**: Four DoFs are used for throwing the ball. **Top-right**: the robot throws the ball directly to the target. **Bottom**: the robot bounce the ball to the target off the wall.

locations of the other 20 demonstrations as task parameter queries, which guarantees that all the hit ground locations are reachable. During the testing, we place a plate on the ground to indicate the current query. The successful task execution is throwing the ball on the plate either directly or by bouncing it off the wall. In the experiment, only 2 out of 20 target hit ground locations were missed with 10 samples. In Fig. 4.10, for one specific task parameter query, we show how two MP parameters, which correspond to two different modes, result in different paths of the ball.

4.5 Conclusion

In this chapter, in the context of the two-steps methods, two approaches are presented to learn a mapping ω from the task parameter queries to movement primitive parameters.

One approach is to use a mixture of experts to model the mapping from task parameter queries to MP parameters. A *Leave-One-Out Expectation Maximization* (LOO-EM) algorithm is developed to learn this mixture of experts model. LOO-EM outperforms the original EM algorithm, which is easily stuck in local minima. Although models obtained with EM can fit all training data, they cannot represent the "correct" multi-modal functions as shown in Fig. 4.2. However, since LOO-EM requires training models for each data point, using LOO-EM to learn a mixture of experts model is time consuming, especially when the

demonstration dataset is large. Moreover, LOO-EM is a deterministic method, hence, cannot handle multiple modes.

To overcome these difficulties and be able to handle multiple modes and models at the same time, we propose using a probabilistic model, namely *Mixture Density Network* (MDN), which outputs a mixture of Gaussian distributions, to learn the mapping from task parameter queries to MP parameters. However, training MDN with the original *negative-log-likelihood* (NLL) cost can still cause mode and model collapse. Hence, we introduce an entropy cost function that forces the MDN to consider multiple modes and models. To further improve the training process, a failure cost is also introduced to avoid similar MP parameters that lead to failure. Experiments show that MDNs with new cost functions outperform previous methods and original MDNs trained with only NLL cost function.

5 Movement Primitive Adaptation and Control

In this chapter, the movement primitive parameters have been already determined by the generalization approaches described in Chapter. 4. The final step is to control the robot to follow the MP generated trajectory to accomplish the task. For this purpose, the task space tracking controller described in Fig. 3.5 can be used. However, for a contact-rich manipulation or human-robot interaction tasks, a trajectory tracking controller is not enough for successful task execution, because additional task parameters such as target force profiles or environment changes should also be considered.

In Chapter. 4, the task parameters for a specific task are represented in a fixed dimensional space. They can be explored with a limited number of demonstrations, which means that all the valid task parameters for the task can be determined by the interpolation of regression models trained on those demonstrations. In the obstacle avoidance example, if the number of obstacles is fixed, the problem can be solved by the MP generalization as described in Section. 4.4.2. However, if the number of obstacles is unknown, it is difficult to solve the problem with the MP generalization because the task parameters have a variable dimension, and we cannot use a limited number of demonstrations to explore all the possibilities.

The same statement applies to human-robot interaction tasks, where a human's activity cannot be predicted or covered by a limited number of demonstrations. For example, in a robot washing system where the robot's arm wipes the back of the human as in the case of an assistive bathing system (Zlatintsi et al. (2020)), the robot cannot predict how the human moves the back during task execution based on a limited number of demonstrations.

Unfortunately, such problems cannot be solved directly with a general approach like in Chapter. 4. Expert knowledges are required for the solutions. For example, using *Via-points Movement Primitive* (VMP), described in Chapter. 3, in obstacle avoidance tasks, we need an expert such as a local task space via-points

planner that outputs a sequence of via-points to avoid obstacles.

In many applications, multiple agents are involved such as in human-robot interaction tasks or bimanual manipulation tasks. For such tasks, in which, motions of one agent depend on those of other agents, or the agents act in a leader-follower manner, we propose a simple leader-follower framework, called *Coordinate Change Movement Primitive* (CC-MP), which adjusts the follower's motion based on the leader's behavior. Based on CC-MP, a robot washing system is developed in the context of the assisitve bathing system mentioned before.

The work of CC-MP has been published in Zhou et al. (2016a) and Zhou et al. (2016b). The robot washing system based on CC-MP has been published in Dometios et al. (2018).

In this chapter, we also realize a compliant control during task execution based on movement primitives. It is difficult to control a robot in a compliant way and guarantee tracking accuracy of a given trajectory at the same time. In order to solve this problem, we propose learning a force predictive model to guide an adaptive controller to change its stiffness. With this control diagram, the robot can adapt to external perturbations in a compliant way and track the given trajectory accurately during task execution. We evaluate this method with a wiping table task.

5.1 Coordinate Change Movement Primitive (CC-MP)

As the name reveals, the *Coordinate Change Movement Primitive* (CC-MP) changes the coordinate frame of the task space MP during task execution. The MP parameter $\boldsymbol{w}$ is learned in a local coordinate frame. Hence, it can adapt to the coordinate transformation afterward. This strategy is similar to the TP-GMM (see Calinon et al. (2013) Section. 2.2.2), where several GMMs are learned in the local frames and adapt to their changes during the task execution.

CC-MP also encodes the movement of the leader if it is possible and necessary. In human-robot interaction tasks, the human is viewed as the leader. The human motion, however, is not controlled by the robot. Hence, it is not necessary to encode human motion with a movement primitive. In bimanual manipulation tasks, in contrast, it makes more senses to encode the motions of both hands, where one serves as the leader, and the other is the follower.

It is task-specific to determine which agent is the leader or the follower. However, uncontrollable agents with difficulty to describe behaviors should be considered as the leader.

all the uncontrollable agents should always be the leader.

Before training the follower's movement primitive, the demonstrated trajectory $\boldsymbol{y}^f_{G,d}(x)$ is transformed in the local cooridnate of the leader such that:

$$\boldsymbol{y}^f_{L,d}(x) = T(\boldsymbol{y}^l_G)\boldsymbol{y}^f_{G,d}(x), \tag{5.1}$$

where the subscript L and G are for the local and the global coordinates separately. The superscript l and f denote the leader and the follower. $\boldsymbol{y}^f_{L,d}(x)$ is learned by the movement primitive. During the task execution, the generated motion of the learned $\boldsymbol{y}^f_L(x)$ is transformed back to the global coordinate, namely

$$\boldsymbol{y}^f_G(x) = T^{-1}(\boldsymbol{y}^l_G)\boldsymbol{y}^f_L(x). \tag{5.2}$$

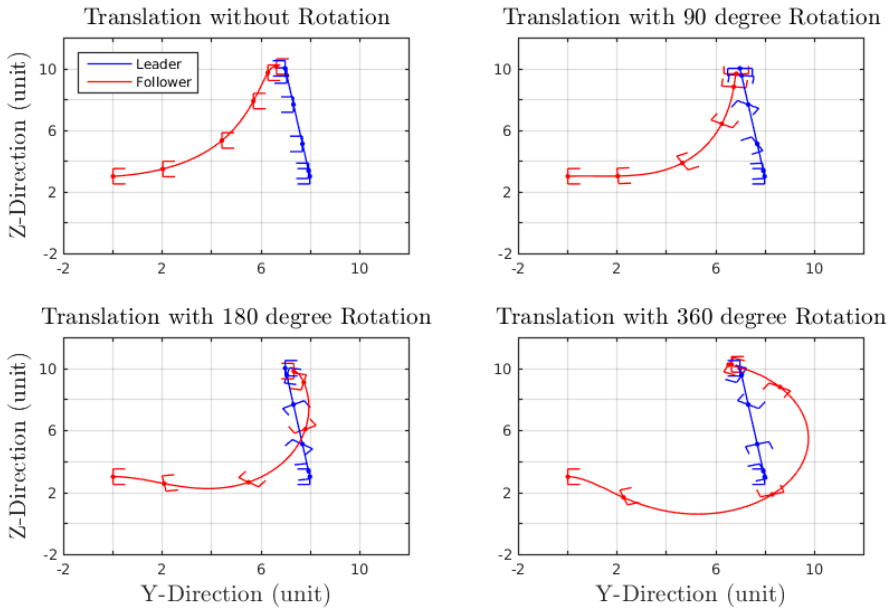

Figure 5.1: The Docking problem is solved by CC-MP, where the red end-effector should approach the blue end-effector with a particular direction. The motion of the red end-effector is learned and encoded with a movement primitive in the local frame of the blue end-effector. During task execution, its global motion is obtained by transforming its local motion back to the global frame.

In Fig. 5.1, one example is shown, where the red end-effectors should approach the blue end-effectors with a specific orientation. Since it is similar to a docking

task of spacecraft to a space station, we call it a docking problem. The blue end-effector is not controllable. Hence, it is regarded as a leader. The motion of the follower (the red end-effector) is learned with the MP. With the previous process, CC-MP simply adapts to the motion of the blue object without changing any parameters. It is difficult to use the MP generalization approaches described in Chapter. 4 for the docking problem because it is hard to predict how the blue target object moves during the task execution.

In the robot wiping task described in the following, a hierarchical leader-follower structure is used, where there are multiple leaders and followers. The leader of one follower can be a follower of the others.

