UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

A Structured Prediction Approach to Robot Imitation Learning

by

Anqing Duan

Thesis submitted for the degree of *Doctor of Philosophy* (33° cycle)

Feb 2021

Daniele Pucci	Scientific Supervisor
Giorgio Metta	Supervisor
Giorgio Cannata	Head of the PhD program

Thesis Jury:

Aude Billard, *EPFL*	External examiner
Elmar Rückert, *Universität zu Lübeck*	External examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Anqing Duan
June 2021

Acknowledgements

During my past PhD journey, I have received incredible support and help from various sources. Therefore, I would like to take this chance to express my deep heart appreciation towards:

- first of all, my supervisor Daniele Pucci, without whose generous admission, my academic career could have just been impossible. His supervision, support, encouragement, and tolerance made the tortuous PhD journey much smoother for me.
- Lorenz Rosasco, from whom I learned quite a lot. His lecture notes on machine learning always remain one of my favorite machine learning tutorials.
- Aude Billard, for allowing me to carry out my secondment at her LASA lab at EPFL and her agreeing to be my thesis committee member. Her insights on robot learning will become an integral part of my future research.
- Elmar Arne Rueckert and Carlo Ciliberto for agreeing to be my other thesis committee members.
- all my collaborators and other tutors from DIC, LCSL, and LASA including Raffaello, Diego, Calandriello, Iason among many others.
- iCub secretaries including Marta, Grazia, and Lucia for making my administration procedure a lot easier.
- my families for their constant support and unconditional love over the years.

- my girlfriend Dan, without whom my thesis shall be finished earlier.

Abstract

This thesis is primarily focused on movement primitives-based imitation learning, within the context of robot programming by demonstration. Specifically, the imitation problem is tackled from a supervised-learning perspective. Therefore, it allows us to resort to theoretical tools from structured prediction, which can handle data-sets with complex structures.

The first part of the thesis provides an overall background, in which we overview state-of-the-art imitation learning algorithms as well as discuss relevant technical tools. We formally introduce our contribution in part II. Our algorithm is not only capable of learning usual Euclidean trajectories (Chapter 7), but also trajectories lying on some manifold (Chapter 8). The capability of adapting manifold trajectories distinguishes our approach from other imitation learning algorithms. Subsequently, we provide a few extensions to augment our approach, including trajectory refinement by policy search (Chapter 10), imitation learning with constraints (Chapter 11), and probabilistic trajectory transfer (Chapter 12). We then conclude the thesis in the epilogue.

Table of contents

List of figures

List of tables

Prologue

0.1 Challenges of Building Intelligent Robots

Programming a robot for a desired skill is never a simple task. The difficulties could result from many aspects. For example, when a given task requires robots to move dynamically or manipulate dexterously, it will be challenging to explicitly specify robot behavior by manually designing a reward function. In this case, neither reward components nor their corresponding weights are straightforward to obtain. An iterative process of trying out different combinations of reward components and tuning parameters is often inevitable, which can be time consuming and mundane. Besides, as nowadays robots are expected to be versatile to be able to operate in cluttered environments and deal with a variety of complex tasks, they are becoming more and more general purpose and consequently possess high degree of freedom. As a result, the innate redundancy of the flexible robots will make it prohibitively challenging to search for the optimal policy due to the so called 'curse of dimensionality' dilemma (Bertsekas et al., 2000). In view of the aforementioned challenges, therefore, exploring novel methodologies that offer a convenient and efficient manner to augment robots' capabilities is always appealing to roboticists and thus remains research trend.

Bio-inspired techniques have always been playing an important role in building autonomous intelligent machines (Hwangbo et al., 2019; Peng et al., 2020). In terms of endowing robots with new skills, a good perspective is to turn to nature, where one notable strategy is learning by imitation (Nehaniv et al., 2002). In the field of robotics, robot imitation learning, also known as learning

by demonstration or learning from demonstration, offers a general framework to transfer experts' skill to robots (Osa et al., 2018). As the terminology implies, a robot learns a new skill from a human teacher by mimicking the teacher's demonstrated behavior. Imitation learning have provided multiple merits in the filed of robotics. Thanks to the generality of the imitation learning framework, robots are able to acquire new skills in a fast and efficient manner. Compared with traditional robot programming methods, the required time investment and expertise knowledge are significantly reduced for the development of novel robot tasks. Besides, it has been revealed that robots with human-like behavior is more acceptable to their human co-workers than those without (You and Robert Jr, 2018). In order to increase human co-workers' willingness to work alongside a robot and boost productivity, it is thus important to make robots behave in a human-like fashion.

As a result, many robotic sub-fields have witnessed benefits by making use of imitation learning (Osa et al., 2018). For example, in the field of humanoid robotics, it is often desirable to make motor learning efficient for complex high dimensional humanoid robots (Schaal, 1999). In addition, imitation learning can also help fly helicopters through a wide range of maneuvers from expert demonstrations (Abbeel et al., 2010). Moreover, several recent developments in reinforcement learning rely on applying imitation learning methods. By bootstrapping from the initial policy drawn from imitation learning, policies learned via reinforcement learning can outperform a suboptimal expert and converge faster than running policy gradient from scratch (Cheng et al., 2018). Motion imitation can be also used to design a reward function for reinforcement learning in order to perform a broad range of complex motions (Peng et al., 2018; Xie et al., 2020). The amount of papers on imitation learning based methods for robotics research has been increasing steadily over the years. Figure 1 from Ravichandar et al. (2020) shows statistical evidence of the popularity of robot imitation learning over the years.

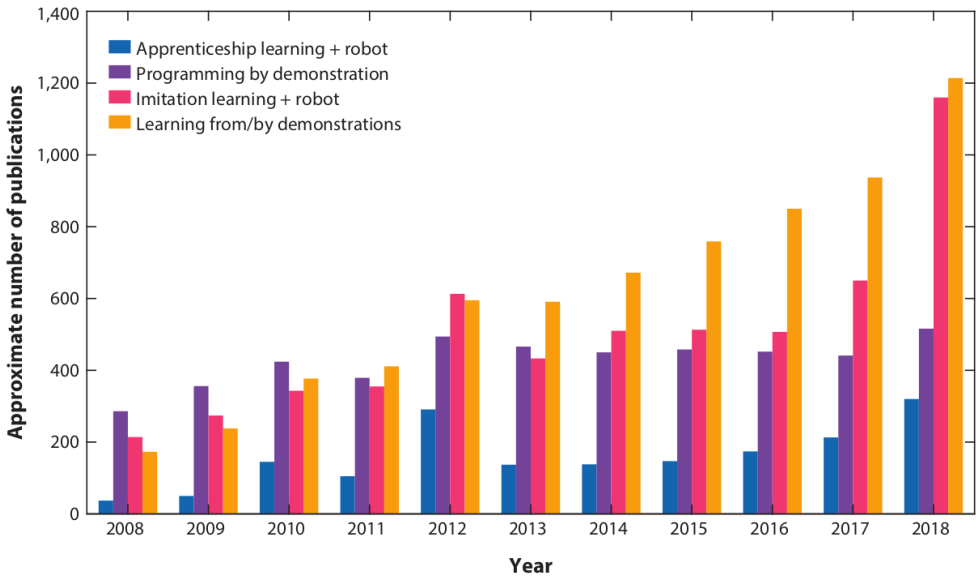


Figure 1 Increasing research attention on the topic of robotics and imitation learning as evidenced by steadily increasing amount of the publications. (Source from Ravichandar et al. (2020))

0.2 Thesis Organization

This thesis is organized into two parts. In Part I, the related background will be reviewed. As this thesis aims to address robot imitation learning issues, different imitation learning algorithms will be given, in particular, we emphasize the usage of imitation in the field of robotics. Later, we will briefly review structured prediction, a specialized supervised learning technique that deals with output with complex structure. Structured prediction is the backbone of our derivation of the proposed imitation learning. The choice of structured prediction is made upon observation that supervised learning has been playing a central role in designing an imitation learning algorithm. Among various approaches to structured prediction, we will spend more space on a specific one.

Once we have sufficient background knowledge, we will be able to introduce our algorithm in Part II, which is the main contribution of this thesis. We will see that by tackling imitation using a tool from structured prediction, we are able to endow robots with the ability to learn output with a variety of

structures. Specifically, thanks to the generality of structured prediction, our algorithmic framework can provide robots the ability to imitate trajectories not only belonging to Euclidean space, but also lying on a manifold. At the core of our approach, in order to guarantee imitation fidelity, we leverage tools from information theory, namely the f-divergence, to construct the minimization objective by measuring the information loss between the demonstrated and reproduced trajectories. Wherein, different types off-divergence will recover different policies, which we coin as imitation modes. Our approach is convenient to incorporate spatial and temporal trajectory modulation, which is necessary for robots to be adaptive when facing the unknown.

To manifest the application prospect of our method, we generalize our approach with a few illustrative extensions. We will see that our method can be seamlessly applied for the scenario of constrained imitation learning, where the evolution range of the trajectory needs to be bounded. And our learned probabilistic trajectory can be also processed with probabilistic trajectory transfer as it has been done for the deterministic case. Also, our trajectory can be further refined using classical policy search. Although our approach is intrinsically a non-parametric method, which makes trajectory improvement by parameter learning not straightforward. Yet we will see that by using a simple trick, we can make our trajectory policy get improved in terms of addition requirements.

We finally close this thesis in epilogue, where we draw conclusions from benchmark experiments with several other state-of-the-art methods. We will see that our proposed method preserves key features of traditional movement primitives yet possesses the merits of learning complex structured output as well as low computational complexity.

List of Publications

- **Duan A.**, Camoriano R., Ferigo D., Huang Y., Calandriello D., Rosasco L., Pucci D. Learning to Avoid Obstacle with Minimal Intervention Control. Frontiers in Robotics and AI.

- **Duan A.**, Camoriano R., Ferigo D., Huang Y., Calandriello D., Rosasco L., Pucci D. Learning to Sequence Multiple Tasks with Competing Constraints. In Intelligent Robots and Systems (IROS), 2019 IEEE International Conference on.

- **Duan A.**, Camoriano R., Ferigo D., Calandriello D., Rosasco L., Pucci D. Constrained DMPs for Feasible Skill Learning on Humanoid Robots. In Humanoid Robots (Humanoids), 2018 IEEE International Conference on 2018 Nov IEEE.

Part I

Background

Chapter 1

State of the Art on Imitation Learning

In this section, we overview mainstream state-of-the-art algorithms for robot imitation learning. It should be noted that in literature the terminologies imitation learning and learning by demonstration are used interchangeably as the case in this thesis. Nevertheless, there could be some subtle definition differences in some contexts. Table 1.1 lists respective definition for several common demonstration-based learning approaches according to Abbeel (2012). The scope of this thesis falls into the category of learning by demonstration or learning from demonstration (Billard et al., 2008) which we will cover extensively in this section. We will also review some other imitation learning approaches especially the ones from behavior cloning as it is closely related to learning by demonstration.

To conduct learning by demonstration, usually three phases are involved. The first phase is skill transfer, where a human expert teaches a robot the desired skills. The second phase is learning phase, where robots process data collected during demonstration phase. The last phase is reproduction, where robots execute learned skills on its own in an environment which could be different from that of demonstration.

The interfaces for knowledge transfer are very versatile and dependent on the characteristics of the skill of interest. For example, when the leaner robot

Terminology	Definition
Inverse Reinforcement Learning	Recover reward function from demonstrations
Apprenticeship Learning	Use the recovered reward to find a good policy
Behavioral Cloning	Learn the expert's policy using supervised learning
Programming by demonstration	Learn the expert's trajectory with movement primitives

Table 1.1 Terminologies and their definitions for common imitation learning approaches.

is a humanoid, it is straightforward to copy the demonstrator's motor skill via whole-body geometric retargeting, a dedicated method for tele-operation of humanoid robots (Darvish et al., 2019). When the learner robot is accessible to the human demonstrator, kinesthetic teaching is a popular method, where the robot's movement is manually guided (Calinon et al., 2007). When it comes to hand skills transfer, the human user can transfer the desired skills via data gloves (Kumar et al., 2012).

In the context of robot programming by demonstration, a key research area is movement primitives (Calinon and Lee, 2019). With the help of movement primitives, the demonstrated trajectory can be expressed as a compact and adaptive movement representation. A salient merit of encoding trajectory with movement primitives is to build unit blocks that can be organized later either in parallel or in series such that more complex behaviors can be generated. This allows demonstrated behavior to be applied to different operating situations.

Being a promising research area, a multitude of algorithms for movement primitives have been developed for advancing robot learning by demonstration. One of the most notable pioneering method is Dynamic Movement Primitives (DMP), whose formulation is based on dynamical systems (Ijspeert et al., 2013). DMP has attracted considerable attention, evidenced by many corresponding derivatives, such as invariant DMP (Ginesi et al., 2019), reversible DMP (San Juan et al., 2020), sequenced DMP (Kulvicius et al., 2011), and probabilis-

tic DMP (Calinon et al., 2012) to name a few. The movement represented by DMP relies on a spring-damper system with a controller modulating the system evolution. By modulating the dynamical system with a so-called forcing term, DMP is able to imitate the demonstrated trajectory shape. The forcing term is usually learned from demonstration trajectory with a combination of radial basis functions. Since the forcing term is designed to be dependent on the phase signal instead of time directly, it will asymptotically vanish as the movement comes to an end. Therefore, stability is guaranteed since only the attractor dynamics are active. Though changing the end point position of the movement is supported, DMP is not flexible for adapting towards intermediate points during the movement. In addition, the original formulation is a non-probabilistic approach and thus cannot capture trajectory distribution.

As a probabilistic formulation of the movement primitives, Probabilistic Movement Primitives (ProMP) allows for maintaining a distribution over trajectories Paraschos et al. (2013). In ProMP, each demonstrated trajectory is assumed to be a weighted sum of several normalized radial basis functions. ProMP exhibits great flexibility, which makes it possible to adjust trajectories to new, unseen situations. Trajectory adaptation is achieved by Gaussian conditioning. Based on new mean and covariance, the weight vector can be calculated accordingly. Like DMP, ProMP also relies on a regression-based procedure with customized basis functions.

The burden of manually designing basis functions would become challenging when dealing with imitation with multi-dimensional inputs, since defining multivariate Gaussian basis functions requires center vectors and covariance matrices. To address these issues, Kernelized Movement Primitives (KMP) expresses motion trajectories with a kernelized strategy. The usage of the kernel trick makes KMP a non-parametric approach and therefore able to deal with multi-dimensional inputs. Such capability of generating trajectory associated with multi-dimensional inputs could be beneficial in some scenarios. For example, a synchronization constraint needs to be respected in human-robot collaboration where the robot is required to react to the human states (Huang et al., 2019) or a task synergy needs to be captured in bi-manual manipulation

where one end-effector is required to reactively behave according to the pose of the other one (Zeestraten et al., 2017a). In addition to the kernel method, other non-parametric methods could be also applied for learning by demonstration. For example, local Gaussian process regression (LGP) can be used to learn a trajectory by using the variations and similarities between demonstrations (Schneider and Ertel, 2010). By using simple local models, LGP is able to reduce the computational cost compared with conventional Gaussian processes and kernel methods. In addition to tackling imitation learning problems with the help of regression techniques, simple mathematical methods such as Learning from Demonstration by Averaging Trajectories (LAT) also showed practical functionality. LAT has very low algorithmic complexity as well as computation costs. Nevertheless, to generate a smooth trajectory, LAT required many data points. In general, the usage of LGP and LAT is more concentrated on reproduction phase, which could limit their applications.