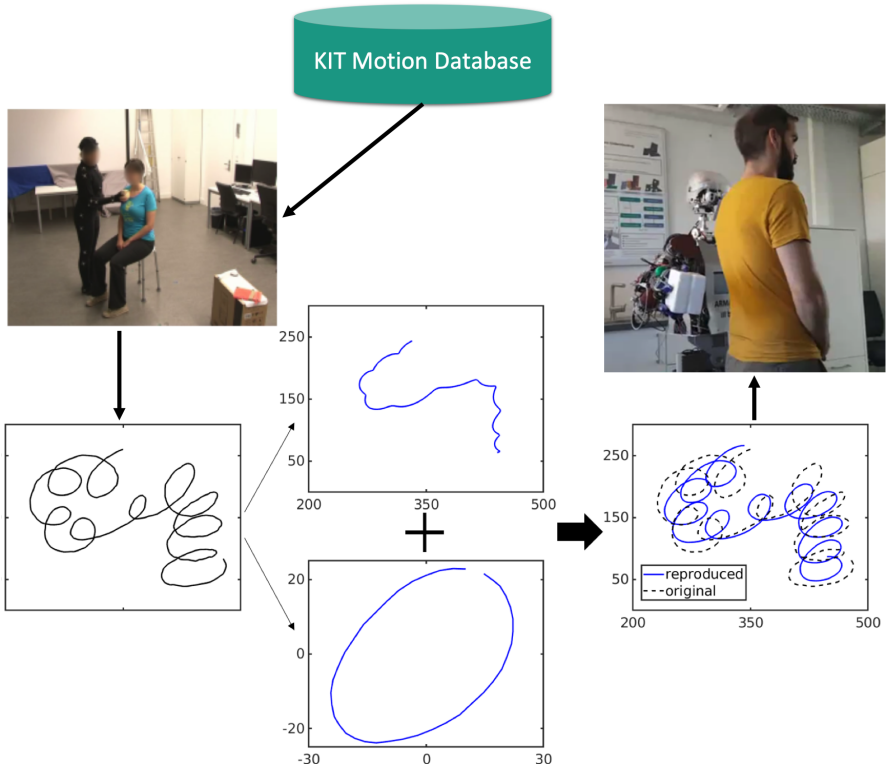

Figure 5.2: Learning wiping motions with CC-MP. **Left:** We extract a human wiping motion from the KIT motion database. **Middle:** The trajectory is separated into the linear and periodic part. **Right:** After learning CC-MP, we can reproduce the original wiping motion and also change the wiping pattern (see Fig. 5.3).

5.1.1 Learning Adaptive Robot Wiping

The robot wiping task is one of the essential applications for service robots (Zlatintsi et al. (2020)). The wiping skill is used in multiple different tasks such

as cleaning the table, mopping the floor, or helping washing the human body. In all these tasks, the robot should be able to generate different wiping patterns and adapt the wiping motions to the changing environment. In the following, we describe how CC-MPs are used to solve the robot wiping task.

As shown in Fig. 5.2, the demonstrations are human motion recordings in the *KIT motion database* described in Mandery et al. (2015). The 3D wiping motion is projected on a 2D surface by eliminating the dimension with the lowest variance. Since the curvature of the wiping surface usually determines the third dimension, it is not learned directly from the demonstration. There are two different ways to learn this wiping motion. We either directly learn the whole motion with one single MP or first separate it into discrete and periodic patterns. The latter solution is much more flexible than the former one. The periodic wiping pattern can be combined with different wiping directions (discrete parts) to generate different wiping motions according to the user's preference or commands.

To extract the periodic wiping pattern, the *moving average* algorithm is first applied on the trajectory, whose result is the discrete part as shown in the upper-middle plot of Fig. 5.2. The moving average algorithm uses a time window to scan the whole trajectory. For each location of the time window, an average value of all the points in the time window is calculated. With a particular size of the time window, we can eliminate the local features of the trajectory such as the periodic pattern in the wiping motion.

By subtracting the discrete part from the original trajectory, we obtain the periodic part. To get one period, we applied the *Fourier transformation* to the periodic part to find its frequency. By cutting the periodic part with the frequency and averaging all the segments, we can obtain one period, as shown in the bottom-middle plot of Fig. 5.2. Then, the leader MP is learned from the discrete trajectory, and the follower MP is learned from the periodic pattern. During the execution, the Eq. 5.2 is used to determine the wiping motion. As shown in the right plot of Fig. 5.2, the reproduction result approximates the original trajectory.

Encoding Periodic Motions

In Section. 2.1.3 and Chapter. 3, we described how to represent discrete motions with DMP or VMP. For a periodic motion, as in Ijspeert et al. (2013), the

transformation system of DMP is as follows:

$$\tau \dot{v} = K(g - y) - Dv + rf(x), \tag{5.3}$$

where r is the amplitude of the periodic motion. Instead of *squared exponential kernels* (SEK), *cosine exponential kernels* (CEK) are used such that

$$\boldsymbol{\psi}_i(x) = exp(h_i(cos(x - c_i) - 1)), \tag{5.4}$$

where h_i is a constant called *concentration* parameter. The CEK is closely related to the von Mises distribution (Ijspeert et al. (2013)), a periodic version of the Gaussian distribution. Instead of a decay system, we use a linear increasing canonical system $\tau x = at$ or $\tau \dot{x} = a$ based on an assumption that a periodic motion never ends. With DMP, we can adapt the learned periodic motion to different amplitudes and speeds.

For VMPs, the elementary trajectory is replaced by a single anchor point g, hence,

$$y(x) = g + rf(x), \tag{5.5}$$

where r is the amplitude. The same kernel functions CEKs are used for the linear regression model $f(x) = \boldsymbol{\psi}(x)^T \boldsymbol{w}$. As DMP, VMP can also adapt to different speeds and amplitudes.

Wiping Trajectory Adaptation

As shown in Fig. 5.3, the wiping motion can be adjusted by changing the hyperparameters of the leader's or follower's MP. For example, the change of the *temporal factor ratio* (TFR), namely τ_f/τ_l, where τ_f is for the follower and τ_l is for the leader, results in different wiping motions as shown in two bottom-right plots.

In addition to different wiping patterns, the leader-follower framework is also used to adapt to the moving or curved wiping surface, as shown in Fig. 5.4 and Fig. 5.5. The wiping surface is regarded as the leader. The wiping motion, as a whole, is the follower and encoded in the local coordinate defined by the wiping surface.

Based on CC-MP, a vision based robot wiping system is proposed in Dometios et al. (2018). As shown in Fig. 5.6, a camera system obtains the points cloud of the wiping surface. By analyzing those points, the surface norm vector around the current local wiping region is extracted. Based on this norm vector and

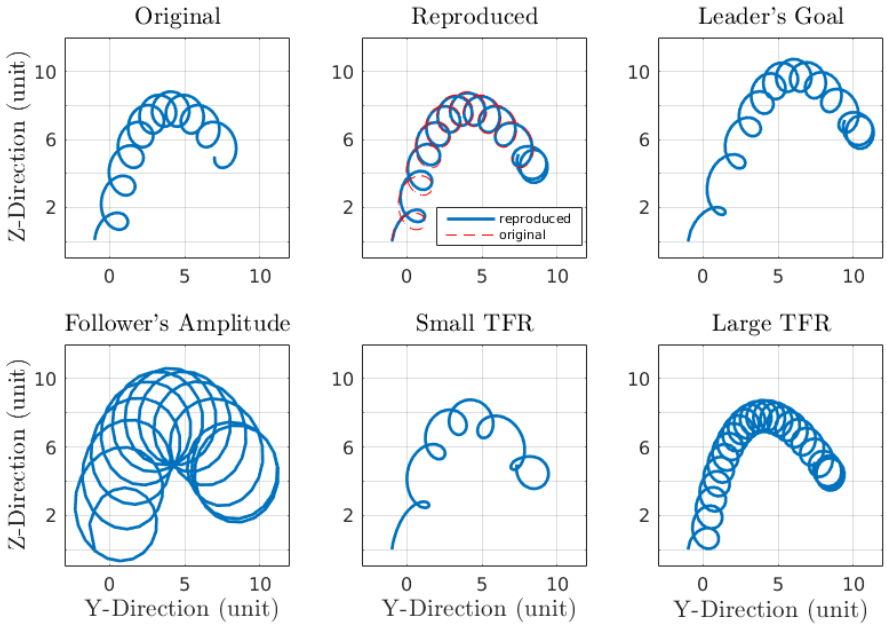

Figure 5.3: Wiping pattern is encoded with CC-MP. By changing the hyper-parameters of the leader or follower movement primitives, we can generate different wiping patterns. For example, as shown in the last two plots, changing the *temporal factor ratio* (TFR) results in different wiping styles.

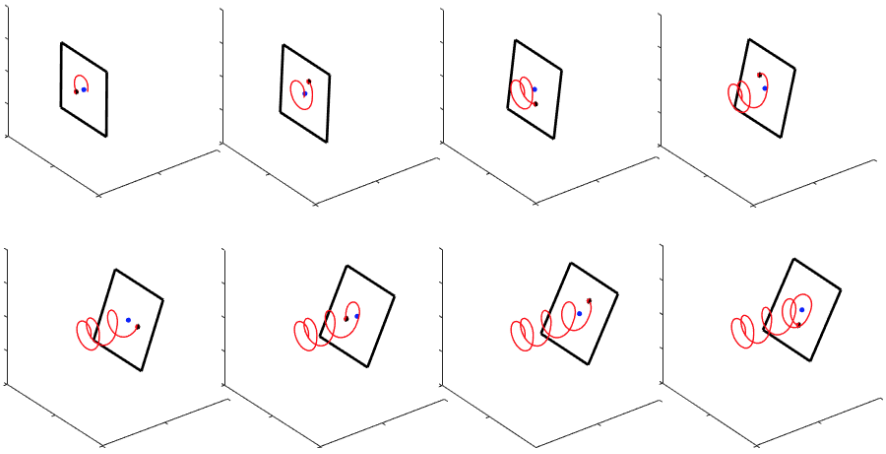

Figure 5.4: CC-MP adapts to the moving wiping surface. A surface is moving and rotating. The learned periodic wiping motion is adapted to the movement of the surface.

a perception based controller, the generated trajectory of the leader MP is adjusted, and the coordinate frame of the periodic follower pattern is changed. With this setup, the wiping motion adapts to the moving or curved surface. For

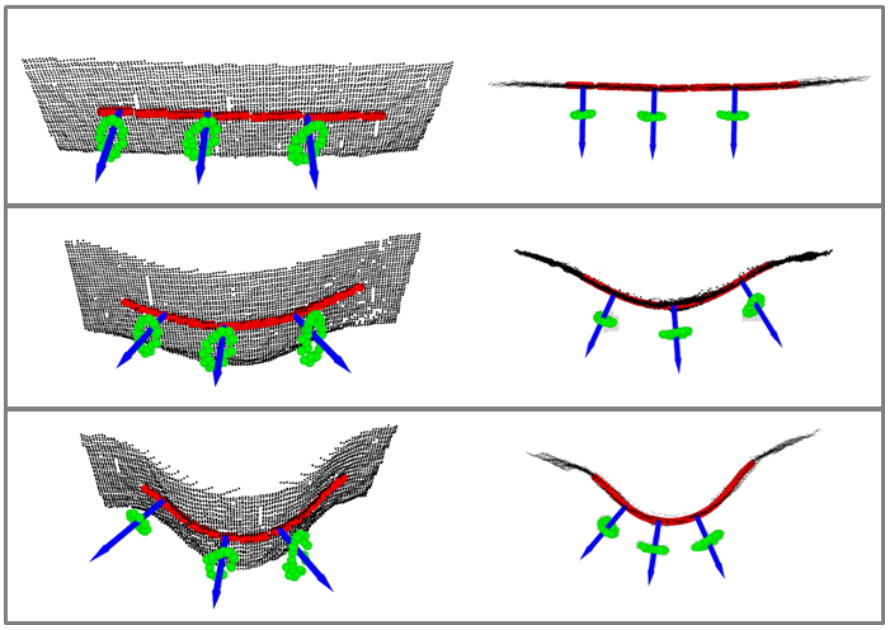

Figure 5.5: CC-MP adapts to the curved surface. The red lines denote the discrete part of a wiping motion. The green dots represent the periodic part. The blue arrows indicate the target force into the wiping surface. When the surface is deformed, which is detected by the vision system, the wiping motion is adapted to the changing curvature of the surface.

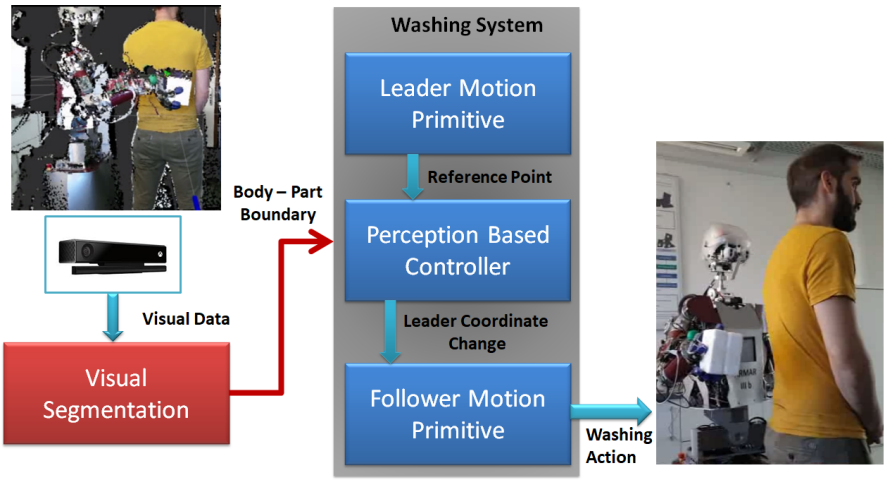

Figure 5.6: A vision based CC-MP wiping system. This work has been published in Dometios et al. (2018).

the perception-based controller, we refer the readers to Dometios et al. (2018) for detail.

As shown in Fig. 5.7 and Fig. 5.8, ARMAR-III (see Asfour et al. (2006)) accom-

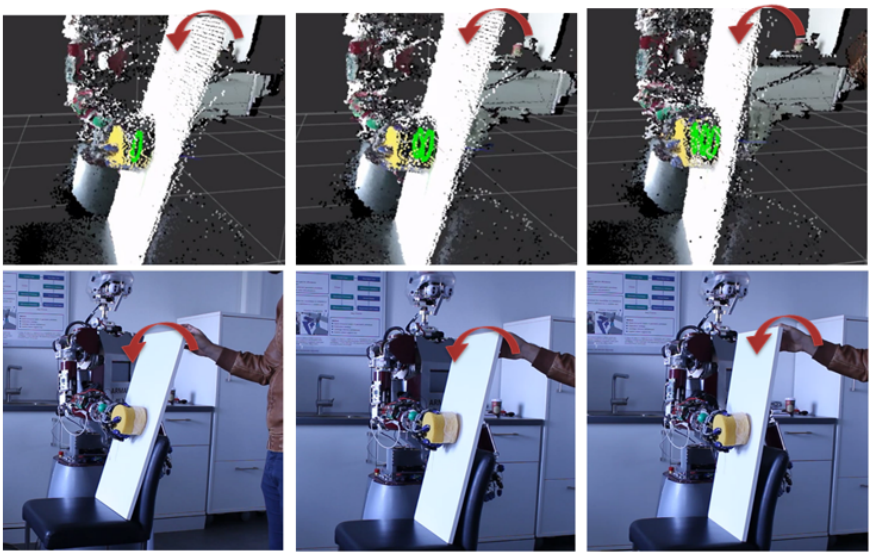

Figure 5.7: ARMAR-III is wiping a moving whiteboard. The **top row** shows the points cloud provided by the kinect camera.

Figure 5.8: ARMAR-III is wiping the back of a human. For the safety reasons, we keep a distance between the human and the robot. The **second and fourth columns** show the points cloud from the kinect camera.

plished the wiping tasks on a moving whiteboard and human back.

Wiping Force Adaptation

Similar to the coupled DMP presented in Gams et al. (2010), a coupling term is added to the follower MP to generate force on the wiping surface. In Zhou et al. (2016b), we use DMP in the CC-MP framework, hence, the coupling term

is added to the output acceleration of the DMP. For VMPs, we can have

$$y(x) = h(x) + f(x) + c(x), \tag{5.6}$$

where the coupling term $c(x)$ denotes a position offset from $h(x) + f(x)$ for a desired force. Like the method described in Gams et al. (2010), the coupling term can be iteratively updated. Since the follower's movement primitive is used to encode the wiping pattern and is learned in the leader's local frame, a mapping from the 2D position in this local frame to the position offset in the direction perpendicular to the wiping surface can be trained with an incremental learning algorithm. In Zhou et al. (2016b), a *Locally Weighted Regression* (LWR) model is used to learn this coupling term, since it allows the incremental learning during the task execution. The coupled VMP can be written as

$$y(x) = h(x) + f(x) + c(\boldsymbol{s}), \tag{5.7}$$

where $\boldsymbol{s}$ is the current projected position on the surface, and

$$c(\boldsymbol{s}) = \frac{\sum_{i=1}^{N} \psi_i(\boldsymbol{s})\boldsymbol{w}_i}{\sum_{i=1}^{N} \psi_i(\boldsymbol{s})}, \tag{5.8}$$

where N is the number of the kernel functions and $\psi_i(.)$ is a 2D exponential function:

$$\psi_i(\boldsymbol{s}) = exp(-h_i||\boldsymbol{s} - \boldsymbol{s}_i||_2), \tag{5.9}$$

with h the normalization factor and $\boldsymbol{s}_i$ a point on the 2D grid. The training data is collected during the wiping execution. Combined with a feedback force controller, the robot can gradually adapt to the curvature of the surface while meeting the desired target force.

5.1.2 Bimanual Manipulation with CC-MP

Another application of CC-MP is to couple both hands of the robot in bimanual manipulation tasks. In this case, one of the hands is regarded as the leader, and the other one is the follower.