As aforementioned methods somewhat require explicit input to generate motion, nonlinear autonomous dynamic system (DS) emerges as a powerful tool due to the merits of global stability and robustness to disturbances. By encoding an autonomous dynamical system using GMM, a stable estimator of dynamical systems (SEDS) can generate robot motion with guaranteed global asymptotic stability (Khansari-Zadeh and Billard, 2011). The entire attractor landscape can be encoded in the state-space of the observed data. This type of state-dependent trajectory representation can overcome the limit of temporal dependency, which in turn increases robustness to external disturbances as the changes in robot's state can be used in seamless re-planning of the task. Dynamical systems can be also extended to control forces at contact (Amanhoud et al., 2019). Learning approaches can be used to model both the motion and force profile (Khoramshahi et al., 2020) and to adapt to a given surface. While (Khoramshahi et al., 2020) can learn and control motion when moving on a complex surface, the surface shape was learned separately and known prior to learning of the motion. Further, to learn highly non-linear trajectories, LPV-DS is able to fit GMM to trajectory data in a physically consistent way with a parameterized quadratic Lyapunov function (Figueroa and Billard, 2018). It should be noted

that methods based on dynamical system are usually computationally expensive especially in high dimensional space.

Despite of the promising achievements made by the aforementioned algorithms, the perspective of their formulation is largely restricted to Euclidean settings, which could pose a challenge when learning trajectories with inherent manifold-type constraints. For some application scenarios, it could be safety-critical to avoid breaking such constraint. For example, an end-effector of a robot arm could be required to move along the surface of a cylinder without any penetration into the cylinder body (Ahmadzadeh and Chernova, 2018). Driven by practical usefulness and theoretical questions, robot imitation learning with manifold constraints is gradually receiving more attention (Calinon, 2020). Indeed trajectory planning considering manifold-type constraint remains a popular topic. For example, in order for convenient fusion of different motion policies, Ratliff et al. (2018) developed Riemannian Motion Policies (RMP), which parameterize non-Euclidean behaviors as a second order dynamic system associated with a corresponding Riemannian metric in intrinsically nonlinear task spaces. RMP can generate collision-avoidance trajectories efficiently in lieu of more complex motion planners. To plan curvature-constrained trajectories, Schulman et al. (2014) proposed TrajOpt, which is based on sequential convex optimization. TrajOpt supports trajectory optimization over manifolds such as the $SE(3)$ Lie group by iteratively optimizing over increments to the trajectory, defined in terms of the corresponding Lie algebra. Bonalli et al. (2019) also address manifold-type constraints with sequential convex programming by using geometric embeddings, which is a mapping that allows to recover manifolds as subsets of Euclidean spaces. Therefore, the problem of manifold trajectory planning is transformed into an equivalent problem that enjoys a Euclidean structure. Kingston et al. (2019) also extend a sampling-based motion planning framework with projection- and continuation-based methods as space representations to plan motion in the presence of manifold constraints. Compared with the previous manifold-constrained motion planning, our approach addresses such problem focusing on imitation learning with the guarantee that generated probabilistic trajectories satisfy the imposed manifold constraint.

In addition to imitation learning with trajectory as policy representation, another closely related area is behavior cloning, which addresses imitation in the context of Markov decision process. For example, one classic approach of behavior cloning is Dataset Aggregation (DAGGER). DAGGER reduced the original harder imitation learning problem into an easier supervised learning. The agent is able to query expert at any state, which makes the learning process interactive. By running roll-outs and concatenating demonstrated trajectories, DAGGER is able to improve the policy by having interactive feedback with the expert. Instead of aggregating data, it can be also considered to aggregate policies. Stochastic Mixing Iterative Learning (SMILe) (Daumé et al., 2009) and SEARN (Ross and Bagnell, 2010) improve learner's performance linearly by combining policies from each iteration.

Chapter 2

Probabilistic Imitation Learning

In this chapter, we review commonly used probabilistic tools in imitation learning, namely Gaussian mixture model (GMM) and Gaussian Mixture Regression (GMR). Before presenting their formula, we will first review multivariate Gaussian distributions and corresponding identities, which lie at the heart of a multitude of probabilistic imitation learning algorithms such as GMM/GMR.

2.1 Multivariate Gaussian Distribution

The Gaussian distribution is a widely used model for the distribution of continuous variables. We are most familiar with its single variable case, which can be written in the form

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}x-\mu)^2, \tag{2.1}$$

where μ is the mean and σ^2 is the variance.

Similarly to (2.1), for a $D-$dimensional vector $\mathbf{x}$, the multivariate Gaussian has the form

$$\mathcal{N}(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp(-\frac{1}{2}(x-\mu)^\top\Sigma^{-1}(x-\mu)), \tag{2.2}$$

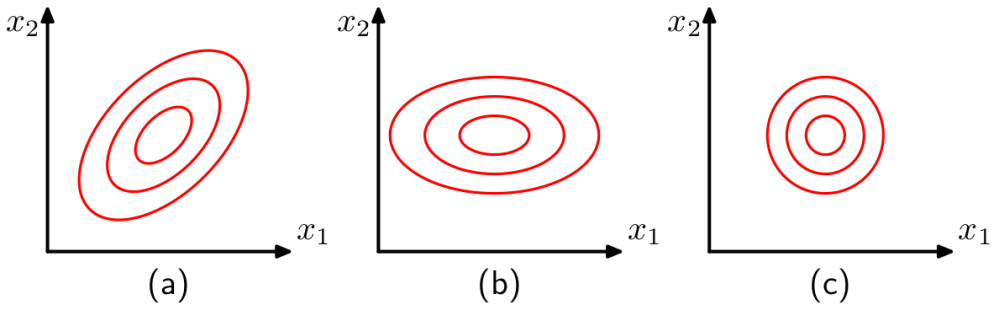


Figure 2.1 Contours of constant probability density for a two dimensional Gaussian distribution with a (a) general, (b) diagonal, and (c) isotropic covariance matrix, respectively. (Source from Bishop (2006))

where μ is a $D-$dimensional mean vector, Σ is a $D \times D$ covariance matrix, and $|\cdot|$ denotes the determinant of a matrix. When $\Sigma = \sigma^2\mathbf{I}$, it is also known as an isotropic covariance. Fig 2.1 shows different possibilities of covariance matrices for a two dimensional Gaussian distribution.

An import property of multivariate Gaussian distribution is the so called conditional Gaussian distributions, which has widely applications in Gaussian process (Rasmussen, 2003), Gaussian mixture regression, etc.

Suppose a vector $\mathbf{x}$ is a $D-$ dimensional vector satisfies Gaussian distribution $\mathcal{N}(\mathbf{x}|\mu,\Sigma)$. We can partition $\mathbf{x}$ into two disjoint subsets $\mathbf{x}_a$ and $\mathbf{x}_b$, i.e.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix}. \tag{2.3}$$

The corresponding mean vector μ can be defined by

$$\mu = \begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}. \tag{2.4}$$

And the corresponding covariance matrix Σ can be given by

$$\Sigma = \begin{bmatrix} \Sigma_{aa}\Sigma_{ab} \\ \Sigma_{ba}\Sigma_{bb} \end{bmatrix}. \tag{2.5}$$

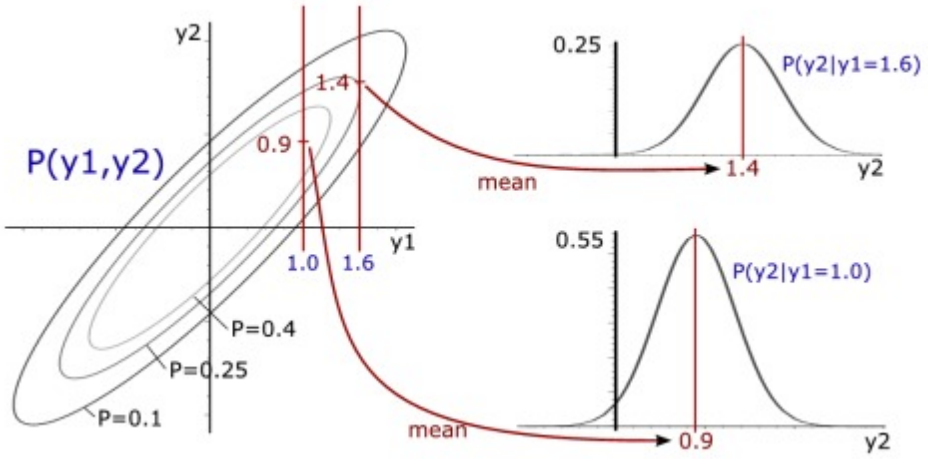


Figure 2.2 Illustration of conditional Gaussian distribution. (Source from http://www.cvlibs.net/projects/gausspro/)

Note that with the symmetry property of $\Sigma = \Sigma^\top$, we have Σ_{aa} and Σ_{bb} are symmetric and $\Sigma_{ab}^\top = \Sigma_{ba}$. The conditional distribution $\mathscr{P}(\mathbf{x}_a|\mathbf{x}_b)$ has the mean

$$\mu_{a|b} = \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \mu_b),$$

and the covariance

$$\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}.$$

The equations may not seem intuitive at a first glance. To better explain the mechanism of conditional Gaussian distribution, a pictorial illustration is shown in Fig. 2.2.

Another important identity of multivariate Gaussian distribution is Gaussian product. Consider two multivariate Gaussian distribution $\mathscr{N}(\mathbf{x}|\mu_a,\Sigma_a)$ and $\mathscr{N}(\mathbf{x}|\mu_b,\Sigma_b)$. The product of two Gaussians lead to another (un-normalized) Gaussian

$$\mathscr{N}(\mathbf{x}|\mu_a,\Sigma_a)\mathscr{N}(\mathbf{x}|\mu_b,\Sigma_b) = Z^{-1}\mathscr{N}(\mathbf{x}|\mu_c,\Sigma_c) \tag{2.6}$$

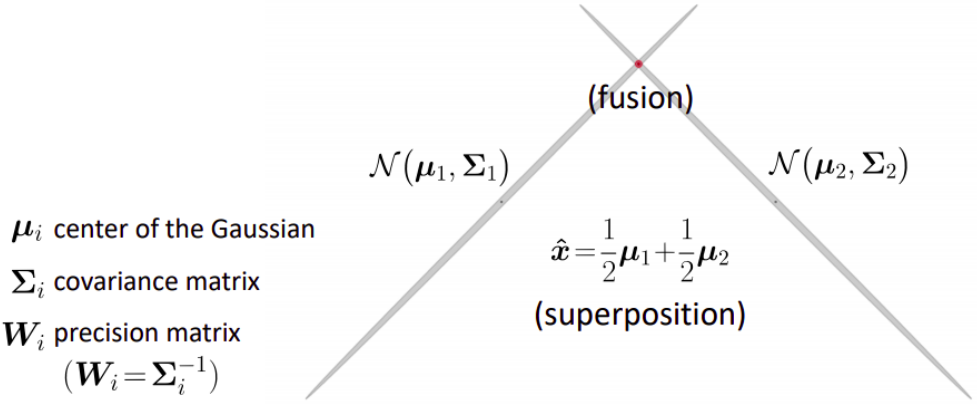


Figure 2.3 Illustration of product of Gaussians (Source from Calinon (2019)).

where

$$\begin{aligned} \mu_c &= \Sigma_c(\Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b) \\ \Sigma_c &= (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1} \end{aligned} \tag{2.7}$$

Finally, the normalizing constant is given by

$$Z^{-1} = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_a + \Sigma_b|^{1/2}} \exp(-\frac{1}{2}(\mu_a - \mu_b)^\top (\Sigma_a + \Sigma_b)^{-1}(\mu_a - \mu_b)). \tag{2.8}$$

Gaussian product usually can be used for information fusion. This property is employed for combination of movement primitives. To see a motivating example, consider the optimization problem

$$\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} \|\mu_1 - \mathbf{x}\|_{\mathbf{W}_1} + \|\mu_2 - \mathbf{x}\|_{\mathbf{W}_2} \\ &= (\mathbf{W}_1 + \mathbf{W}_2)^{-1}(\mathbf{W}_1\mu_1 + \mathbf{W}_2\mu_2) \\ &= (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2) \end{aligned} \tag{2.9}$$

where $\mathbf{W}_i = \Sigma_i$ is called the precision matrix. Fig. 2.3 shows the fusion effect of the Gaussian product.

2.2 Gaussian Mixture Model

In order to convey more information (such as variability and correlation) to a robot learner, typically a human teacher presents multiple demonstrations for a single task. Assume that the raw data $\{\{\mathbf{x}_n^m, \mathbf{y}_n^m\}_{n=1}^N\}_{m=1}^M$ collected from demonstrations contain M demonstrations with each one having fixed length N, where $\mathbf{x}_n^m \in \mathscr{X}$ is the input and $\mathbf{y}_n^m \in \mathscr{Y}$ the output. Here let us consider some cases of robot imitation learning to exemplify the input space $\mathscr{X}$ and output space $\mathscr{Y}$. Probably one of the most basic scenarios is that the robot needs to learn a time-indexed trajectory, i.e. $\mathscr{X} = \mathbb{R}$. For a more complex task such as human-robot collaboration, the robot could be required to react to a human co-worker's position. In this case, the input to the robot is a vector value rather than a scalar, i.e. $\mathscr{X} = \mathbb{R}^{\mathscr{I}}$ with $\mathscr{I} > 1$ being the dimensionality of the input space. Moreover, the output space $\mathscr{Y}$ usually enjoys a Euclidean structure $\mathbb{R}^{\mathscr{O}}$ with $\mathscr{O}$ being the dimensionality of the output space. There are also some problem instances where manifold-type constraints are enforced to the output value, denoted by $\mathscr{Y} = \mathscr{M}$. Such manifold-type constraint usually arises when the imitated trajectory are only allowed to evolve on some subset of the ambient space, which is mathematically modeled with manifolds.

In order to exploit the probabilistic properties from raw demonstration dataset, proper statistical learning tools such as mixture models (Billard et al., 2008) and their various generalizations (Simo-Serra et al., 2017; Zeestraten et al., 2017b) can be employed, namely

$$\{\{\mathbf{x}_n^m, \mathbf{y}_n^m\}_{n=1}^N\}_{m=1}^M \xrightarrow[\text{processing}]{\text{data}} \mathbb{D} = \{\mathbf{x}_n, \tilde{\mathbf{y}}_n\}_{n=1}^N, \tag{2.10}$$

where output $\tilde{\mathbf{y}}$ in the dataset $\mathbb{D}$ lies in a probabilistic space as a result of multiple demonstrations and usually satisfies some Gaussian-like distribution $\mathscr{P}(\mathscr{Y})$.