For holding a rigid object, the relationship between two hands does not change. By only learning the leader's movement, both hands are automatically coupled. In Fig. 5.9, the robot put down a heavy guard in collaboration with a human. The leader VMP learns the motion of the left hand. Since the right-hand does not move in the left-hand coordinate, the follower VMP encodes a static po-

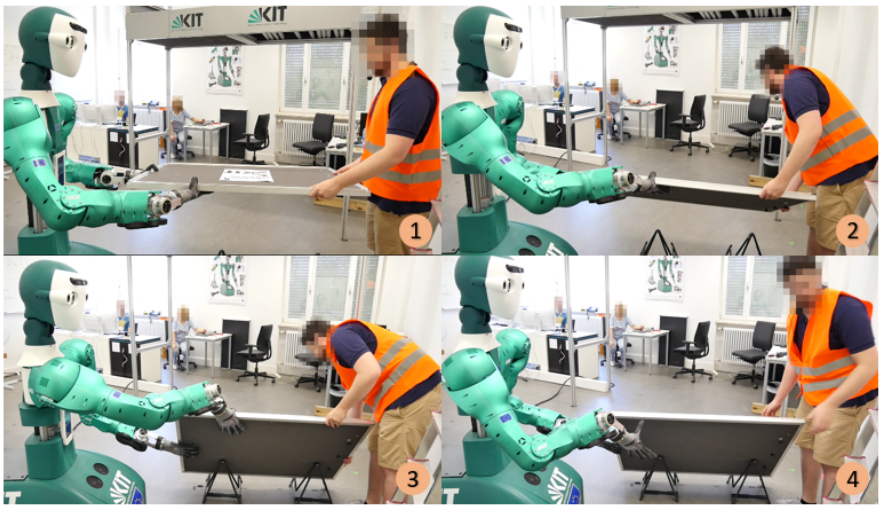

Figure 5.9: ARMAR-6 is placing down the board bimanually with the human cooperation.

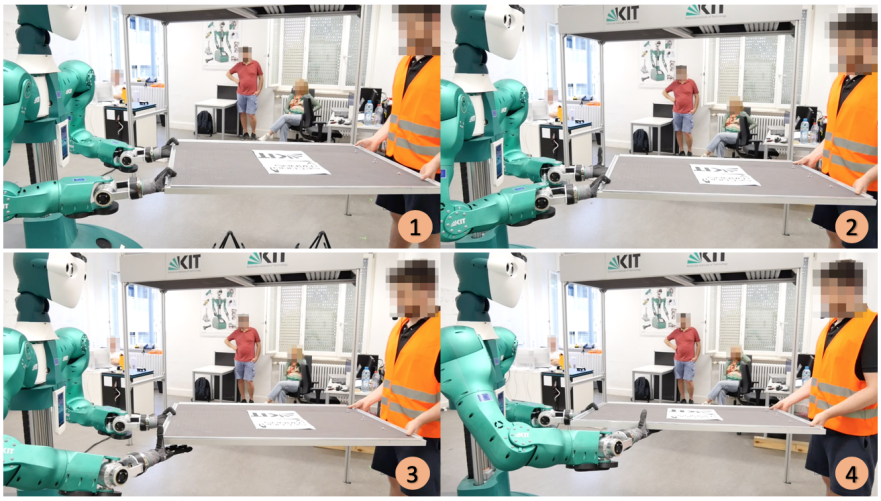

Figure 5.10: ARMAR-6 is regrasping the board with the human cooperation.

sition. With CC-MP, only one motion is to be learned for this guard lowering task. In this case, CC-MP serves as a constant offset between the left and right hand.

In Fig. 5.10, the robot regrasped the guard while manipulating is jointly with a human. In this bimanual task, the leader VMP is learned by observing the motion of the left hand during the human demonstration. The motion of the right hand described in the leader's coordinate frame is obtained with Eq. 5.1 and learned by the follower VMP. The via-points are integrated to guarantee that the hand does not collide with the guard. In the case when two separate

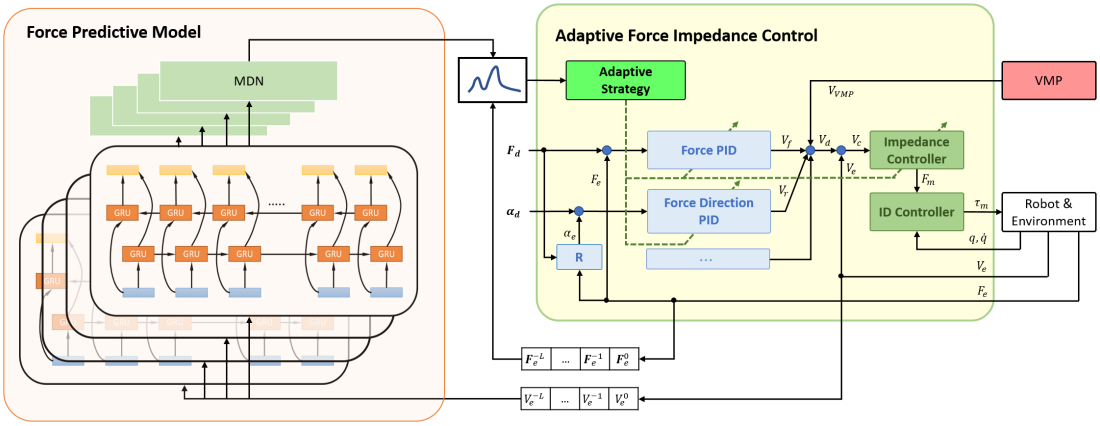

Figure 5.11: **Left:** force predictive model **Right:** adaptive force control

VMPs are used and coupled with one canonical system, we need to change both VMPs if the initial grasping positions on the guard change. With the CC-MP, we only need to change the leader's VMP. CC-MP simplifies the adaptation to the new task requirements.

5.2 Compliance Adaptation in Contact-Rich Manipulation

As described in Chapter. 3, the task space VMP control framework (see Fig. 3.5) is a standard impedance control framework. Most of the tasks mentioned before are executed using this controller. The standard controller only takes the position and velocity feedback into consideration to track the VMP generated trajectories. For a contact-rich manipulation, however, a position tracking controller is not enough. In Fig. 5.6, the robot uses the visual feedback to realize wiping on a curved and moving surface. Here, we consider how to use force sensor feedback to realize compliant behaviors and follow desired force profiles. We still focus on the wiping task. However, the method is not limited to this task.

It is not trivial to realize compliant behavior for a contact-rich manipulation. For example, in the wiping task, if the friction of the surface is not negligible, a more stiff impedance controller is required to realize an accurate position tracking for the VMP generated wiping motion. A high stiffness, however, conflicts with the desired compliant behavior.

One solution is to use an adaptive controller, which changes its stiffness on-

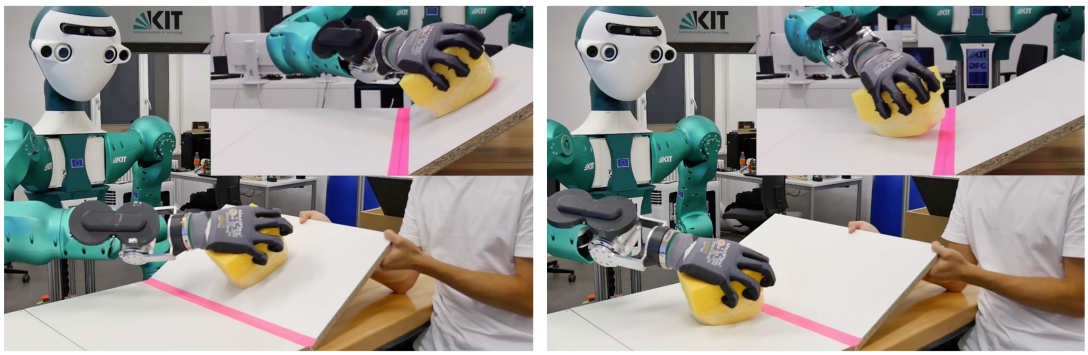

Figure 5.12: The rotational velocity $\boldsymbol{v}_r$ aligns the wiping tool with the wiping surface.

line according to the requirements. During the wiping without external disturbances, its stiffness is high to guarantee the tracking accuracy on the wiping surface with negligible friction. Once an unexpected disturbance is detected, its stiffness decreases to realize a compliant reaction and avoid the damage of itself or the environment.

We assume that an unusual force profile corresponds to an unexpected disturbance during the task execution. To detect this unusual force profile, we learn a force predictive model that can predict the normal force profile during correct task execution. By comparing the current force profile with the predicted one, we can tell whether the force profile is unusual or not, then, guide an adaptive controller to realize compliant behaviors.

As shown in Fig. 5.11, the result control diagram consists of two parts: an adaptive force controller and a force predictive model.

5.2.1 Adaptive Force Control

A standard impedance controller is used to output a general force F_m based on the desired velocity $\boldsymbol{v}_d$ and the current velocity $\boldsymbol{v}_e$. With the task space inverse dynamic, the joint torque targets τ_m are calculated and passed to a lower level torque controller for the motor commands.

For the wiping task, the desired velocity $\boldsymbol{v}_d$ consists of $\boldsymbol{v}_{VMP}$ and $\boldsymbol{v}_{force}$. The VMP velocity $\boldsymbol{v}_{VMP}$ is to track the desired motion trajectory. $\boldsymbol{v}_{force}$ is separated to its translational velocity $\boldsymbol{v}_f$ and rotational velocity $\boldsymbol{v}_r$. A PID force controller calculates the translational velocity $\boldsymbol{v}_f$ with the desired force F_d and the current force F_e in the z-direction perpendicular to the wiping surface. This direction

is established in the local frame of the robot's end-effector. An another PID controller calculates the rotational velocity $\boldsymbol{v}_r$ with the desired angle α_d and the current angle α_e to rotate the hand to align the wiping tool with the wiping surface (see Fig. 5.12).

The structure of the adaptive controller is shown in Fig. 5.11 (right). The adaptive strategy (green box) changes the stiffness of all PID controllers mentioned before, according to the result given by the force predictive model. With a low stiffness, the controller is compliant and cannot follow the VMP generated trajectory well. With high stiffness, the controller guarantees a high tracking accuracy but is not compliant. When adjusting the stiffness K_p, the adaptive controller also adjusts derivative K_d and integral K_i gains in the same way.