Here the mechanism of Gaussian Mixture Models/Gaussian Mixture Regression (GMM/GMR) is briefly reviewed. In the context of kinesthetic teaching of a robot manipulator, we assume that we have M demonstrations, each of fixed length N. The dataset comprises the robot's end-effector Cartesian position $\mathbf{x} \in \mathbb{R}^3$ and joint positions $\mathbf{q} \in \mathbb{R}^d$, with d the number of active degrees of

freedom. Both quantities are indexed by time t. Thus, we denote the dataset obtained from the demonstrations as $\{\{t_{n,m}, \mathbf{x}_{n,m}, \mathbf{q}_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}$.

Upon collecting the demonstrations dataset, Gaussian mixture models (GMMs) are employed to model the joint probability distributions $p(t,\mathbf{x})$ and $p(t,\mathbf{q})$, respectively. Without loss of generality, we use $\mathbf{s}_t$ to denote either $\mathbf{x}_t$ or $\mathbf{q}_t$. A GMM with $H \in \mathbb{N}^+$ components is defined by a probability density function

$$p(t,\mathbf{s}_t) = \sum_{h=1}^{H} \eta_h \mathcal{N}(\mu_h, \Sigma_h), \tag{2.11}$$

where

$$\mu_h = \begin{bmatrix} \mu_{t,h} \\ \mu_{s,h} \end{bmatrix}, \tag{2.12}$$

$$\Sigma_h = \begin{bmatrix} \Sigma_{tt,h} & \Sigma_{ts,h} \\ \Sigma_{st,h} & \Sigma_{ss,h} \end{bmatrix}. \tag{2.13}$$

η_h, μ_h and Σ_h are the parameters of the h-th Gaussian component, defining the prior, mean, and covariance, respectively. Note that η_h is subject to $\sum_h \eta_h = 1$. Typically, there are several covariance constraints that can be used in GMM and the one used here is called *full covariance type* (Calinon and Lee, 2016). See Fig. 2.4 for a thorough introduction to different types of covariance matrices. The mixture parameters $\{\eta_h, \mu_h, \Sigma_h\}_{h=1}^{H}$ can be obtained from maximum likelihood estimation using the standard expectation maximization algorithm (Dempster et al., 1977), which consists of a two-step iterative process consisting of an expectation and a maximization step, illustrated as follows. Here we assume a dataset of N observations $\{\xi_i\}_{i=1}^{N}$

E-step:

$$\pi_{i,j} = \frac{\eta_j \mathcal{N}(\xi_i | \mu_j, \Sigma_j)}{\sum_{h=1}^{H} \eta_h \mathcal{N}(\xi_i | \mu_h, \Sigma_h)} \tag{2.14}$$

M-step:

$$\begin{aligned}
\eta_j &\longleftarrow \frac{\sum_{i=1}^{N} \pi_{i,j}}{N}, \\
\mu_j &\longleftarrow \frac{\sum_{i=1}^{N} \pi_{i,j} \xi_i}{\sum_{i=1}^{N} \pi_{i,j}} \\
\Sigma_j &\longleftarrow \frac{\sum_{i=1}^{N} \pi_{i,j} (\xi_i - \mu_j)(\xi_i - \mu_j)^\top}{\sum_{i=1}^{N} \pi_{i,j}}
\end{aligned} \tag{2.15}$$

The above iterative process will continue to repeat until the algorithm converges with respect to some criterion. It should be noted that in this non-convex case, it is not guaranteed that EM can find a global minimum. In fact, the optimization algorithm is sensitive to the initialization values for the mixture model parameters. Therefore, it should be considered to initialize the parameters before running EM algorithm. One example trick could be k-means (MacQueen et al., 1967). As a probabilistic approach, GMM distinguishes it from other techniques in that it can encode the covariance matrices of the demonstrated trajectory variability. This provides the information on how much a movement is allowed to deviate during reproduction, which in turn can be combined with a minimal intervention controller for trajectory tracking (Zeestraten et al., 2016). Followed by further processing with Gaussian Mixture Regression, efficient computation of motor primitives can be achieved to synthesize and generalize demonstrated motion skills.

2.3 Gaussian Mixture Regression

Gaussian mixture regression (GMR) has a simple formulation that has been employed to generate robot movements Calinon (2016). The corresponding output $\hat{\mathbf{s}}(t)$ at each reproduction step t can be estimated in terms of conditional probability:

$$\hat{\mathbf{s}}(t) \sim \sum_{h=1}^{H} w_h(t) \mathcal{N}\left(\hat{\mu}_h(t), \hat{\Sigma}_h(t)\right), \tag{2.16}$$

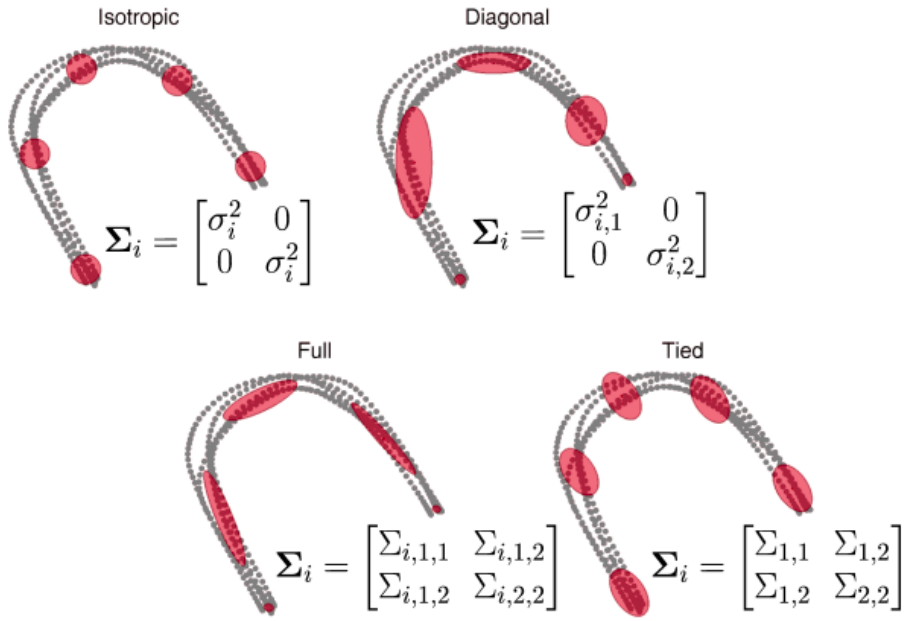


Figure 2.4 Different types of covariance in GMM. (Source from Calinon and Lee (2016))

where $w_h(t)$ are the activation functions defined as

$$w_h(t) = \frac{\eta_h \mathcal{N}(t \mid \mu_{t,h}, \Sigma_{tt,h})}{\sum_{i=1}^{H} \eta_i \mathcal{N}(t \mid \mu_{t,i}, \Sigma_{tt,i})}, \tag{2.17}$$

with

$$\hat{\mu}_h(t) = \mu_{s,h} + \Sigma_{st,h}\Sigma_{tt,h}^{-1}(t - \mu_{t,h}), \tag{2.18}$$

$$\hat{\Sigma}_h(t) = \Sigma_{ss,h} - \Sigma_{st,h}\Sigma_{tt,h}^{-1}\Sigma_{ts,h}. \tag{2.19}$$

Note that (2.16) can also be represented using a unimodal output distribution for the generated trajectory, i.e. $\hat{\mathbf{s}}(t) \sim \mathcal{N}(\hat{\mu}_t^s, \hat{\Sigma}_t^s)$. By resorting to the law of total mean and variance, the approximated normal distribution can be derived

as in Calinon (2016):

$$\hat{\mu}_t^s = \sum_{h=1}^{H} w_h(t)\hat{\mu}_h(t), \tag{2.20}$$

$$\hat{\Sigma}_t^s = \sum_{h=1}^{H} w_h(t)\left(\hat{\Sigma}_h(t) + \hat{\mu}_h(t)\hat{\mu}_h(t)^\top\right) - \hat{\mu}_t^s\hat{\mu}_t^{s\top}. \tag{2.21}$$

GMR has been mostly used in three areas by far:

- as an autonomous system with both position $\mathbf{x}$ and velocity $\dot{\mathbf{x}}$ being considered, i.e. $\xi = [\mathbf{x}^\top, \dot{\mathbf{x}}^\top]^\top$. A series of velocity commands can be retrieved by estimating $\mathscr{P}(\dot{\mathbf{x}}|\mathbf{x})$ during reproduction with GMR (Hersch et al., 2008);
- as time-indexed trajectories, i.e. $\xi = [t, \mathbf{x}^\top]^\top$. The trajectories are smooth as it they are infinitely differentiable (Calinon et al., 2007);
- as probabilistic treatment of dynamic movement primitives (DMP) (Calinon et al., 2012; Ti et al., 2019).

When the trajectory to imitate lies on some Riemannian manifold, (2.20) and (2.21) cannot be applied directly because the weighted sum operation is not defined on a manifold. In order to calculate the mean, we have to computed the weighted mean iteratively in the tangent space $\mathscr{T}_{\hat{\mu}_t^s}\mathscr{M}$ as given by Zeestraten et al. (2017b)

$$\mathfrak{m} = \sum_{h=1}^{H} w_h(t)\mathtt{Log}_{\hat{\mu}_t^s}(\hat{\mu}_h), \tag{2.22}$$

$$\hat{\mu}_t^s \longleftarrow \mathtt{Exp}_{\hat{\mu}_t^s}(\mathfrak{m}). \tag{2.23}$$

The calculation of the covariance follows similarly.

An illustrative example of the mechanism of GMR is shown in Fig 2.5.

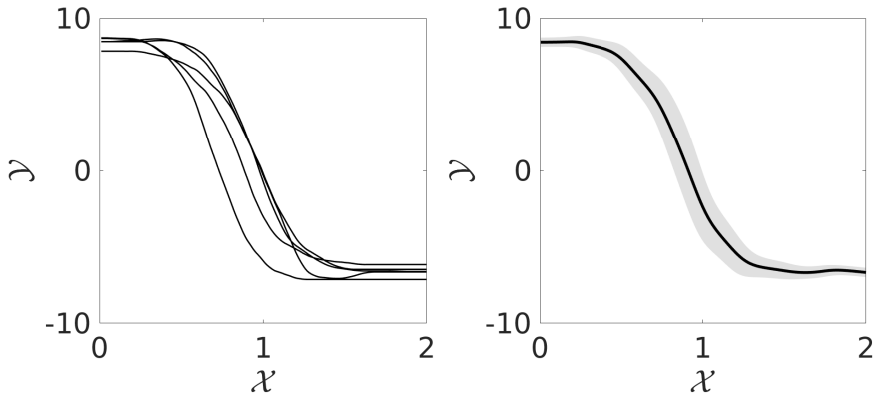


Figure 2.5 An illustrative example of data processing for probabilistic imitation learning. The *left* figure plots multiple demonstrations and the *right* figure plots the obtained probabilistic trajectory using GMR with the solid line represents the mean and the shallow area represents the covariance.

Chapter 3

Kernel Methods

In this chapter, we review some basics on kernel methods, which will facilitate understanding of subsequent chapters.

3.1 Regularized Least Squares

In this section, we review regularized least squares, a fundamental supervised learning method that modifies the least squares model with Tikhonov regularization. Formally, given n input-output training data pairs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^d$ is the i-th input and $\mathbf{y}_i \in \mathbb{R}^k$ its corresponding output, the learning problem is formulated as

$$\min_{\mathbf{w}\in\mathbb{R}^{d\times k}} \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{y}_i - \mathbf{w}^\top \mathbf{x}_i\|^2 + \lambda \|\mathbf{w}\|_F^2, \tag{3.1}$$

where $\|\cdot\|_f^2$ denotes the Frobenius norm and λ is a regularization parameter. By setting the gradient of (3.1) with respect to $\mathbf{w}$ equal to zero, we obtain the solution of the regularized least squares

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + n\lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{Y}, \tag{3.2}$$

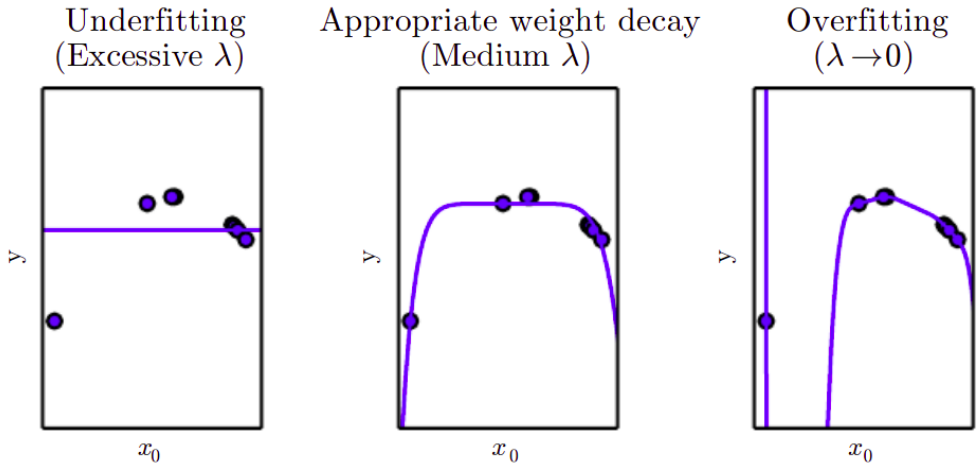


Figure 3.1 Schematic illustration of the effects for different λ. (Source: Goodfellow et al. (2016))

where

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}, \tag{3.3}$$

$$\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]^\top \in \mathbb{R}^{n \times k}. \tag{3.4}$$

It should be noted that the parameter λ controls the model complexity by regulating the trade-off between the sum-of-squares error and the regularization term. When λ is too big, model complexity will be penalized. Thus it cannot capture the dynamics of data, which causes the issue of underfitting. Likewise, if the value of λ is chosen to be too small, the model will be powerful enough to model the underlying noise from data. And this will case the overfitting problem. A pictorial illustration of overfitting and underfitting problem is shown in Fig. 3.1.

The type of regularization term we choose in (3.1) is the quadratic regularizer. A more general form of the regularizer can be also considered

$$\lambda \sum_{i=1}^{d} \sum_{j=1}^{k} \|\mathbf{w}_{i,j}\|^q.$$

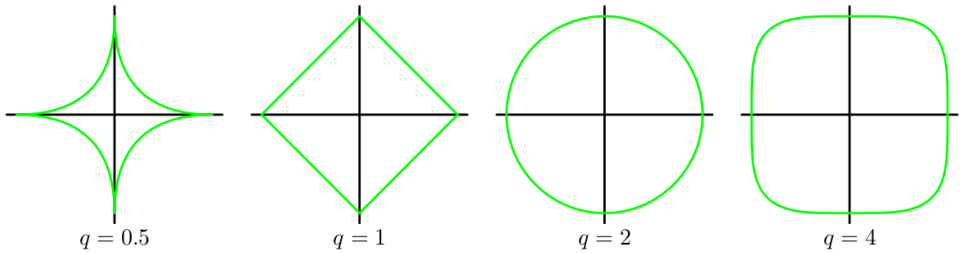


Figure 3.2 Schematic illustration of the contours for different values of q. (Source: Bishop (2006)).

When $q = 1$, it is also known as the lasso regularizer in the statistics literature. Contours of different q are shown in Fig. 3.2.

3.2 Linear Models with Nonlinear feature

Recall that we employed a linear model for learning the underlying relationship between the given input and output data. This could be restrictive when the relationship contains complex nonlinearity. Therefore, a model that can capture nonlinear relationship is usually desired. To this end, we introduce feature map, a key concept that allows the rugularized least squares model to be generalized to nonlinear models.

Formally, a feature map is a map

$$\boldsymbol{\phi} : \mathscr{X} \to \mathscr{V}$$

from the input space $\mathscr{X}$ into a new space $\mathscr{V}$ called the feature space. The feature space is also associated with a scalar product expressed by $\langle \cdot, \cdot \rangle_{\mathscr{V}}$. The feature space could be finite or even infinite dimensional. With the feature map, the linear model in (3.1) can be replace by

$$\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}), \tag{3.5}$$

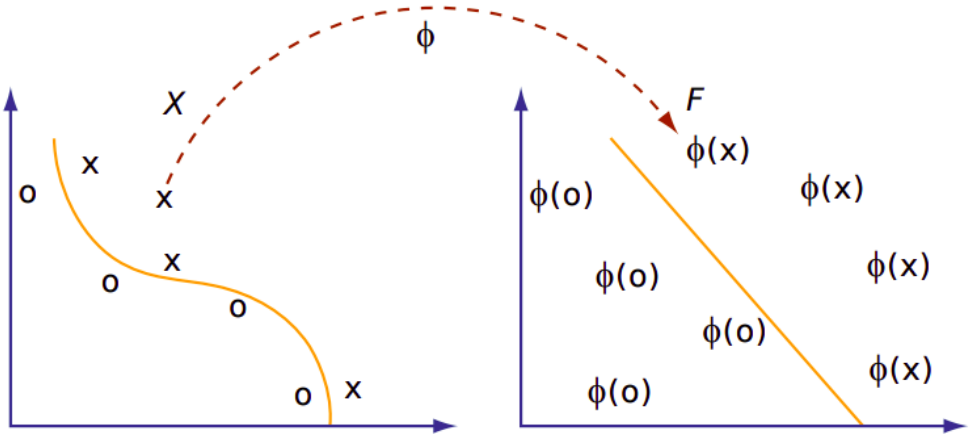


Figure 3.3 Schematic illustration of the effects of a nonlinear feature. A hard-to-solve classification problem with a linear model can be solved easily with the help of a feature map (Image by MIT OpenCourseWare).

which converts the regularized sim-of-squares error function into

$$\frac{1}{n}\sum_{i=1}^{n}\|\mathbf{y}_i-\mathbf{w}^\top\phi(\mathbf{x}_i)\|^2+\lambda\|\mathbf{w}\|_F^2. \tag{3.6}$$

With a slight abuse of notation, $\mathbf{w}$ denotes the new weight matrix with the dimension in accordance to that of the feature map. Although seeming simple, the model (3.5) allows to capture very complex data relationships. Problems that are not easy to solve with a linear model can be easily tackled with a linear model in the feature space. An illustrative example is shown in Fig. 3.3.

Therefore given a new input $\mathbf{x}$, By setting the gradient of (3.6) with respect to $\mathbf{w}$ equal to zero, in analogy to (3.2), we obtain

$$\mathbf{w}=(\Phi\Phi^\top+n\lambda\mathbf{I})^{-1}\Phi\mathbf{Y}=\Phi(\Phi^\top\Phi+n\lambda\mathbf{I})^{-1}\mathbf{Y}, \tag{3.7}$$

where we used the trick of the following identity (Welling, 2002):

$$(P^{-1}+B^\top R^{-1}B)^{-1}B^\top R^{-1}=PB^\top(BPB^\top+R)^{-1}.$$

By substituting (3.7) into (3.5), the predicted value given a new test point is

$$\mathbf{w}^\top \phi(\mathbf{x}) = \mathbf{Y}(\Phi^\top \Phi + n\lambda \mathbf{I})^{-1} \Phi^\top \Phi = \mathbf{Y}(\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{k}(\mathbf{x}), \tag{3.8}$$

where $\mathbf{K}_{i,j} = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$ and $\mathbf{k}(\mathbf{x}) = \mathbf{K}(\mathbf{x}_i, \mathbf{x})$. The important implication here is that we do not need access to nonlinear feature map but only their inner product, which is defined as the kernel function. The detailed properties of the kernel function shall be revealed in the subsequent section.

3.3 Kernels

The formal definition for kernel is as follows. Let $\mathscr{X}$ be a non-empty set. A function $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ is a kernel if there exists a Hilbert space $\mathscr{H}$ associated with an inner product $\langle \cdot, \cdot \rangle_{\mathscr{H}}$ and a feature map $\phi : \mathscr{X} \to \mathscr{H}$ such that for any $\mathbf{x}, \mathbf{x}' \in \mathscr{X}$,

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathscr{H}} \\ &= \sum_{i=1}^{M} \phi_i(\mathbf{x}) \phi_i(\mathbf{x}'), \end{aligned} \tag{3.9}$$

where $\phi_i(\cdot)$ are the basis functions. Different basis functions choice can lead to different kernel functions as illustrated in Fig. 3.4.

Instead of manually picking specific basis functions to build a kernel function, an alternative approach is to construct a kernel function directly.

Example (Polynomial Kernel)

As a simple example, consider a polynomial kernel defined as

$$k(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^\top \mathbf{b})^3. \tag{3.10}$$

We can do sanity check that (3.10) is indeed a valid kernel by identifying its corresponding nonlinear feature map. We can take the particular case of a two dimensional input space, i.e. $\mathbf{a} = (a_1, a_2)$ and $\mathbf{b} = (b_1, b_2)$. Thus, their kernel

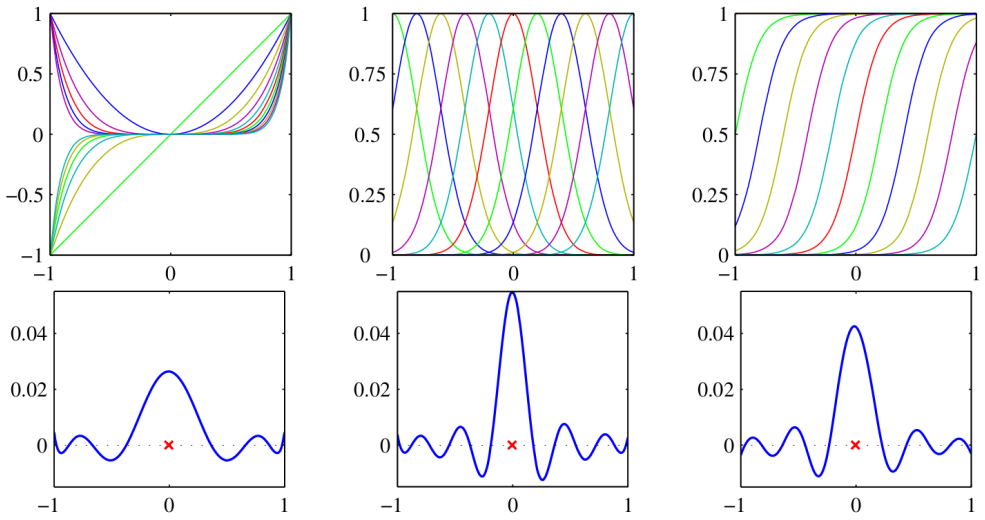


Figure 3.4 Illustration of polynomial, Gaussians, and logistic sigmoid basis functions (*upper row* from left to right) and the corresponding kernels (*bottom row*). The red points denote the evaluation points of the kernel functions. (Image from Bishop (2006)).

value becomes

$$\begin{aligned} k(\mathbf{a},\mathbf{b}) &= (\mathbf{a}^\top \mathbf{b})^3 = ((a_1,a_2)^\top (b_1,b_2))^3 \\ &= (a_1 b_1 + a_2 b_2)^3 \\ &= a_1^3 b_1^3 + 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \\ &= (a_1^3, \sqrt{3}a_1^2 a_2, \sqrt{3}a_1 a_2^2, a_2^3)^\top (b_1^3, \sqrt{3}b_1^2 b_2, \sqrt{3}b_1 b_2^2, b_2^3) \\ &= \phi(\mathbf{a})^\top \phi(\mathbf{b}). \end{aligned} \tag{3.11}$$

Therefore, we can see that the corresponding nonlinear feature map to the kernel (3.10) has the form

$$\phi(\mathbf{x}) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)^\top.$$

In general, it will be difficult to determine weather a kernel is valid by explicitly constructing its corresponding feature map. For a function $k(\mathbf{x},\mathbf{x}')$ to be a valid kernel, a necessary and sufficient condition is that the Gram matrix $\mathbf{K}$, whose elements are given by $k(\mathbf{x}_i,\mathbf{x}_j)$, should be positive semidefinite for

all possible choices of the set $\{\mathbf{x}_n\}$. In practice, there are a few types of kernels that are commonly used:

- **Polynomial**
  $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^d$, where d is the degree of the polynomial.
- **Gaussian**
  $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$.
- **Sigmoid**
  $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta(\mathbf{x}_i^\top \mathbf{x}_j) + a)$.

It should be noted here that although the names of the kernels are the same as those of the basis functions as shown in Fig. 3.4, there are fundamental differences. Foe example, the kernel function built upon Gaussian basis functions have finite dimension of the feature map, which is the same as the number of the basis functions. While the Gaussian kernel has infinite dimension of feature vector.

In addition to the existing basic kernel functions, one powerful technique to construct new kernels is to build them based on simpler kernels. Below lists a few representative tricks to construct new kernels. For a more extensive instructions of constructing new kernels, please refer to Bishop (2006).

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= ck_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q(k_1(\mathbf{x}, \mathbf{x}')) \\
k(\mathbf{x}, \mathbf{x}') &= \exp(k_1(\mathbf{x}, \mathbf{x}')) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^\top \mathbf{A}\mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
\tag{3.12}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ represents a polynomial with non-negative coefficients, $\phi(\mathbf{x}) : \mathbf{x} \longrightarrow \mathbb{R}^M$ is a function, $k_3(,)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and $k_a(,)$ and $k_b(,)$ are valid kernel functions over their corresponding spaces.

Chapter 4

Structured Prediction

In the context of robot imitation learning, one of the most concerning issues is to generate a trajectory such that a robot can imitate a demonstrator as closely as possible. Therefore, we can understand imitation learning from such perspective: Given a query of input, the robot should respond the way as the demonstrator does. Consequently, it is critical to find a suitable mapping rule between the input and output values based on the collected human demonstration data. Indeed, this classical perspective of imitation learning is reminiscent of the objective of supervised learning: To find a functional relation between an input space and an output space. In this thesis, we focus on probabilistic imitation learning. Our goal is to address the motion imitation issue by learning the mapping rule $\mathbf{s} : \mathscr{X} \to \mathscr{P}(\mathscr{Y})$ underlying the dataset $\mathbb{D}$. The problem representation is conceptualized as

$$\text{find } \mathbf{s} : \mathscr{X} \to \mathscr{P}(\mathscr{Y}) \quad \text{given } \{\mathbf{x}_n, \tilde{\mathbf{y}}_n\}_{n=1}^{N}. \tag{4.1}$$

Before addressing problem (4.1) directly, let us first consider the deterministic case, namely

$$\text{find } \mathbf{s} : \mathscr{X} \to \mathscr{Y} \quad \text{given } \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}. \tag{4.2}$$

It shall be noted that problem (4.2) is characterized as a structured prediction problem as $\mathscr{Y}$ could possess complex structures such as manifolds (Ciliberto et al., 2016).

4.1 Structured Prediction via Surrogate Approach

Due to the complex structure of the output space, solving a structured prediction problem is usually not straightforward. By contrast, vector-valued prediction problem is well studied and theoretically sound (Álvarez et al., 2012). It is thus natural to think of transcribing a structured prediction problem into a vector-valued problem so that structured prediction can become tractable. Following this line, we address structured prediction with a strategy called surrogate frameworks in the literature (Ciliberto et al., 2016). In brief, the core idea underlying the surrogate approach to structured prediction is to 1) design an encoding rule which embeds the structural output values into a linear space, 2) solve the learning problem efficiently in the surrogate space instead, and finally 3) interpret the learned surrogate solution with a proper decoding rule.

More formally, the procedure of a surrogate method for structured prediction consists of the following three steps:

1. *Encoding*: $\mathbf{c} : \{-1, 1\} \to \mathbb{R}$ the identity map.
2. *Learning*: $\mathbf{g} : \mathscr{X} \to \mathbb{R}$ which minimizes $\mathscr{L}$ given the surrogate dataset $\{\mathbf{x}_n, \mathbf{c}(\mathbf{y}_n)\}_{n=1}^N$.
3. *Decoding*: Choose a decoding $\mathbf{c}^{-1} : \mathscr{H} \to \mathscr{Y}$.
   Recover $\mathbf{s} = \mathbf{c}^{-1} \circ \mathbf{g} : \mathscr{X} \to \mathscr{Y}$.

A pictorial illustration of structured prediction by surrogate framework is shown in Fig. 4.1.

Example (Binary Classification)

In the task of binary classification, the goal is to learn a binary-valued function $\mathbf{s} : \mathscr{X} \to \mathscr{Y} = \{-1, 1\}$. The procedure to address this illustrative problem with structured prediction by surrogate framework is as follows.

1. *Encoding*: $\mathbf{c} : \mathscr{Y} = \{-1, 1\} \to \mathscr{H} = \mathbb{R}$ is the identity map.
2. *Scalar learning*: $\mathbf{g}_n : \mathscr{X} \to \mathbb{R}$ which minimizes $\mathscr{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ (e.g. least-squares, hinge, logistic, etc.).
3. *Decoding*: Choose the decoding rule as $\mathbf{c}^{-1} = \texttt{sign} : \mathbb{R} \to \{-1, 1\}$.

Finally, the classifier is then obtained as

$$\mathbf{s}_n(x) = \texttt{sign}(\mathbf{g}_n(x)).$$

Example (Multi-class Classification)

In the task of multi-class classification, the goal is to learn a binary-valued function $\mathbf{s} : \mathscr{X} \to \mathscr{Y} = \{1, 2, \ldots, M\}$. The procedure to address this illustrative problem with structured prediction by surrogate framework is as follows.