For the wiping task, we consider two different modes of the controller.

- **Normal mode**: the normal mode is for the case without external disturbance. The stiffness K_p is calculated by

$$K_p(t) = min(K_p(t_0) + \beta_p(t - t_0), K_{p,max}), \tag{5.10}$$

  where where t_0 is the start time point when the controller is switched to the normal mode and the constant β_p decides how fast the stiffness should reach its maximum $K_{p,max}$.

- **Adaptive mode**: the adaptive mode takes over once disturbances are detected. The stiffness K_p is determined by

$$K_p(t) = max(K_p(t_0) - \alpha_p(t - t_0), 0), \tag{5.11}$$

  where t_0 is the start time point when the controller is switched to the adaptive mode and the constant α_p decides how fast the stiffness should be decreased to zero. Once the stiffness is zero, the zero torque controller only compensates the gravity.

5.2.2 Force Predictive Models

The force predictive model is shown in Fig. 5.11 (left). The output of the model is compared with the force profile during the task execution. If they are different, the adaptive strategy decreases the stiffness of the PID controllers. Otherwise, it keeps the high stiffness.

Bi-directional Gated Recurrent Units (Bi-GRU), described in Schuster and Paliwal (1997), are used to model the mapping from a velocity profile to a force profile.

Bi-GRU has been proved to be useful for learning the mapping from language to motion (see Plappert et al. (2018)). Other recurrent neural networks can be used here to model this mapping. Bi-GRU takes the velocity profile defined in the local coordinate as input and outputs a sequence of the force values. As shown in Chapter. 4, a mixture density network can be used to encode multiple modes in the distribution. Here, we again use a mixture density network connected to the Bi-GRU for the case when the force distribution contains multiple modes. To train this network, we minimize the negative-log-likelihood (NLL) of the observed target force profile:

$$l_{NLL}(\boldsymbol{\Theta}) = -\sum_{i=1}^{N} log\Big(\sum_{k=1}^{K} \pi_k(\boldsymbol{v};\boldsymbol{\Theta})\mathcal{N}\Big(\boldsymbol{F};\boldsymbol{\mu}(\boldsymbol{v};\boldsymbol{\Theta}),\boldsymbol{\Sigma}(\boldsymbol{v};\boldsymbol{\Theta})\Big)\Big), \tag{5.12}$$

where $\boldsymbol{\Theta}$ denotes the parameters of the whole network. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the result functions for both mean and variance. $\boldsymbol{v}$ and $\boldsymbol{F}$ are sequential velocity and force profiles, which are collected by the sliding window with a fixed sequence length L. The data number N is determined by both the sliding window size and the length of the whole observed time series data.

In Fig. 5.11, the force and velocity profile are stored in a queue. The last L velocities are taken as input of the predictive model, which outputs a Gaussian mixture distribution of the force profile. Based on this distribution, the probability of the L steps force profile is calculated. The logarithm of this probability is called score, and its integral is called abnormal score. The integral avoids the reaction to short disturbances or noisy force sensor measurement.

Learning Multiple Force Predictive Models

It is not expected that the learned predictive model generalizes well to any cases because it is difficult to collect enough data that covers all different situations. However, in reality, many contact-rich manipulations, including the wiping task, face different situations, where one force predictive model is not enough. In order to overcome this problem, we propose an approach to learning and creating a mixture of experts.

1. In the beginning, no predictive model exists. We let the robot execute the task with a high stiffness controller. The velocity and force profiles are collected to train the first model;
2. Once more than one models exist, we let the robot execute the task with the control framework shown in Fig. 5.11;

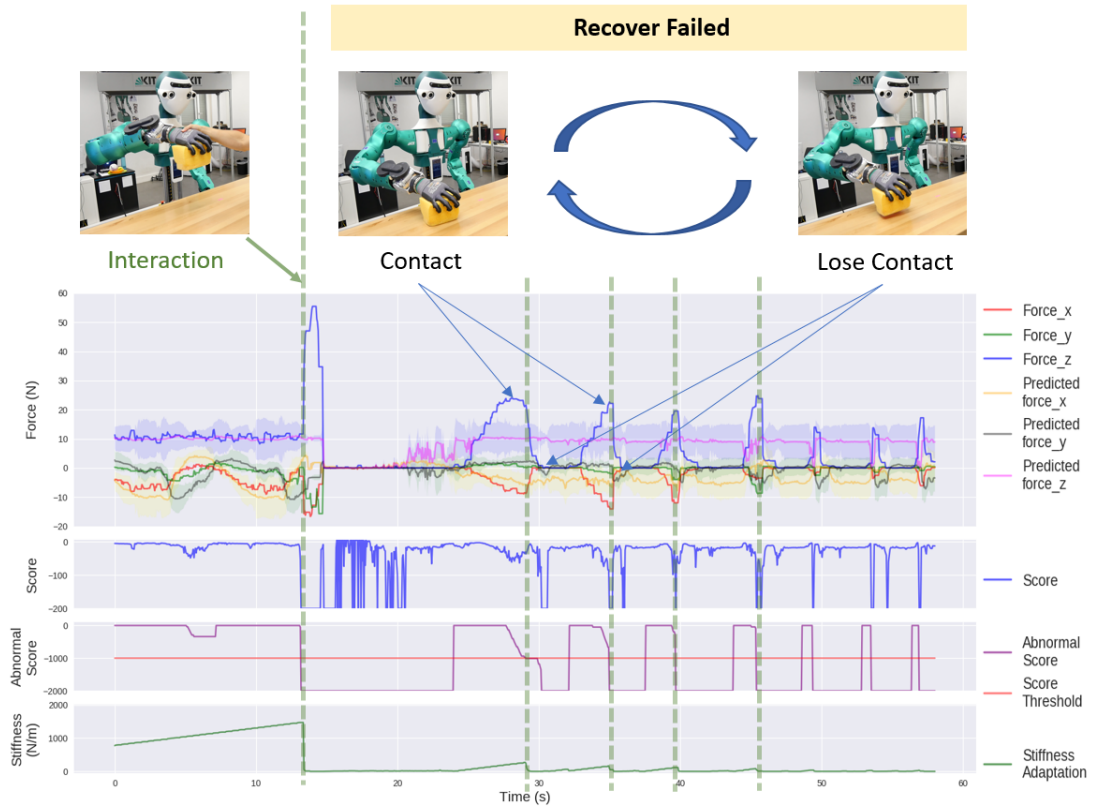

Figure 5.13: **Top Left:** Human interrupts the robot wiping task. **Top Right:** Recovery fails because one predictive model does not consider the force profile during the recovery. **Bottom:** The diagrams are shown for the force profile, scores and abnormal scores, and the adjusted stiffness separately.

3. If the robot cannot execute the task as expected, we use current data to train another predictive model. All predictive models work as a mixture of experts. According to the sum rule, the probabilities of the current force profile calculated based on all the predictive models are added together, and its logarithm is the corresponding score.

By iterating between the second and the third steps, we build a mixture of experts, which takes all experienced situations into consideration.

5.2.3 Learning Compliant Wiping Task

For the wiping task as an example, the first predictive model is learned after executing several periods of the wiping motion with a high stiff controller. In the second step, the robot follows the control diagram shown in Fig. 5.11 to execute the wiping motion. Since the first predictive model already learns the

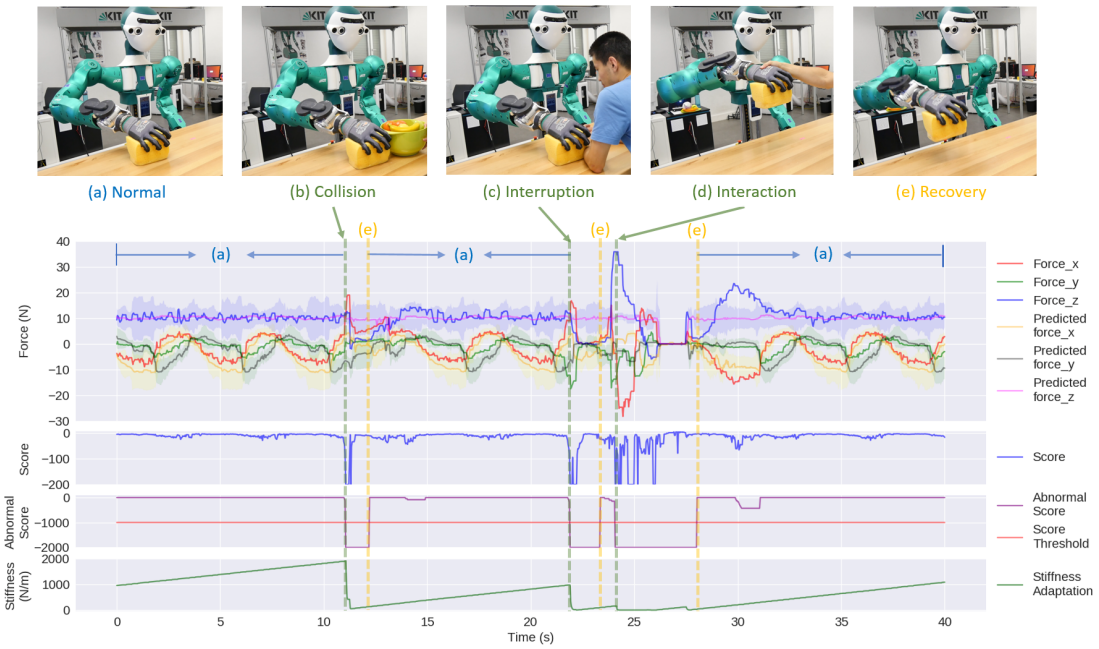

Figure 5.14: Two force predictive models are trained for a successful compliant wiping task.

friction between the table and the wiping tool, the robot can predict the force profile very well if the friction is the only external force source. Hence, the robot executes the motion with high stiffness to guarantee high tracking accuracy. Once additional external force is detected, the probability of the force profile concerning the predicted force distribution decreases. According to the adaptive strategy, the stiffness of all PID controllers decreases, and the robot is in a compliant mode.