1. *Encoding*: $\mathbf{c} : \mathscr{Y} \to \{e_1, e_2, \ldots, e_M\} \subset \mathbb{R}^M$ is canonical basis with $\mathbf{c}(j) = e_j \in R^M$
2. *Multi-variate learning*: $\mathbf{g}_n : \mathscr{X} \to \mathbb{R}^M$.
3. *Decoding*: Choose the decoding rule as $\mathbf{c}^{-1} : \mathbb{R}^M \to \mathscr{Y} = \{1, 2, \ldots, M\}$.

Finally, the multi-classifier is then obtained as

$$\mathbf{s}_n(x) = \underset{j=1,\ldots,M}{\texttt{argmax}} \quad e_j^\top \mathbf{g}_n(x).$$

4.2 Implicit Encoding Framework

Although the aforementioned surrogate framework is capable of addressing structured prediction, it is restricted to special cases such as classification (Mroueh et al., 2012), ranking (Duchi et al., 2010), multi-labeling (Gao and Zhou, 2011) etc. In addition, explicit design of encoding and decoding rules

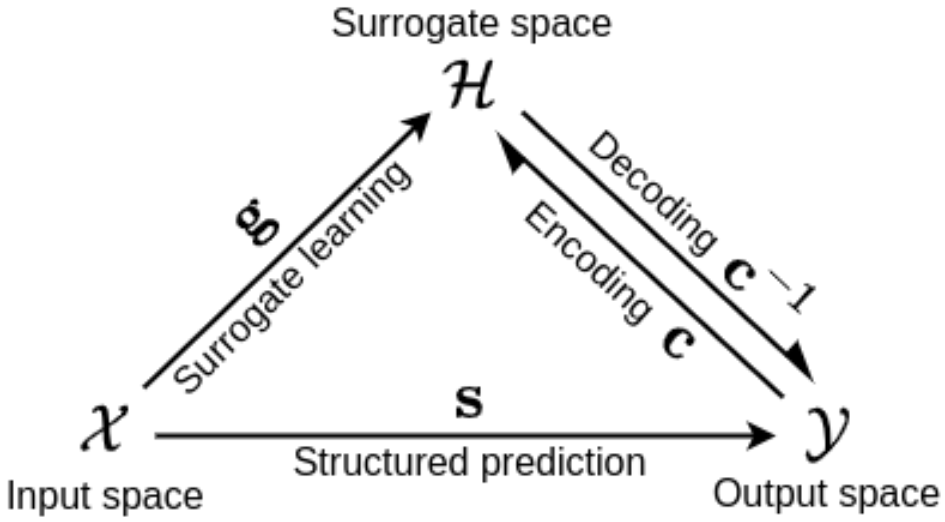


Figure 4.1 Schematic illustration of the surrogate approach to structured prediction.

can be onerous as it is ad hoc for each learning problem. Here we employ a general relaxation approach, which allows for the training in the surrogate space implicit (Ciliberto et al., 2020). To start with, a key structural assumption on the loss is introduced. Formally, $\Delta : \mathscr{Y} \times \mathscr{Y} \to \mathbb{R}$ is called a Structure Encoding Loss Function (SELF) if there exist a separable Hilbert space $\mathscr{H}_{\mathscr{Y}}$ with inner product $\langle \cdot, \cdot \rangle_{\mathscr{H}_{\mathscr{Y}}}$, a continuous feature map $\mathbf{c} : \mathscr{Y} \to \mathscr{H}_{\mathscr{Y}}$, and a continuous linear operator $V : \mathscr{H}_{\mathscr{Y}} \to \mathscr{H}_{\mathscr{Y}}$ such that for all $\mathbf{y}, \mathbf{y}' \in \mathscr{Y}$

$$\Delta(\mathbf{y}, \mathbf{y}') = \langle \mathbf{c}(\mathbf{y}), V\mathbf{c}(\mathbf{y}') \rangle_{\mathscr{H}_{\mathscr{Y}}}. \tag{4.3}$$

For the encoding step, we assume $\mathbf{c} : \mathscr{Y} \to \mathscr{H}_{\mathscr{Y}}$ is a canonical feature map of $\mathscr{H}_{\mathscr{Y}}$. For the surrogate learning step, we consider to use multi-variate learning with ridge regression. Therefore, $\mathbf{g}$ is parameterized by

$$\mathbf{g}(\mathbf{x}) = \mathbf{W}\boldsymbol{\varphi}(\mathbf{x}) \in \mathbb{R}^M, \tag{4.4}$$

where $\varphi : \mathscr{X} \to \mathbb{R}^P$ is a feature map and

$$\begin{aligned}\widehat{\mathbf{W}} =& \underset{\mathbf{W}\in\mathbb{R}^{M\times P}}{\texttt{argmin}} \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{W}\varphi(\mathbf{x}_n) - \mathbf{c}(\mathbf{y}_n)\|^2_{\mathscr{H}_\mathscr{Y}} + \lambda\|\mathbf{W}\|_F^2 \\ =& \mathbf{CA}, \end{aligned} \tag{4.5}$$

where

$$\mathbf{C} = [\mathbf{c}(\mathbf{y}_1), \ldots, \mathbf{c}(\mathbf{y}_N)] \in \mathbb{R}^{M\times N}$$

represent output features, and

$$\mathbf{A} = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top \in \mathbb{R}^{N\times N}$$

with input features

$$\mathbf{\Phi} = [\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_N)] \in \mathbb{R}^{P\times N},$$

and $\|\cdot\|_F^2$ denotes the squared Frobenius norm of a matrix, i.e. the sum of all its squared elements. By substituting (4.5) into (4.4), the solution to the surrogate function is obtained as

$$\widehat{\mathbf{g}}(\mathbf{x}) = \mathbf{CA}\varphi(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n(\mathbf{x})\mathbf{c}(\mathbf{y}_n), \tag{4.6}$$

where $\alpha_n(\mathbf{x})$ is the n-th entry of $\alpha(\mathbf{x}) \in \mathbb{R}^N$:

$$\alpha(\mathbf{x}) = [\alpha_1(\mathbf{x}), \ldots, \alpha_N(\mathbf{x})]^\top \tag{4.7}$$

$$= \mathbf{A}\varphi(\mathbf{x}) \tag{4.8}$$

$$= (\mathbf{K} + N\lambda\mathbf{I}_N)^{-1}\mathbf{k}_x. \tag{4.9}$$

Here, $\lambda > 0$ is a regularization parameter and $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ denotes the identity matrix of size N. Given a positive definite kernel $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$, the empirical kernel matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$ is constructed as $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $\mathbf{k}_x \in \mathbb{R}^N$ is a vector defined by

$$\mathbf{k}_x = [k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_N)]^\top$$

.

Finally, the decoding rule of relaxation with SELF is designed as

$$\mathbf{s}(\mathbf{x}) = \underset{\mathbf{y}\in\mathscr{Y}}{\texttt{argmin}} \left\langle \mathbf{c}(\mathbf{y}), V\mathbf{g}(\mathbf{x}) \right\rangle_{\mathscr{H}_\mathscr{Y}}. \tag{4.10}$$

By plugging (4.6) into (4.10), we have

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{\mathbf{y}\in\mathscr{Y}}{\texttt{argmin}} \left\langle \mathbf{c}(\mathbf{y}), V\left(\sum_{n=1}^{N} \alpha_n(\mathbf{x})\mathbf{c}(\mathbf{y}_n)\right)\right\rangle_{\mathscr{H}_\mathscr{Y}} \tag{4.11}$$

$$= \underset{\mathbf{y}\in\mathscr{Y}}{\texttt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x})\Delta(\mathbf{y},\mathbf{y}_n), \tag{4.12}$$

where we move the coefficients α_n out of the inner product with the property of linearity and use the definition of the loss function (4.3).

To apply (4.12) to solve the structured prediction problem (4.2), there are two steps to follow:

1. *Surrogate learning*: Calculate the input-dependent weights α.

2. *Decoding*: Optimize the α-weighted linear combination of losses $\Delta(\mathbf{y},\mathbf{y}_n)$.

It can be observed that the surrogate space $\mathscr{H}_\mathscr{Y}$ is no loner needed to design explicitly and hence an implicit encoding framework is admitted.

Chapter 5

Robotic Platforms

In this chapter, the main robotic platforms involved during the thesis are introduced, namely the iCub humanoid robot and the KUKA LWR IV+ robot manipulator.

5.1 iCub humanoid robot

The iCub humanoid robot is a state-of-the-art humanoid robot platform developed at Italian Institute of Technology (Metta et al., 2008). Originally as a result of the European project RobotCub, the iCub humanoid robot platform has been disseminated rapidly and the platforms reach out not only robotics labs in Europe, but also as far as Asia and America. By the time of this thesis writing, there are at least 42 iCubs around the world being used [1]. Such high acceptance and popularity makes the iCub becomes a de facto standard humanoid platform for studying embodied intelligence. iCub is an ideal platform to conduct a wide spectrum of artificial intelligence related robotics research, such as tele-existence (Elobaid et al., 2019), human-robot interaction (Bossi et al., 2020), object recognition (Pasquale et al., 2019), event driven vision (Rea et al., 2013) etc.

As the improvements of the robot keep making progresses and the iCub copies are customized towards the specific research purposes of the hosting

[1] For up-to-date information on the iCub, please visit https://icub.iit.it/

lab, there are several versions for the iCub. The one introduced in this section is internally referred to as iCub 2.5, used by the Dynamic Interaction Control research line [2] at iit.

In general, the iCub is a child-sized humanoid robot as shown in Fig 5.1. It has a height of 104 cm and weighs 23 kg. There are in total 53 actuated degrees of freedom, which are distributed as 6 for the head and eyes, 7 for each arm, 9 for each hand, 3 for the torso, and 6 for each leg. A summary of iCub's specification is given by Table 5.1.

Table 5.1 Specification for the iCub (Adapted from Nava (2020))

Height	104 cm
Weight	23 kg
Sensors	Stereo Cameras, microphones, encoders, force/torque sensors, tactile sensors, gyroscopes, accelerometers
Actuators	Brushless motors, DC motors.
Computing (On-board)	20 micro-controller boards for movement, 16 boards for sensors and a Pentium duo for data acquisition and synchronization
Software (On-board)	Linux
Middleware	YARP
Degrees of Freedom	53 motors controlling 76 joints
Structure and Materials	Ergal, steel, plastic
Chronology	Started 2004, first release 2008

On the software aspect, iCub is usually shipped with the software Yet Another Robot Platform (YARP) [3], a middleware with the purpose of facilitating low-latency communications between different components of the robot (Metta et al., 2006). As YARP is able to provide interfaces independently from the actual implementation, code reuse and modularity are facilitated.

[2] Lab website: https://dic.iit.it/

[3] Project webpage: www.yarp.it

Figure 5.1 The iCub humanoid robot.

To control the robot, it is usually necessary to calculate kinematic and dynamic quantities. To this end, the WBToolbox library is employed (Romano et al., 2017). Based on the dataflow framework blockfactory, the WBToolbox library wraps the functionalities of YARP and of rigid-body dynamics library iDynTree. An example of using WBToolboxfor prototyping a momentum controller in simulink is shown in Fig 5.2.

5.2 KUKA LWR IV+

The other robot used during the thesis is a KUKA Lightweight robot LWR IV+, a robot manipulator developed for collaborative tasks. It has been adopted by many robotics lab worldwide, which makes the KUKA LWR robot a reference manipulator platform for robotics research, as shown in Fig 5.3. As the name implies, it has a very low weight of 16 kg. In total it has seven axes, which contributes to making it reachable to a wide range of configuration space. A more complete specifications of the KUKA LWR IV+ robot manipulator is summarized in Table 5.2.

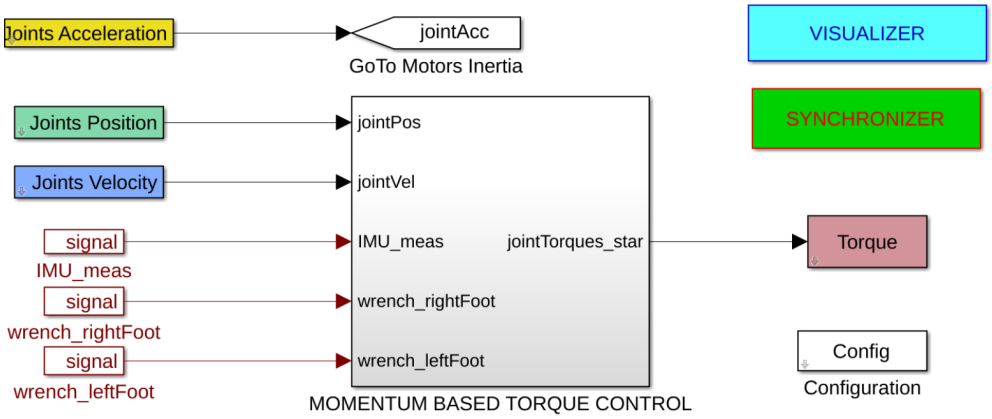


Figure 5.2 An example of using WBToolbox in simulink. (Source: Nava (2020)).

Table 5.2 Specification for the iCub (Adapted from Nava (2020))

Payload	7 kg
Number of axes	7
Wrist variant	in-line wrist
Mounting position	Any
Repeatability	± 0.05mm
Weight	16 kg

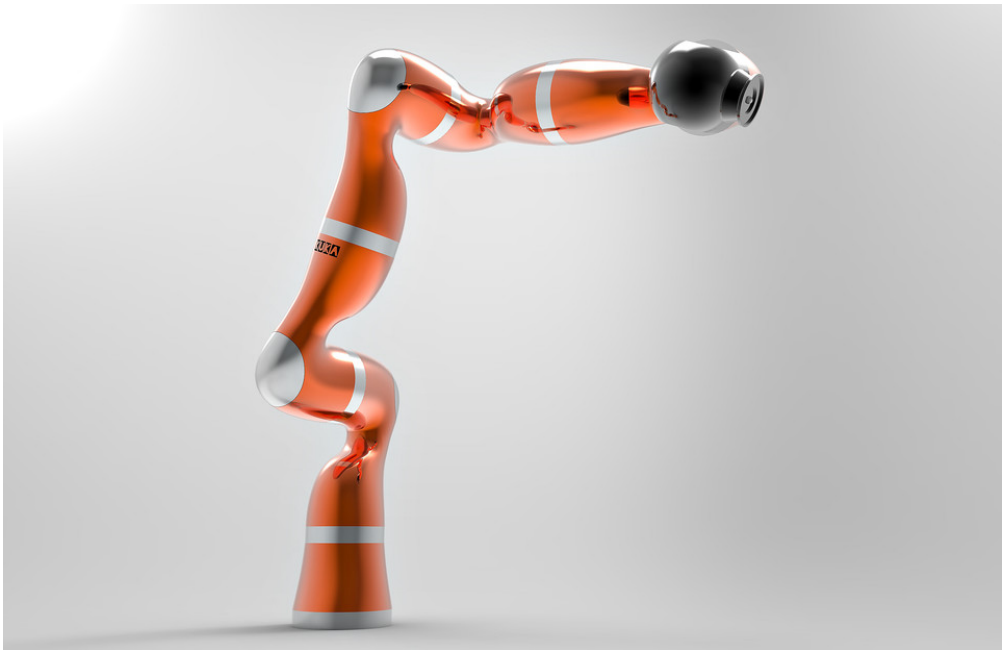


Figure 5.3 The KUKA LWR IV+ (Source of the image: https://grabcad.com/library/kuka-lwr-4-arm-1).

Part II

Contribution

Chapter 6

Imitation by f-Divergence minimization

Upon observing that the regression methods heavily influence the design of imitation learning algorithms (Osa et al., 2018), we consider to develop our approach from a supervised-learning perspective. Specifically, we lay the theoretical foundation on structured prediction, which targets learning tasks with complex output space structures (Ciliberto et al., 2017). In this chapter, we present our algorithmic framework that allows for a uniform approach to probabilistic imitation learning, followed by trajectory modulation strategies. Our devised algorithmic framework is persistent in that it can handle imitation of trajectories lying in Euclidean space or on a manifold. Our proposed approach also preserves essential features for a movement primitive. Similar to KMP, our algorithm also shares a non-parametric formalism and therefore alleviates the burden of manually setting basis functions. In addition, our algorithm supports learning of trajectories driven by multi-dimensional inputs thanks to its kernel formulation.

We seek to tackle robot imitation learning with tools from supervised learning. Specifically, we highlight structured prediction as it can predict output with complex structures. Thanks to the generality of structured prediction, various types of trajectory output including but not limited to manifold-valued ones can be predicted. It should be noted that our output estimator allows us to construct

a customized loss function between the predicted output and reference value. This gives rise to the core of our approach: We seek to guarantee imitation fidelity based on an insightful finding that imitation learning is closely related to the f-divergence minimization between the expert's and the learner's trajectory distributions. Hence, we propose to construct the loss function by leveraging tools from information theory, namely the family of f-divergences. Such point of view towards imitation learning has gradually attracted attention, as the evidenced by recent literature (Ghasemipour et al., 2019; Ke et al., 2019). In fact, a number of existing imitation learning algorithms can be regarded as minimization of some f-divergence. For example, behavior cloning minimizes the Kullback-Leibler (KL) divergence (Pomerleau, 1989), KMP minimizes the reverse KL divergence (Huang et al., 2019) and GAIL minimizes the Jensen-Shannon (JS) divergence (Ho and Ermon, 2016). In this thesis, we show that by plugging in different divergences as loss functions, we are able to obtain different imitation strategies. We coin the imitation strategies emerging from different choices of f-divergences as *imitation modes*, identifying a novel general framework for supervised learning based motion imitation.

A comparison between our proposed approach and the state-of-the-art is provided in Table 6.1. To summarize, our contribution is the development of an algorithmic framework, which allows for

(i) Prediction of various types of structured output (Euclidean or manifold trajectories);

(ii) Imitation of expert demonstrations with a variety of imitation modes by means of different f-divergences;

(iii) Spatial and temporal trajectory modulation in a convenient and straightforward manner.

Table 6.1 Comparison of features with respect to state-of-the-art methods.

	Probabilistic	Trajectory modulation	Manifold output	Manifold modulation	Multiple imitation modes	Multi-dim input	Multiple output types
DMP (Ijspeert et al., 2013)	-	-	-	-	-	-	-
ProMP (Paraschos et al., 2013)	✓	✓	-	-	-	-	-
LAT (Reiner et al., 2014)	✓	-	-	-	-	-	-
GMM (Zeestraten et al., 2017b)	✓	-	✓	-	-	✓	-
LGP (Schneider and Ertel, 2010)	✓	-	-	-	-	-	-
TLGC (Ahmadzadeh and Chernova, 2018)	-	-	✓	-	-	-	-
KMP (Huang et al., 2019)	✓	✓	-	-	-	✓	-
LPV-DS (Figueroa and Billard, 2018)	-	✓	✓	✓	-	✓	-
Our Approach	✓	✓	✓	✓	✓	✓	✓

6.1 Loss Function Design by f-Divergence

Recall that under the context of probabilistic imitation learning with multiple demonstrations, our goal is to address problem (4.1) where unlike the deterministic case the output space in our dataset $\mathbb{D}$ is a probability space. By naively giving an input $\mathbf{x}$ to (4.12) for prediction of the optimal output distribution, we have

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{\tilde{\mathbf{y}} \in \mathscr{P}}{\texttt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Delta(\tilde{\mathbf{y}}, \tilde{\mathbf{y}}_n). \tag{6.1}$$

It should be noted that the formulation as (6.1) does not hold because the output space is not a compact set anymore (Ciliberto et al., 2017). Yet we still adopt the formulation temporarily and we will later elaborate in Chapter 7 on how to deal with this issue under the common assumption that the output probability satisfies the Gaussian distribution.

Given the estimator (6.1), our objective now becomes constructing a proper loss function $\Delta(\tilde{\mathbf{y}}, \tilde{\mathbf{y}}_n)$ which is capable of measuring the distance between two probability distributions $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{y}}_n$. To this end, we leverage the tools from an information-theoretic perspective, specifically the family of f-divergences (Sason, 2018). In fact, the importance of the role of f-divergence is gradually attracting attention in the field of imitation learning. One insightful finding is to view imitation learning as divergence minimization between expert and learner trajectory distributions (Ghasemipour et al., 2019; Ke et al., 2019). The f-divergence generalizes similarity measures between probability distributions. Its expression is given as[1]

$$D_f\big(\tilde{\mathbf{y}}_n(\mathbf{x}), \tilde{\mathbf{y}}(\mathbf{x})\big) \triangleq \mathbb{E}_{\tilde{\mathbf{y}}(\mathbf{x})}\left[f\left(\frac{d\tilde{\mathbf{y}}_n(\mathbf{x})}{d\tilde{\mathbf{y}}(\mathbf{x})} \right) \right], \tag{6.2}$$

where $f : R^+ \to R$ is a convex function with $f(1) = 0$. By choosing a different f, a broad class of divergences can be recovered. Common choices include the KL divergence, reverse KL divergence, and the Jensen-Shannon divergence (see Pardo (2018) for a full list). As we will end up with different policies using

[1] We put "true" distribution first to comply with forward KL divergence.

different types of f-divergence, we refer to these different imitation policies as *imitation modes*.

Finally, by substituting (6.2) into (6.1), we obtain the estimator as

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{\tilde{\mathbf{y}} \in \mathscr{P}}{\texttt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x}) \mathbb{E}_{\tilde{\mathbf{y}}(\mathbf{x})} \left[f\left(\frac{d\tilde{\mathbf{y}}_n(\mathbf{x})}{d\tilde{\mathbf{y}}(\mathbf{x})} \right) \right]. \tag{6.3}$$

6.2 Trajectory Modulation

One of the most concerning challenges in robot imitation learning is to modulate the learned motor skills to meet additional constraints. The capability of adaptation can make robots more flexible in cluttered environments. Indeed, it is usually the case that the trajectory demonstration and reproduction contexts differ. For example, modulating the trajectory shape by setting a series of desired *via-points* can be a convenient collision-avoidance strategy. Furthermore, being able to set a different trajectory *end-point* can also have practical usages, e.g. in pushing or pick-and-place tasks. Our approach guarantees that the movement can be adapted anytime during the execution of the trajectory. Moreover, in order to generate more complex behaviors, multiple movement trajectories can be co-activated simultaneously (Duan et al., 2019). Such trajectories combination can also significantly improve motion expressiveness. To do so, each single trajectory is assigned its own priority with the total sum of the priorities normalized. Such concurrent co-activation of different trajectories is also known as trajectory *superposition*.

Besides spatial modulation, temporal modulation is also an important property as it can further generalize trajectory imitation by speeding up or slowing down the robot movement. Some tasks could be sensitive to correct timing, e.g. striking-based manipulation or walking speed adjustment for stable locomotion. Temporal modulation can also be used to avoid time-dependent collisions, thus enhancing safety in a human-robot interactive scenarios (Koert et al., 2019).

Algorithm 1: Imitation Learning by Structured Prediction

Initialization:

Retrieve dataset $\mathbb{D}$ from demonstrations;
Define kernel k and hyperparameter λ;
Choose imitation mode f;

Trajectory modulation:

Specify desired points in $\mathbb{D}_v$;
Aggregate dataset as $\mathbb{D} \cup \mathbb{D}_v$;

Motion generation:

Input: query point $\mathbf{x}$;
Calculate weights $\alpha'(\mathbf{x})$;
Output: estimated value $\widehat{\mathbf{s}}(\mathbf{x})$;

6.2.1 Spatial Modulation

Here, we address the issue of passing through additional desired *via-points*. Assume that there are J new desired points stored in the dataset $\mathbb{D}_v = \{\mathbf{x}_j, \tilde{\mathbf{y}}_j\}_{j=1}^{J}$ with each one repeating for $w_j > 1$ times in the dataset. In order to take these new requirements into account, we concatenate the new desired dataset to the original demonstrated one. Consequently, the updated dataset to train our estimator now becomes $\mathbb{D} \cup \mathbb{D}_v$ with a total number of points $N' = N + \sum_j^J w_j$. Let us illustrate the mechanism of adaptation towards new desired via-points in the deterministic case. The optimization problem of (4.5) now becomes

$$\underset{\mathbf{W} \in \mathbb{R}^{M \times P}}{\texttt{argmin}} \frac{1}{N'} \sum_{n=1}^{N+J} w_n \|\mathbf{W}\boldsymbol{\varphi}(\mathbf{x}_n) - \mathbf{c}(\mathbf{y}_n)\|^2_{\mathscr{H}_\mathscr{Y}} + \lambda \|\mathbf{W}\|^2_F \tag{6.4}$$

where the weight w_n is defined as

$$w_n = \begin{cases} w_j & \mathbf{x}_n \in \mathbb{D}_v, \\ 1 & \texttt{otherwise}. \end{cases} \tag{6.5}$$

The estimator can still be expressed as in (4.12), except that the coefficients now become

$$\alpha'(\mathbf{x}) = (\mathbf{K}' + N'\lambda \mathbf{I}_{N+J})^{-1}\mathbf{k}'_x, \tag{6.6}$$

where $\mathbf{K}'$ and $\mathbf{k}'_x$ are obtained by weighing the rows of $\mathbf{K}$ and $\mathbf{k}_x$ that involve $\mathbf{x}_j$ by w_j. It should be noted that by treating the desired via-points in a weighted form, the size of the kernel matrix can be reduced to $\mathbb{R}^{(N+J)\times(N+J)}$ instead of $\mathbb{R}^{N'\times N'}$, yielding faster computations.

In the case of trajectory *superposition*, the robot is expected to follow a bundle of prioritized trajectories, which we denote as $\mathbb{D}_s = \{w_h, \{\mathbf{x}_n^h, \tilde{\mathbf{y}}_n^h\}_{n=1}^N\}_{h=1}^H$ with the priorities being normalized $\sum_{h=1}^H w_h = 1$. Given the assigned priorities, we propose to construct the loss function by weighing the individual loss evaluations as:

$$\Delta(\tilde{\mathbf{y}}, \tilde{\mathbf{y}}_n) = \sum_{h=1}^{H} w_h D_f(\tilde{\mathbf{y}}_n^h, \tilde{\mathbf{y}}), \tag{6.7}$$

which results in the estimator as

$$\widehat{\mathbf{s}}(\mathbf{x}) = \underset{\tilde{\mathbf{y}}\in\mathscr{P}}{\texttt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x}) \sum_{h=1}^{H} w_h D_f(\tilde{\mathbf{y}}_n^h, \tilde{\mathbf{y}}). \tag{6.8}$$

An algorithmic framework for our proposed uniform imitation learning approach incorporating trajectory adaption is summarized in Algorithm 1.

6.2.2 Temporal Modulation

When dealing with temporal modulation, a phase variable z is introduced to decouple the dependence from time. The choice of phase $z(t)$ can be any monotonic increasing function with respect to the temporal input t. The movement now depends on the phase instead of time. Therefore, temporal modulation of the movement can be achieved by modifying the rate of the phase variable (see also Ijspeert et al. (2013); Paraschos et al. (2013)).

It should be noted that Algorithm 1 is a meta-algorithm and thus cannot be applied directly. In this section, we will illustrate how to deploy the algorithm practically. Specifically, we make the common assumption that the output distribution satisfies the Gaussian distribution: $\tilde{\mathbf{y}}_n \sim \mathscr{N}_\alpha(\mu_n, \Sigma_n)$ where $\mathscr{N}_\alpha$ denotes a Gaussian distribution in accordance with the output data (e.g. Gaussian on Euclidean $\mathscr{N}$ or manifold $\mathscr{N}_\mathscr{M}$ (Zeestraten et al., 2017b)) with mean μ_n and covariance Σ_n. As a result, the dataset can be retrieved as $\mathbb{D}_\mathscr{N} = \{\mathbf{x}_n, (\mu_n, \Sigma_n)\}_{n=1}^N$.

To instantiate the proposed meta-algorithm, the basic principle is to find the estimators $\mathbf{s}_m$ and $\mathbf{s}_c$, which predict mean μ and covariance Σ, respectively. Therefore, given a query point $\mathbf{x}$, the corresponding output is $\tilde{\mathbf{y}}(\mathbf{x}) \sim \mathcal{N}_\alpha(\mathbf{s}_m(\mathbf{x}), \mathbf{s}_c(\mathbf{x}))$. To do so, we need proper loss functions Δ_m and Δ_c. We propose to investigate the ingredients of f-divergence and decouple the losses incurred by prediction of mean and covariance:

$$D_f(\tilde{\mathbf{y}}_n, \tilde{\mathbf{y}}) = \Delta_m(\mu_n, \mu) + \Delta_c(\Sigma_n, \Sigma) + \text{const.} \tag{6.9}$$

Then following (4.12), the individual prediction can be made by solving

$$\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\mu \in \mathscr{Y}}{\texttt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Delta_m(\mu_n, \mu), \tag{6.10}$$

$$\widehat{\mathbf{s}}_c(\mathbf{x}) = \underset{\Sigma \in \mathscr{Y}^2}{\texttt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Delta_c(\Sigma_n, \Sigma). \tag{6.11}$$

By making the predictions for mean and covariance separately, we are able to transform the ill-posed problem of predicting a probability distribution into a well defined one. Moreover, by choosing different types of f-divergence, different imitation modes can be realized as shown in Chapter 7. In addition, in Chapter 8 we show how to deal with manifold-valued predictions, emphasizing the generality of our algorithmic framework.