In Fig. 5.13, we show how the robot behaves with the first predictive model. The curves denoted as "*Predcited force_x*", "*Predicted force_y*" and "*Predicted force_z*" are the mean of the distribution. The shadowed pattern indicates the $\pm 3\sigma$ region of the distribution. At the beginning of the task, the observed force profile is in the acceptable region. Hence, the controller uses a high stiffness for good tracking accuracy. When encountering the human interruption, the observed force profile is different from the predicted one, the stiffness decreases quickly (bottom diagram in Fig. 5.13) to zero, and the robot is very compliant with only gravity compensation. After perturbation, the robot, however, cannot recover from the compliant mode, and the force profile is still abnormal because the velocity and force profiles during the recovery are not in the training dataset of the first predictive model.

To solve this problem, we can learn a second predictive model according to the third step in the previous process. The data for training the second model is collected by observing the velocity and the force profiles during the failed recovery. In Fig. 5.13, this corresponds to the force curves after the human interruption. As shown in Fig. 5.14, with the combined output distributions given by both models, the robot quickly recovers from failure because the force profile is normal and a high stiff controller continues tracking the VMP generated motion trajectory.

5.3 Conclusion

In this chapter, *Coordinate Change Movement Primitive* (CC-MP) is introduced. Its basic idea is to create a leader-follower framework and learn the follower's MP in the leader's coordinate frame. CC-MP has been evaluated in different tasks such as wiping tasks, bimanual manipulation tasks.

In addition, a force predictive model based adaptive controller is introduced to realize both compliant control and accurate tracking for the VMP generated trajectory. The basic idea is to learn several force predictive models and compare the force profile during task execution with the predicted one. The adaptive strategy decreases the stiffness of the PID controllers if both profiles are different from each other. Otherwise, a high stiff controller guarantees a high tracking accuracy.

6 Conclusion and Future Works

In this work, several new approaches are proposed to provide solutions to movement primitive representation, generalization and adaptation. These methods endow a humanoid robot with adaptable movement primitives and can generalize the learned movement primitives to different task parameters and adapt the generated motion to different task requirements.

6.1 Scientific Contribution

The scientific contributions can be summarized as follows,

6.1.1 Movement Primitive Representation

In Chapter. 3, a new movement primitive representation called *via-points movement primitive* (VMP) is developed in Zhou et al. (2019). VMPs inherit the advantages of both *dynamic movement primitive* (DMP) and *probabilistic movement primitive* (ProMP), two popular approaches to represent motions in literature. VMPs allow integration of via-points into the original learned trajectory distribution to meet task requirements. Compared to DMPs, VMPs are probabilistic formulation and allow integration of intermediate via-points into the demonstrated trajectory. Compared to ProMPs, VMPs can extropolate to the via-points out of the range of demonstrations. Several robot applications show that VMPs outperform DMPs and ProMPs in the case of changing task constraints.

6.1.2 Movement Primitive Generalization

In Chapter. 4, two approaches are introduced to handle multiple modes and models that exist in the human demonstrations for the same and different task parameters, which current state of the art methods cannot deal with. The first

approach is to use a mixture of experts. In order to avoid the local minima that leads to the model collapse when using the *Expectation Maximization* (EM) algorithm, a *Leave-One-Out Expectation Maximization* (LOO-EM) algorithm is proposed in Zhou and Asfour (2017). However, this method is time-consuming for a relatively large training dataset. The second approach uses a *mixture density network* (MDN) to learn the mapping from the task parameter to the distribution of the movement primitive parameters. In order to avoid mode and model collapses during training MDN on a limited number of demonstrations, the entropy cost is introduced to achieve a more balanced association of demonstrations to GMM mixture components. Since it is often easier to collect failed examples by using an underfitted MDN model instead of additional human demonstrations, a failure cost is used to reduce the occurrence of failures in future executions. Compared to previous methods, the proposed method shows better performance in different tasks. This work has been published in Zhou et al. (2020).

6.1.3 Movement Primitive Adaptation and Control

In Chapter. 5, two different approaches are developed for trajectory adaptation and force adaptation separately. The first approach is to create a leader-follower framework for the task. The follower's movement primitive is learned in the leader's coordinate frame. During the execution, the leader's coordinate changes, and the follower adapts its motion to the leader's movement. Hence, the method is called *Coordinate Change Movement Primitive* (CC-MP). By adding a coupling term to the movement primitive formulation, the trajectory adaptation is used to also ensure a desired force of the interaction. The works for CC-MP and its force adaptation have been published in Zhou et al. (2016a) and Zhou et al. (2016b). With CC-MP, a vision-based robot wiping system is created in Dometios et al. (2018). The resulting system adapts the learned wiping pattern to a moving and curved wiping surface.

The second approach is introduced to realize a compliant robot control during contact-rich manipulation tasks. A force predictive model is learned when the robot executes the task with a high stiffness controller. Based on the predictive model, an adaptive controller is created to adapt the stiffness. Once an abnormal force profile is detected, the adaptive controller decreases the stiffness to allow human-robot interactions or avoid damages of the robot or the environment. The approach is evaluated in the robot wiping task.

6.2 Future Works

6.2.1 Movement Primitive Representation

Local Via-points Planner

The *via-points movement primitive* (VMP) introduced in Chapter. 3 adapts to arbitrary via-points. For many applications such as obstacle avoidance, VMP provides an efficient solution without learning new movement primitives from new demonstrations. However, this solution is based on where the via-points are placed. In the future, a local via-points planner can be used to determine where the via-points should be placed for a given task.

For different tasks, different local via-points planners should be developed. For example, once an obstacle is found, the local planner should place several via-points along the timeline to adjust the trajectory to avoid the obstacle. It is not a trivial task to design such a planner. As an example shown in Fig. 3.11, it is easy to decide where to place the first via-point, which has an offset away from the obstacle. However, this via-point makes grasping of the target object difficult because of the robot's kinematic constraint. Hence, a new via-point is required to improve the robot manipulability. As mentioned before, for the task requirements which cannot be represented in a fixed dimensional space, an implicit or explicit rule-based method is necessary. A local via-points planner is designed based on a set of rules for a specific task.

Active Learning

The advantage of VMP over ProMP is that it extrapolates to the via-points outside of the demonstration region. The extrapolation capability is due to the VMP structure, which is similar to DMP. The elementary trajectory of VMP can be regarded as the local frame where the shape modulation is defined. Almost all the methods that solve the extrapolation in literature are based on local frames.

However, even though VMP improves the extrapolation of the via-points adaptation, it might generate motions that are different from the demonstrations when many via-points are required for some task constraints. In this case, it does not make any sense to use one single VMP to accomplish the task. An intelligent robot decides when to ask for additional demonstrations to accomplish the task efficiently.

6.2.2 Movement Primitive Generalization

Generating a Motion from a Distribution

In Section. 4.3, a *mixture density network* (MDN) is learned to map task parameters to a distribution of the MP parameters. As shown in the hit ball example (see Section. 4.4.4), drawing multiple samples from the output distribution improves the performance of the MP generalization.

However, multiple samples require multiple robot trials, which are costly, especially when the failed trials might result in the damage of the robot or its surroundings. In order to solve this problem, in future work, a discriminator can be traine to learn the success probability of task execution based on task parameters and their corresponding MP parameters. Instead of real robot trials, the discriminator can find the one from multiple samples with the best chance of successful task execution. Designing and training such a discriminator is, however, not a trivial task for a complex task such as the hit ball task described in Section. 4.4.4. A fully connected network as the discriminator was investigated for the hit ball task, but it did not improve the performance, compared to selecting the most probable MP parameter.

Reinforcement Learning and Supervised Learning

In Kober and Peters (2014), *reinforcement learning* (RL) algorithms are used to find one MP parameter that accomplishes the task. Unlike learning from demonstrations or *supervised learning* (SL), the RL agents explore the MP space containing all possible MP parameters to increase the reward during the task execution. The basic idea is to calculate the gradient of the value function, namely accumulated rewards along with the task execution, concerning the MP parameter. Gradient ascent methods are used to find its local maximum. The difficulty is that the value function is sometimes not directly related to the MP parameters. Thus, sampling methods are necessary to approximate the gradient.

Since the RL result maximizes the value function that directly indicates how the task is executed, it is more task-oriented than the result given by learning from demonstrations (supervised learning). On the other hand, for learning from demonstrations, the model is trained based on the similarity between the generated and the demonstrated motions. In theory, it does not guarantee the successful task execution, but only succeeds if the assumption is correct that *the*

tasks with similar task parameters can be accomplished with similar motions. However, on the other side, the RL agent does not necessarily generate human-like (more intuitive) motions unless it is explicitly required and this requirement is formulated in the reward function.

Moreover, RL is more complicated to train than SL and requires more samples. Instead of using RL from scratch, RL is used to improve the result of the SL. In the future, a complete MP generalization system consists of RL and SL. RL is used on the SL results to provide more task-oriented MP parameters. Moreover, the RL result can be further used to improve SL to reduce the necessary samples required by the RL agent.