Chapter 7

Imitation in Euclidean Space

As discussed in Chapter 6, different choices of f can result in a learner's different imitation strategies $\tilde{\mathbf{y}} \sim \mathcal{N}(\mu, \Sigma)$ for imitation of the expert policy $\tilde{\mathbf{y}}_n \sim \mathcal{N}(\mu_n, \Sigma_n)$ in response to a query input $\mathbf{x}$, which we refer to as *imitation mode*. Here we consider the case where the output of the trajectory lies in the Euclidean space. Specifically, we show how different imitation learning algorithms can be obtained by plugging in two common f-divergences, namely the KL divergence and the reverse KL divergence.

7.1 Imitation by Minimizing the KL Divergence

We start with the well-known KL-divergence to illustrate the motivation for the loss function design. The KL divergence is realized by setting f as $f(u) = u\log(u)$. Subsequently, by making use of the properties of the KL divergence between two multivariate Gaussian distributions, the corresponding f-divergence that measures the difference between the learner and the expert trajectory distri-

bution as in (6.2) is expressed by

$$
\begin{aligned}
&D_{\mathrm{KL}}(\tilde{\mathbf{y}}_n, \tilde{\mathbf{y}}) = \mathbb{E}_{\tilde{\mathbf{y}}}\left[\frac{d\tilde{\mathbf{y}}_n}{d\tilde{\mathbf{y}}} \log\left(\frac{d\tilde{\mathbf{y}}_n}{d\tilde{\mathbf{y}}}\right)\right] \\
=&\frac{1}{2}\Big(\underbrace{\underbrace{(\mu-\mu_n)^\top \Sigma^{-1}(\mu-\mu_n)}_{\Delta_m} + \log|\Sigma| + \mathtt{Tr}(\Sigma^{-1}\Sigma_n)}_{\Delta_c} \\
&- \log|\Sigma_n| - \mathtt{dim}(\mathscr{Y})\Big),
\end{aligned} \tag{7.1}
$$

where $|\cdot|$ and $\mathtt{Tr}(\cdot)$ denote the determinant and trace of a matrix, respectively, and $\mathtt{dim}(\mathscr{Y})$ indicates the dimensionality of the output space.

By grouping all the terms that include μ and Σ together, we then choose to specify the loss functions Δ_m for mean prediction and Δ_c for covariance prediction according to (7.1). After the loss functions are determined, they are then plugged into (6.10) and (6.11), respectively. Finally, the estimators with the KL-divergence imitation mode are given as:

$$
\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\mu\in\mathscr{Y}}{\mathtt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x})\big((\mu-\mu_n)^\top\Sigma^{-1}(\mu-\mu_n)\big), \tag{7.2}
$$

$$
\widehat{\mathbf{s}}_c(\mathbf{x}) = \underset{\Sigma\in\mathscr{Y}^2}{\mathtt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x})\big((\mu-\mu_n)^\top\Sigma^{-1}(\mu-\mu_n) + \log|\Sigma| + \mathtt{Tr}(\Sigma^{-1}\Sigma_n)\big). \tag{7.3}
$$

To compute the optimal mean and covariance predictions, we set to zero the derivatives of (7.2) and (7.3) with respect to μ and Σ, obtaining

$$
\mu = \frac{\sum_{n=1}^{N}\alpha_n(\mathbf{x})\mu_n}{\sum_{n=1}^{N}\alpha_n(\mathbf{x})}, \tag{7.4}
$$

$$
\Sigma = \frac{\sum_{n=1}^{N}\alpha_n(\mathbf{x})\big((\mu-\mu_n)(\mu-\mu_n)^\top + \Sigma_n\big)}{\sum_{n=1}^{N}\alpha_n(\mathbf{x})} \tag{7.5}
$$

$$
\approx \frac{\sum_{n=1}^{N}\alpha_n(\mathbf{x})\Sigma_n}{\sum_{n=1}^{N}\alpha_n(\mathbf{x})}. \tag{7.6}
$$

It should be noted that the prediction of the covariance matrix according to (7.5) is dependent on the predicted mean value given by (7.4). Therefore, we have to predict the mean first before proceeding with the covariance prediction. In addition, to alleviate the interference from the mean value on the covariance prediction, we approximate (7.5) with (7.6) by omitting $(\mu-\mu_n)(\mu-\mu_n)^\top$.

7.2 Imitation by Minimizing the Reverse KL Divergence

Besides the KL divergence mentioned above, there are also plenty of other divergences that can be used to construct the loss functions. For example, we here consider the reverse KL divergence, which reflects the asymmetry of the KL divergence. The reverse KL divergence is formally defined by $f(u)=-\log(u)$. Following the definition, the associated loss function can be obtained as

$$\begin{aligned} &D_{\texttt{RKL}}(\tilde{\mathbf{y}}_n,\tilde{\mathbf{y}}) = \mathbb{E}_{\tilde{\mathbf{y}}}\left[\log\left(\frac{d\tilde{\mathbf{y}}}{d\tilde{\mathbf{y}}_n}\right)\right] \\ =&\frac{1}{2}\big(\underbrace{(\mu-\mu_n)^\top\Sigma_n^{-1}(\mu-\mu_n)}_{\Delta_m}\underbrace{-\log|\Sigma|+\texttt{Tr}(\Sigma_n^{-1}\Sigma)}_{\Delta_c} \\ &+\log|\Sigma_n|-\texttt{dim}(\mathscr{Y})\big). \end{aligned} \tag{7.7}$$

Similarly to what we did in Section 7.1, we propose to design the loss functions Δ_m and Δ_c for mean and covariance predictions by collecting all the terms that involve μ and Σ, respectively. By substituting the loss functions given by (7.7) into (6.10) and (6.11), the reverse-KL imitation mode estimators can be expressed as

$$\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\mu\in\mathscr{Y}}{\texttt{argmin}}\sum_{n=1}^{N}\alpha_n(\mathbf{x})\big((\mu-\mu_n)^\top\Sigma_n^{-1}(\mu-\mu_n)\big), \tag{7.8}$$

$$\widehat{\mathbf{s}}_c(\mathbf{x}) = \underset{\Sigma\in\mathscr{Y}^2}{\texttt{argmin}}\sum_{n=1}^{N}\alpha_n(\mathbf{x})\big(-\log|\Sigma|+\texttt{Tr}(\Sigma_n^{-1}\Sigma)\big). \tag{7.9}$$

The optimal prediction values are thus calculated by setting the derivatives of (7.8) and (7.9) with respect to a query point $\mathbf{x}$ equal to zero, respectively. Therefore, the optimal predictions for mean and covariance are given by

$$\mu = \left(\sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Sigma_n^{-1} \right)^{-1} \sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Sigma_n^{-1} \mu_n, \tag{7.10}$$

$$\Sigma = \left(\frac{\sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Sigma_n^{-1}}{\sum_{n=1}^{N} \alpha_n(\mathbf{x})} \right)^{-1}. \tag{7.11}$$

Note that the computation of Σ must be subject to additional associated constraints. In principle, to be a valid covariance matrix, the square matrix Σ needs to be symmetric, i.e. $\Sigma = \Sigma^\top$, and positive semi-definite, which is denoted by $\Sigma \succeq 0$. Although we did not specify the constraints explicitly, it can be observed that the solutions in (7.5) and (7.11) are indeed symmetric positive semi-definite matrices. This can be verified by recalling that (i) the outer product of a vector, (ii) the inverse of a symmetric positive semi-definite matrix, and (iii) the sum of the symmetric positive semi-definite matrices are all symmetric positive semi-definite (PSD) matrices. Nevertheless, for other possible choices of f, the constraint for Σ being a symmetric positive semi-definite matrix should be taken into account explicitly when solving the optimization problem in (6.11).

Lastly, in order to ensure that the employed estimators are valid, the loss functions Δ_m and Δ_c used by (6.10) and (6.11) are required to be SELF, as defined by (4.3). The corresponding proof is provided in Appendix 13.

Our main results are summarized in Table 7.1, while the algorithm is reported in Algorithm 2.

Algorithm 2: Imitation with Euclidean-Valued Output

Collect trajectories from multiple demonstrations;
`/* Assumption of Gaussian output */`
Process raw data for $\mathbb{D}_{\mathscr{N}} = \{\mathbf{x}_n, (\mu_n, \Sigma_n)\}_{n=1}^{N}$;
Define the kernel k and parameter λ;
Choose imitation mode f;
switch *imitation mode* f **do**
  for $t = 1, \ldots, T$ **do**
    Input: a query point $\mathbf{x}_t$;
    Calculate the weights $\alpha(\mathbf{x}_t)$;
    case $f(u) = u\log(u)$ **do**
      `/* Predict with the KL divergence mode */`
      Output: mean μ as per (7.4);
      Output: covariance Σ as per (7.5);
    case $f(u) = -\log(u)$ **do**
      `/* Predict with the reverse KL divergence mode */`
      Output: mean μ as per (7.10);
      Output: covariance Σ as per (7.11);
    otherwise do
      Formulate Δ_m and Δ_c motivated by (6.9);
      Output: mean μ by optimizing (6.10);
      Output: covariance Σ by optimizing (6.11);
      // s.t. $\Sigma = \Sigma^\top$ and $\Sigma \succeq 0$

Table 7.1 List of imitation modes based on different divergences

	Kullback-Leibler divergence	Reverse Kullback-Leibler divergence
$f(u)$	$u\log(u)$	$-\log(u)$
$\Delta_m(\mu_n,\mu)$	$(\mu-\mu_n)^\top\Sigma^{-1}(\mu-\mu_n)$	$(\mu-\mu_n)^\top\Sigma_n^{-1}(\mu-\mu_n)$
$\widehat{\mathbf{s}}_m(\mathbf{x})$	$\dfrac{\sum_{n=1}^N \alpha_n(\mathbf{x})\mu_n}{\sum_{n=1}^N \alpha_n(\mathbf{x})}$	$\left(\sum_{n=1}^N \alpha_n(\mathbf{x})\Sigma_n^{-1}\right)^{-1}\sum_{n=1}^N \alpha_n(\mathbf{x})\Sigma_n^{-1}\mu_n$
$\Delta_c(\Sigma_n,\Sigma)$	$(\mu-\mu_n)^\top\Sigma^{-1}(\mu-\mu_n)+\log\lvert\Sigma\rvert+\mathrm{Tr}(\Sigma^{-1}\Sigma_n)$	$-\log\lvert\Sigma\rvert+\mathrm{Tr}(\Sigma_n^{-1}\Sigma)$
$\widehat{\mathbf{s}}_c(\mathbf{x})$	$\dfrac{\sum_{n=1}^N \alpha_n(\mathbf{x})\left((\mu-\mu_n)(\mu-\mu_n)^\top+\Sigma_n\right)}{\sum_{n=1}^N \alpha_n(\mathbf{x})}$	$\left(\dfrac{\sum_{n=1}^N \alpha_n(\mathbf{x})\Sigma_n^{-1}}{\sum_{n=1}^N \alpha_n(\mathbf{x})}\right)^{-1}$

7.3 Temporal Trajectory Adaptation

Sometimes, learning the velocity profile of temporal trajectories may be required, in particular when both position and velocity play an important role in successful skills transfer. Yet, a direct application of the structured prediction method developed so far is infeasible, since simple weighted sum of all the mean of the output values will break the constraint of the derivative relationship between position and velocity outputs. Therefore, we need to treat the learning of temporal trajectories explicitly with the output value composed of position μ and velocity $\dot{\mu}$:

$$\mathbf{y} = \begin{bmatrix} \mu \\ \dot{\mu} \end{bmatrix}. \tag{7.12}$$

Recall that we used the least-squares estimator (4.5) to solve the surrogate problem. As this treatment fails to take the implicit temporal constraint of output value into account, a more expressive estimator for $\mathbf{g}^*$ is needed such that it can capture the temporal relationship. We motivate our design for the approximation value $\widehat{\mathbf{g}}$ by referring to analytic insight of the feature map. We choose the KL-divergence imitation mode as an example due to its relatively simple expression. Therefore, according to (13.1), the approximated value $\widehat{\mathbf{g}}$ is suggested as:

$$\widehat{\mathbf{g}}(t) = \begin{bmatrix} \widehat{\mathbf{y}}_t^\top \Sigma^{-1} \widehat{\mathbf{y}}_t \\ \widehat{\mathbf{y}}_t \\ 1 \end{bmatrix}, \tag{7.13}$$

where $\widehat{\mathbf{y}}_t$ is required to satisfy the temporal constraint and parameterized by

$$\widehat{\mathbf{y}}_t = \begin{bmatrix} \widehat{\mathbf{W}}^\top \phi_t \\ \widehat{\mathbf{W}}^\top \dot{\phi}_t, \end{bmatrix}, \tag{7.14}$$

with the weight matrix $\widehat{\mathbf{W}}$ as well as basis function ϕ and its time derivative

$$\dot{\phi}(t) = \lim_{\delta \to 0} \frac{\phi(t+\delta) - \phi(t-\delta)}{2\delta}. \tag{7.15}$$

In order to find the optimal $\mathbf{W}$, we solve the following optimization problem

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\texttt{argmin}} \frac{1}{N} \sum_{n=1}^{N} \left\| \begin{bmatrix} \phi_n^\top \\ \dot{\phi}_n^\top \end{bmatrix} \mathbf{W} - \begin{bmatrix} \mu_n^\top \\ \dot{\mu}_n^\top \end{bmatrix} \right\|_F^2 + \lambda \|\mathbf{W}\|_F^2. \tag{7.16}$$

By setting the derivative of (7.25) w.r.t. $\mathbf{W}$ to zero, we then obtain

$$\widehat{\mathbf{W}} = \Phi(\Phi^\top \Phi + N\lambda \mathbf{I})^{-1} \mathbf{Y}, \tag{7.17}$$

where

$$\begin{aligned} \Phi &= [\phi_1, \dot{\phi}_1, \ldots, \phi_N, \dot{\phi}_N], \\ \mathbf{Y} &= [\mu_1, \dot{\mu}_1, \ldots, \mu_N, \dot{\mu}_N]^\top. \end{aligned} \tag{7.18}$$

Consequently, given a query time step t, we have

$$\widehat{\mathbf{y}}_t = \begin{bmatrix} \mathbf{Y}^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}^p \\ \mathbf{Y}^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}^v \end{bmatrix}, \tag{7.19}$$

where the vectors $\mathbf{k}^p$ and $\mathbf{k}^v$ are defined as

$$\mathbf{k}_i^p = \begin{cases} \phi_i^\top \phi_t \\ \dot{\phi}_i^\top \phi_t \end{cases} \quad \text{and} \quad \mathbf{k}_i^v = \begin{cases} \phi_i^\top \dot{\phi}_t & i = 2n-1, \\ \dot{\phi}_i^\top \dot{\phi}_t & i = 2n. \end{cases} \tag{7.20}$$

With slight abuse of notation, the kernel matrix $\mathbf{K}$ is constructed as

$$\mathbf{K}_{i,j} = \begin{cases} \phi_i^\top \phi_j & i = 2n-1, \quad j = 2n'-1, \\ \phi_i^\top \dot{\phi}_j & i = 2n-1, \quad j = 2n', \\ \dot{\phi}_i^\top \phi_j & i = 2n, \quad j = 2n'-1, \\ \dot{\phi}_i^\top \dot{\phi}_j & i = 2n, \quad j = 2n'. \end{cases} \tag{7.21}$$

Where n and $n' = 1, 2, 3, \ldots, N$. By substituting $\dot{\phi}$ with its definition (7.26) into (7.21), we then apply the kernel trick for the matrix as shown in Table 7.2, where $t^\pm \triangleq t \pm \delta$. Also kernelized version for (7.20) can be obtained similarly.