6.2.3 Movement Primitive Adaptation and Control

Deciding the Leader or the Follower

In Section. 5.1, CC-MP is introduced for the trajectory adaptation. In the described applications, the leader and the follower are manually determined. In many applications, the leader and the follower also exchanges their roles. In future work, the problem of how to determine the leader and the follower should be addressed.

Appendix

A Proof: Dot Product Kernel for Fitting the Polynomial

Lemma A.1. *Consider a 5-th order polynomial with coefficients $\boldsymbol{q}$ and a linear regression model with parameters $\boldsymbol{w}$ to fit the polynomial. The purpose is to learn a mapping from $\boldsymbol{q}$ to $\boldsymbol{w}$. If the Gaussian process regression with dot product kernels is used to model this function, the function value can almost perfectly reproduce the polynomial.*

Proof. In order to perfectly reproduce the polynomial, the result mapping $\boldsymbol{\omega}(\cdot)$ should satisfy that

$$\boldsymbol{\Psi}\boldsymbol{\omega}(\boldsymbol{q}) = X^T\boldsymbol{q}, \tag{A.1}$$

where

$$\boldsymbol{\Psi} = \begin{pmatrix} \boldsymbol{\psi}(x_0)^T \\ \boldsymbol{\psi}(x_1)^T \\ \vdots \\ \boldsymbol{\psi}(x_T)^T \end{pmatrix}, \; and \; X^T = \begin{pmatrix} x_0^5 & x_0^4 & \dots & x_0 & 1 \\ x_1^5 & x_1^4 & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_T^5 & x_T^4 & \dots & x_T & 1 \end{pmatrix}. \tag{A.2}$$

Consider the pseudo-inverse of the kernel matrix $\boldsymbol{\Psi}$, we obtain that

$$\boldsymbol{w} = \boldsymbol{\omega}(\boldsymbol{q}) = (\boldsymbol{\Psi}^T\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^T X^T\boldsymbol{q} = \Gamma^T\boldsymbol{q}. \tag{A.3}$$

For n different coefficients and their corresponding polynomials, we have

$$W_n = Q_n\Gamma, \tag{A.4}$$

where

$$W_n = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix}, \quad Q_n = \begin{pmatrix} \boldsymbol{q}_1^T \\ \boldsymbol{q}_2^T \\ \vdots \\ \boldsymbol{q}_n^T \end{pmatrix}. \tag{A.5}$$

Consider the pseudo-inverse of the matrix Q_n, we replace the matrix Γ and get the result $\boldsymbol{w}^*$ for new coefficients $\boldsymbol{q}^*$ such that

$$\boldsymbol{w}_*^T = \boldsymbol{q}_*^T (Q_n^T Q_n)^{-1} Q_n^T W_n. \tag{A.6}$$

To prove the statement, we need to prove that GPR with DPK gives the same result.

For DPK, we construct the kernel matrix:

$$K(Q_n, Q_n) = \begin{pmatrix} \boldsymbol{q}_1^T \boldsymbol{q}_1 & \boldsymbol{q}_1^T \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_1^T \boldsymbol{q}_n \\ \boldsymbol{q}_2^T \boldsymbol{q}_1 & \boldsymbol{q}_2^T \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_2^T \boldsymbol{q}_n \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{q}_n^T \boldsymbol{q}_1 & \boldsymbol{q}_n^T \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_n^T \boldsymbol{q}_n \end{pmatrix} = Q_n Q_n^T. \tag{A.7}$$

Then we have

$$Q_n^T K(Q_n, Q_n) = Q_n^T (Q_n Q_n^T). \tag{A.8}$$

By reformulating this equation with the associative rule, we get

$$(Q_n^T Q_n)^{-1} Q_n^T = Q_n^T K(Q_n, Q_n)^{-1}. \tag{A.9}$$

The result of GPR is that

$$\boldsymbol{w}_*^T = K(\boldsymbol{q}_*, Q_n) K(Q_n, Q_n)^{-1} W_n, \tag{A.10}$$

where

$$K(\boldsymbol{q}_*, Q_n) = \begin{pmatrix} \boldsymbol{q}_*^T \boldsymbol{q}_1 & \boldsymbol{q}_*^T \boldsymbol{q}_2 & \ldots & \boldsymbol{q}_*^T \boldsymbol{q}_n \end{pmatrix} = \boldsymbol{q}_*^T Q_n^T. \tag{A.11}$$

We multiply both sides of Eq. A.9 with $\boldsymbol{q}_*^T$ and W_n:

$$\boldsymbol{q}_*^T (Q_n^T Q_n)^{-1} Q_n^T W_n = \boldsymbol{q}_*^T Q_n^T K(Q_n, Q_n)^{-1} W_n = K(\boldsymbol{q}_*, Q_n) K(Q_n, Q_n)^{-1} W_n. \tag{A.12}$$

GPR result coincides with Eq. A.6.

□

List of Figures

Bibliography

Asfour, T., Kaul, L., Wächter, M., Ottenhaus, S., Weiner, P., Rader, S., Grimm, R., Zhou, Y., Grotz, M., Paus, F., Shingarey, D., and Haubert, H. (2018). ARMAR-6: A collaborative humanoid robot for industrial environments. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 447–454. *Cited on pages 73 and 99.*

Asfour, T., Regenstein, K., Azad, P., Schröder, J., Vahrenkamp, N., and Dillmann, R. (2006). ARMAR-III: An integrated humanoid platform for sensory-motor control. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 169–175. *Cited on page 112.*

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning. *Artif. Intell. Rev.*, 11(1-5):11–73. *Cited on pages 16 and 28.*

Ben Amor, H., Neumann, G., Kamthe, S., Kroemer, O., and Peters, J. (2014). Interaction primitives for human-robot cooperation tasks. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2831–2837. *Cited on page 40.*

Bennequin, D., Fuchs, R., Berthoz, A., and Flash, T. (2009). Movement timing and invariance arise from several geometries. In *PLoS Computational Biology*. *Cited on page 11.*

Bishop, C. M. (1994). Mixture density networks. Technical report. *Cited on pages 85 and 87.*

Bizzi, E., Accornero, N., Chapple, W., and Hogan, N. (1984). Posture control and trajectory formation during arm movement. *Journal of Neuroscience*, 4(11):2738–2744. *Cited on pages 25, 47, and 52.*

Bizzi, E., Mussa-Ivaldi, F., and Giszter, S. (1991). Computations underlying the execution of movement: a biological perspective. *Science*, 253(5017):287–291. *Cited on pages 9, 10, 18, and 48.*

Borràs, J., Heudorfer, R., Rader, S., Kaiser, P., and Asfour, T. (2018). The KIT swiss knife gripper for disassembly tasks: A multi-functional gripper for bimanual manipulation with a single arm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4590–4597. *Cited on page 73.*

Calinon, S. (2016). A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29. *Cited on pages 35, 36, and 97.*

Calinon, S., Alizadeh, T., and Caldwell, D. G. (2013). On improving the extrapolation capability of task-parameterized movement models. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 610–616. *Cited on pages 14, 34, 36, 49, 97, and 106.*

Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298. *Cited on page 13.*

Da Silva, B. C., Konidaris, G., and Barto, A. G. (2012). Learning parameterized skills. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1443–1450, USA. Omnipress. *Cited on pages 31, 82, and 86.*

Deniša, M., Gams, A., Ude, A., and Petrič, T. (2016). Learning compliant movement primitives through demonstration and statistical generalization. *IEEE/ASME Transactions on Mechatronics*, 21(5):2581–2594. *Cited on page 47.*

Dometios, A. C., Zhou, Y., Papageorgiou, X. S., Tzafestas, C. S., and Asfour, T. (2018). Vision-based online adaptation of motion primitives to dynamic surfaces: Application to an interactive robotic wiping task. *IEEE Robotics and Automation Letters (RA-L)*, 3(3):1410–1417. *Cited on pages 106, 110, 112, and 124.*

Dragan, A. D., Muelling, K., Bagnell, J., and Srinivasa, S. S. (2015). Movement primitives via optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2339–2346. IEEE. *Cited on pages 18, 53, 56, and 62.*

Ewerton, M., Neumann, G., Lioutikov, R., Amor, H. B., Peters, J., and Maeda, G. (2015). Learning multiple collaborative tasks with a mixture of interaction primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1535–1542. *Cited on pages 22, 38, 49, and 95.*

Flash, T. and Hogan, N. (1985). The coordination of arm movements: an experimentally confirmed mathematical model. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 5 7:1688–703. *Cited on page 64.*

Forte, D., Gams, A., Morimoto, J., and Ude, A. (2012). On-line motion synthesis and adaptation using a trajectory database. *Robotics and Autonomous Systems*, 60:1327–1339. *Cited on pages 30, 32, and 86.*

Frühwirth-Schnatter, S. (2006). Finite mixture and markov switching models. *Cited on page 92.*

Gams, A., Do, M., Ude, A., Asfour, T., and Dillmann, R. (2010). On-line periodic movement and force-profile learning for adaptation to new surfaces. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 560–565. *Cited on pages 17, 113, and 114.*

Gams, A., Nemec, B., Ijspeert, A. J., and Ude, A. (2014). Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics*, 30(4):816–830. *Cited on page 44.*

Gams, A., Petric, T., Do, M., Nemec, B., Morimoto, J., Asfour, T., and Ude, A. (2016). Adaptation and coaching of periodic motion primitives through physical and visual interaction. *Robotics and Autonomous Systems*, 75, Part B:340–351. *Cited on pages 17 and 43.*

Gomi, H. and Kawato, M. (1997). Human arm stiffness and equilibrium-point trajectory during multi-joint movement. *Biological Cybernetics*, 76:163–171. *Cited on page 53.*

Gribovskaya, E., Khansari-Zadeh, S., and Billard, A. (2011). Learning nonlinear multivariate dynamics of motion in robotic manipulators. *The International Journal of Robotics Research*, 30(1):80–117. *Cited on pages 23, 48, 51, and 53.*

Hjorth, L. U. and Nabney, I. T. (1999). Regularisation of mixture density networks. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 521–526 vol.2. *Cited on page 90.*

Hoffmann, H., Pastor, P., Park, D., and Schaal, S. (2009). Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance. In *2009 IEEE International Conference on Robotics and Automation*, pages 2587–2592. *Cited on pages 15, 18, and 40.*

Hogans, N. (1984). An organizing principle for a class of voluntary movements. *Journal of neuroscience*, 4:2745–54. *Cited on page 64.*

Huber, L., Billard, A., and Slotine, J. (2019). Avoidance of convex and concave obstacles with convergence ensured through contraction. *IEEE Robotics and Automation Letters*, 4(2):1462–1469. *Cited on page 41.*

Ijspeert, A., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems 15*, pages 1547–1554. *Cited on pages 15, 16, and 17.*

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Peter, P., and Schaal, S. (2013). Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Comput.*, 25(2):328–373. *Cited on pages 18, 22, 48, 109, and 110.*

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive mixture of local expert. *Neural Computation*, 3:78–88. *Cited on page 82.*

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957. *Cited on page 23.*

Khatib, O. (1990). *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, pages 396–404. Springer New York, New York, NY. *Cited on page 75.*

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*. *Cited on page 89.*

Kober, J. and Peters, J. (2014). *Reinforcement Learning in Robotics: A Survey*, pages 9–67. Springer International Publishing, Cham. *Cited on page 126.*

Koert, D., Pajarinen, J., Schotschneider, A., Trick, S., Rothkopf, C., and Peters, J. (2019). Learning intention aware online adaptation of movement primitives. *IEEE Robotics and Automation Letters*, 4(4):3719–3726. *Cited on page 42.*

Kramberger, A., Shahriari, E., Gams, A., Nemec, B., Ude, A., and Haddadin, S. (2018). Passivity based iterative learning of admittance-coupled dynamic movement primitives for interaction with changing environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6023–6028. *Cited on pages 45 and 46.*

Kronander, K. and Billard, A. (2016). Passive interaction control with dynamical systems. *IEEE Robotics and Automation Letters*, 1(1):106–113. *Cited on pages 45 and 46.*

Lacquaniti, F., Terzuolo, C., and Viviani, P. (1983). The law relating kinematic and figural aspects of drawing movments. *Acta psychologica*, 54:115–30. *Cited on page 11.*

Lavalle, S. M., Kuffner, J. J., and Jr. (2000). Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308. *Cited on page 75.*

Maeda, G., Ewerton, M., Lioutikov, R., Ben Amor, H., Peters, J., and Neumann, G. (2014). Learning interaction for collaborative tasks with probabilistic movement primitives. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 527–534. *Cited on page 36.*

Makansi, O., Ilg, E., Çiçek, Ö., and Brox, T. (2019). Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). Cited on page 91.*

Mandery, C., Terlemez, O., Do, M., Vahrenkamp, N., and Asfour, T. (2015). The KIT whole-body human motion database. In *International Conference on Advanced Robotics (ICAR)*, pages 329–336. *Cited on page 109.*

Maoz, U., Berthoz, A., and Flash, T. (2009). Complex unconstrained three-dimensional hand movement and constant equi-affine speed. *Journal of neurophysiology*, 101:1002–15. *Cited on page 11.*

Matsubara, T., Hyon, S.-H., and Morimoto, J. (2010). Learning stylistic dynamic movement primitives from multiple demonstrations. volume 24, pages 1277–1283. *Cited on pages 31 and 56.*

Mclachlan, G. and Basford, K. (1988). *Mixture Models: Inference and Applications to Clustering*, volume 38. *Cited on page 88.*

Meier, F. and Schaal, S. (2016). A probabilistic representation for dynamic movement primitives. *CoRR. Cited on page 56.*

Meirovitch, Y., Bennequin, D., and Flash, T. (2016). Geometrical invariance and smoothness maximization for task-space movement generation. *IEEE Transactions on Robotics*, 32(4):837–853. *Cited on page 12.*

Mussa-Ivaldi, F. and Bizzi, E. (2001). Motor learning through the combination of primitives. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 355:1755–69. *Cited on pages 9 and 10.*

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2009). Local gaussian process regression for real time online model learning and control. In *Advances in neural information processing systems 21*, pages 1193–1200, Red Hook, NY, USA. Max-Planck-Gesellschaft, Curran. *Cited on page 31.*

Pahic, R., Gams, A., Ude, A., and Morimoto, J. (2018). Deep encoder-decoder networks for mapping raw images to dynamic movement primitives. pages 1–6. *Cited on page 31.*

Paraschos, A., Daniel, C., Peters, J., and Neumann, G. (2018). Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):529–551. *Cited on pages 19, 20, and 21.*

Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. (2013). Probabilistic movement primitives. In *Advances in Neural Information Processing Systems 26*, pages 2616–2624. Curran Associates, Inc. *Cited on pages 19, 21, 48, and 53.*

Park, D., Hoffmann, H., Pastor, P., and Schaal, S. (2008). Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pages 91–98. *Cited on pages 41 and 44.*

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768. *Cited on page 40.*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. *Cited on page 33.*

Pervez, A. and Lee, D. (2018). Learning task-parameterized dynamic movement primitives using mixture of gmms. *Intelligent Service Robotics*, 11(1):61–78. *Cited on page 31.*

Pham, Q. and Nakamura, Y. (2015). A new trajectory deformation algorithm based on affine transformations. *IEEE Transactions on Robotics*, 31(4):1054–1063. *Cited on pages 10, 11, and 12.*

Plappert, M., Mandery, C., and Asfour, T. (2018). Learning a bidirectional mapping between human whole-body motion and natural language using deep recurrent neural networks. *Robotics and Autonomous Systems*, 109:13–26. *Cited on page 119.*

Pollick, F. and Sapiro, G. (1997). Constant affine velocity predicts the 13 power law of planar motion perception and generation. *Vision research*, 37:347–53. *Cited on page 11.*

Prada, M., Remazeilles, A., Koene, A., and Endo, S. (2013). Dynamic movement primitives for human-robot interaction: Comparison with human behavioral observation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1168–1175. *Cited on page 40.*

Rueckert, E. and D'Avella, A. (2013). Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems. *Frontiers in computational neuroscience*, 7:138. *Cited on page 71.*

Schaal, S. (2003). Dynamic movement primitives - a framework for motor control in humans and humanoid robots. In *The International Symposium on Adaptive Motion of Animals and Machines*. *Cited on pages 15, 16, 17, and 48.*

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681. *Cited on page 118.*

Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. (2013). Learning compact parameterized skills with a single regression. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 417–422. *Cited on pages 26, 31, 32, 34, and 48.*

Stulp, F., Theodorou, E., Kalakrishnan, M., Pastor, P., Righetti, L., and Schaal, S. (2011). Learning motion primitive goals for robust manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 325–331. *Cited on page 71.*

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE. *Cited on page 99.*

Ude, A., Gams, A., Asfour, T., and Morimoto, J. (2010). Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, 26(5):800–815. *Cited on pages 27 and 86.*

Villani, L. and De Schutter, J. (2008). *Force Control*, pages 161–185. Springer Berlin Heidelberg, Berlin, Heidelberg. *Cited on page 43.*

Zhou, Y. and Asfour, T. (2017). Task-oriented generalization of dynamic movement primitive. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3202–3209. *Cited on pages 28, 80, 95, and 124.*

Zhou, Y., Do, M., and Asfour, T. (2016a). Coordinate change dynamic movement primitives - a leader-follower approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5481–5488. *Cited on pages 106 and 124.*

Zhou, Y., Do, M., and Asfour, T. (2016b). Learning and force adaptation for interactive actions. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 1129–1134. *Cited on pages 106, 113, 114, and 124.*

Zhou, Y., Gao, J., and Asfour, T. (2019). Learning via-point movement primitives with inter- and extrapolation capabilities. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. *Cited on pages 51 and 123.*

Zhou, Y., Gao, J., and Asfour, T. (2020). Movement primitive learning and generalization using mixture density networks. *IEEE Robotics & Automation Magazine (to appear)*, pages 0–0. *Cited on pages 80 and 124.*

Zlatintsi, A., Dometios, A., Kardaris, N., Rodomagoulakis, P. K. I., Papageorgiou, X., Maragos, P., Tzafestas, C., Varholomeos, P., Hauer, K., Werner, C., Annicchiarico, R., Lombardi, M. G., Adriano, F., Asfour, T., Sabatini, A. M., Laschi, C., Cianchetti, M., Guler, R. A., Kokkinos, I., Klein, B., and Lopez, R. (2020). I-support: a robotic platform of an assistive bathing robot for the elderly population. *Robotics and Autonomous Systems*, pages 0–0. *Cited on pages 105 and 108.*