Table 7.2 Definition for kernel matrix $\mathbf{K}$.

$\mathbf{K}_{i,j}$	$i = 2n-1$	$i = 2n$
$j = 2n'-1$	$k(t_i,t_j)$	$(k(t_i^+,t_j) - k(t_i^-,t_j))/2\delta$
$j = 2n'$	$(k(t_i,t_j^+) -$ $k(t_i,t_j^-))/2\delta$	$(k(t_i^+,t_j^+) - k(t_i^+,t_j^-) -$ $k(t_i^-,t_j^+) + k(t_i^-,t_j^-))/4\delta^2$

By substituting (7.13) into (4.10), we obtain the optimization problem as

$$\widehat{\mathbf{s}}(t) = \underset{\mathbf{y}\in\mathscr{Y}}{\texttt{argmin}}\, \mathbf{y}^\top \Sigma^{-1}\mathbf{y} - 2\mathbf{y}^\top\Sigma^{-1}\widehat{\mathbf{y}}_t + \widehat{\mathbf{y}}_t^\top\Sigma^{-1}\widehat{\mathbf{y}}_t. \tag{7.22}$$

In view of (7.22), separating α-like scalar weights out of feature map $\Psi(\mathbf{y}_n)$ is not as straightforward as the least square surrogate framework. Therefore, we consider to solve for (7.22) directly by setting its derivative to zero. As a result, the optimal prediction at time step t is calculated as (7.30), which is a self-contained solution as it exactly recovers our chosen proxy for $\mathbf{g}^*(t)$.

We here re-write the solution to our estimator $\widehat{\mathbf{s}}(t)$ using the formalism of a weighted sum of output values:

$$\widehat{\mathbf{s}}(t) = \sum_{n=1}^{N} \alpha_n \mathbf{y}_n. \tag{7.23}$$

The weight matrix α_n is defined as

$$\alpha_n = \begin{bmatrix} \alpha_{2n-1}^p \mathbf{I} & \alpha_{2n}^p \mathbf{I} \\ \alpha_{2n-1}^v \mathbf{I} & \alpha_{2n}^v \mathbf{I} \end{bmatrix}, \tag{7.24}$$

where $\alpha^p = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}^p$ and $\alpha^v = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}^v$. For the temporal trajectory going through some desired via-point and/or via-velocity, $\mathbf{K}$, $\mathbf{k}^p$, and $\mathbf{k}^v$ need to be modified similarly to (6.6), i.e. the corresponding rows that involve desired adaptive behavior need to be weighed, respectively.

7.4 Jerk Minimization

Sometimes it is desired to generate a jerk-minimization trajectory. To this end, we can consider to modify (7.25) by incorporating another line of learning acceleration profile

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\texttt{argmin}} \frac{1}{N} \sum_{n=1}^{N} \left\| \begin{bmatrix} \phi_n^\top \\ \dot{\phi}_n^\top \\ \ddot{\phi}_n^\top \end{bmatrix} \mathbf{W} - \begin{bmatrix} \mu_n^\top \\ \dot{\mu}_n^\top \\ \mathbf{0}^\top \end{bmatrix} \right\|_F^2 + \lambda \|\mathbf{W}\|_F^2, \tag{7.25}$$

where we set the targeted acceleration value as zero in order to minimize the jerk. And $\ddot{\phi}$ is defined as

$$\ddot{\phi}(t) = \lim_{\delta \to 0} \frac{\dot{\phi}(t+\delta) - \dot{\phi}(t-\delta)}{2\delta}. \tag{7.26}$$

Similarly, the desired value to $\widehat{\mathbf{W}}$ also share the same solution form as (7.17), except that Φ and $\mathbf{Y}$ need to be redefined as

$$\begin{aligned} \Phi &= [\phi_1, \dot{\phi}_1, \ddot{\phi}_1, \ldots, \phi_N, \dot{\phi}_N, \ddot{\phi}_N], \\ \mathbf{Y} &= [\mu_1, \dot{\mu}_1, \mathbf{0}, \ldots, \mu_N, \dot{\mu}_N, \mathbf{0}]^\top. \end{aligned} \tag{7.27}$$

Following (7.20), we can define

$$\mathbf{k}_i^p = \begin{cases} \phi_i^\top \phi_t \\ \dot{\phi}_i^\top \phi_t \\ \ddot{\phi}_i^\top \phi_t \end{cases} \quad \mathbf{k}_i^v = \begin{cases} \phi_i^\top \dot{\phi}_t \\ \dot{\phi}_i^\top \dot{\phi}_t \\ \ddot{\phi}_i^\top \dot{\phi}_t \end{cases} \quad \text{and} \quad \mathbf{k}_i^a = \begin{cases} \phi_i^\top \ddot{\phi}_t & i = 3n-2 \\ \dot{\phi}_i^\top \ddot{\phi}_t & i = 3n-1 \\ \ddot{\phi}_i^\top \ddot{\phi}_t & i = 3n \end{cases}. \tag{7.28}$$

And the kernel matrix shall be modified as

$$\mathbf{K}_{i,j} = \begin{cases} \boldsymbol{\phi}_i^\top \boldsymbol{\phi}_j & i = 3n-2, \quad j = 3n'-2, \\ \boldsymbol{\phi}_i^\top \dot{\boldsymbol{\phi}}_j & i = 3n-2, \quad j = 3n'-1, \\ \boldsymbol{\phi}_i^\top \ddot{\boldsymbol{\phi}}_j & i = 3n-2, \quad j = 3n', \\ \dot{\boldsymbol{\phi}}_i^\top \boldsymbol{\phi}_j & i = 3n-1, \quad j = 3n'-2, \\ \dot{\boldsymbol{\phi}}_i^\top \dot{\boldsymbol{\phi}}_j & i = 3n-1, \quad j = 3n'-1, \\ \dot{\boldsymbol{\phi}}_i^\top \ddot{\boldsymbol{\phi}}_j & i = 3n-1, \quad j = 3n', \\ \ddot{\boldsymbol{\phi}}_i^\top \boldsymbol{\phi}_j & i = 3n, \quad j = 3n'-2, \\ \ddot{\boldsymbol{\phi}}_i^\top \dot{\boldsymbol{\phi}}_j & i = 3n, \quad j = 3n'-1, \\ \ddot{\boldsymbol{\phi}}_i^\top \ddot{\boldsymbol{\phi}}_j & i = 3n, \quad j = 3n'. \end{cases} \tag{7.29}$$

Hence, given a query point t, we have the predicted output as

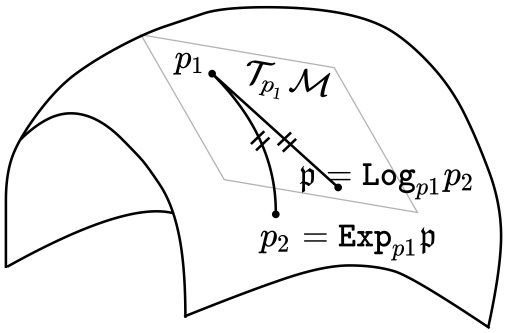
$$\widehat{\mathbf{y}}_t = \begin{bmatrix} \mathbf{Y}^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}^p \\ \mathbf{Y}^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}^v \end{bmatrix}. \tag{7.30}$$

Chapter 8

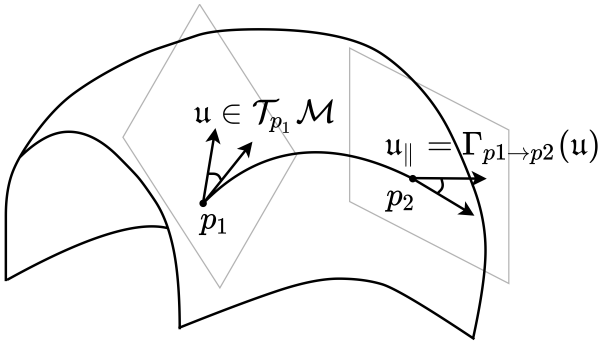
Imitation with Manifold Constraint

In this section, we show the generality of our proposed algorithmic framework in the context of manifold structured prediction, i.e. when the output space $\mathscr{Y}$ is a manifold $\mathscr{M}$. Differently from the previous chapter, the learner's and the expert's trajectories are now constrained to lie on Riemannian manifold. Here we first provide basic notions and corresponding notations on Riemannian statistics that are used throughout this paper. We refer interested readers to Absil et al. (2009) for more details on manifolds. Informally, a d-dimensional Riemannian manifold $(\mathscr{M}, g)$ is a topological space that locally behaves like the Euclidean space $\mathbb{R}^d$. Every point $\mathbf{p} \in \mathscr{M}$ has a tangent space $\mathscr{T}_{\mathbf{p}}\mathscr{M}$[1] where an inner product g is defined. When $\mathscr{M}$ is a *submanifold* of $\mathbb{R}^{d+1}$, the inner product can inherit from the standard Euclidean inner product in a natural way. The minimum distance between two points $\mathbf{p}_1$ and $\mathbf{p}_2$ on a Riemannian manifold is called *geodesic distance* $\texttt{dist}(\mathbf{p}_1, \mathbf{p}_2)$, which generalizes the concept of straight lines in Euclidean spaces. The *exponential map* $\texttt{Exp}_{\mathbf{p}} : \mathscr{T}_{\mathbf{p}}\mathscr{M} \to \mathscr{M}$ maps a point in the tangent space to the manifold with the distance and direction preserved. A *retraction* map $R_{\mathbf{p}} : \mathscr{T}_{\mathbf{p}}\mathscr{M} \to \mathscr{M}$ is a first order approximation of the exponential map. The inverse mapping of the exponential map is called the *logarithm map* $\texttt{Log}_{\mathbf{p}} : \mathscr{M} \to \mathscr{T}_{\mathbf{p}}\mathscr{M}$. The logarithmic map is defined except for the *cut locus* of

[1] Elements on the tangent space are in *fraktur* typeface, e.g. $\mathfrak{p} \in \mathscr{T}_{\mathbf{p}}\mathscr{M}$



(a) Manifold exponential and logarithmic mappings.



(b) Parallel transport of vector.

Figure 8.1 Illustration of basic Riemannian notions.

the base, which is the set of points that are connected by more than one geodesic curve with the base. *Parallel transport* $\Gamma_{\mathbf{p}_1 \to \mathbf{p}_2}(\mathfrak{u}) : \mathscr{T}_{\mathbf{p}_1}\mathscr{M} \to \mathscr{T}_{\mathbf{p}_2}\mathscr{M}$ moves vectors between tangent spaces such that the angles between the vectors and the geodesic curve connecting the bases are conserved. This operation is necessary to transport information available in one tangent space to another. The *Cartesian product* of two Riemannian manifolds $\mathscr{M}_1 \times \mathscr{M}_2$ is still a Riemannian manifold and the corresponding manifold operations mentioned above can be obtained by concatenating the individual operations. A pictorial illustration is shown in Figure 8.1.

When the Riemannian Gaussian distribution $\mathscr{N}_{\mathscr{M}}$ is used to represent the policy, it is usually approximated as (see, Simo-Serra et al. (2017); Zeestraten

et al. (2017b)):

$$\mathscr{N}_{\mathscr{M}}(\mathbf{y};\mu,\Sigma)=\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}e^{-\frac{1}{2}\mathtt{Log}_{\mu}(\mathbf{y})^{\top}\Sigma^{-1}\mathtt{Log}_{\mu}(\mathbf{y})}, \tag{8.1}$$

where $\mu\in\mathscr{M}$ is the Riemannian mean, and Σ the covariance matrix defined in the tangent space $\mathscr{T}_{\mu}\mathscr{M}$. To construct the corresponding loss function, we also propose to compute the f-divergence between the learner policy $\tilde{\mathbf{y}}\sim\mathscr{N}_{\mathscr{M}}(\mu,\Sigma)$ and the expert policy $\tilde{\mathbf{y}}_n\sim\mathscr{N}_{\mathscr{M}}(\mu_n,\Sigma_n)$. Here, we take the KL divergence as an example. The loss function is therefore constructed as

$$\begin{aligned}D_{\mathtt{KL}}(\tilde{\mathbf{y}}_n,\tilde{\mathbf{y}})=\frac{1}{2}\int\Bigg(&\log\frac{|\Sigma|}{|\Sigma_n|}-\mathtt{Log}_{\mu_n}(\mathbf{y})\Sigma_n^{-1}\mathtt{Log}_{\mu_n}(\mathbf{y})\\&+\mathtt{Log}_{\mu}(\mathbf{y})\Sigma^{-1}\mathtt{Log}_{\mu}(\mathbf{y})\Bigg)\tilde{\mathbf{y}}_n d\mathbf{y}.\end{aligned} \tag{8.2}$$

In order to facilitate the computation of (8.2), we make use of the approximation

$$\mathtt{Log}_{\mu}(\mathbf{y})\approx\mathtt{Log}_{\mu}(\mu_n)+\mathtt{Log}_{\mu_n}(\mathbf{y}). \tag{8.3}$$

As a result, (8.2) can be approximated as

$$\begin{aligned}&\frac{1}{2}\Big(\underbrace{\underbrace{\mathtt{Log}_{\mu}(\mu_n)^{\top}\Sigma^{-1}\mathtt{Log}_{\mu}(\mu_n)}_{\Delta_m}+\log|\Sigma|+\mathtt{Tr}(\Sigma^{-1}\Sigma_n)}_{\Delta_c}\\&-\log|\Sigma_n|+\mathtt{dim}(\mathscr{Y})\Big).\end{aligned} \tag{8.4}$$

In view of the complex form of Δ_m, we propose to ease the computation by omitting the weight Σ upon observing the fact that it plays no role in the Euclidean case (7.4). Therefore, the loss function for mean prediction Δ_m now becomes

$$\Delta_m=\mathtt{Log}_{\mu}(\mu_n)^{\top}\mathtt{Log}_{\mu}(\mu_n)=\mathtt{dist}^2(\mu_n,\mu), \tag{8.5}$$

Algorithm 3: Imitation with Manifold Output

Collect multiple Riemannian trajectories;
Process raw data for $\mathbb{D}_{\mathscr{M}} = \{\mathbf{x}_n, (\mu_n, \Sigma_n)\}_{n=1}^{N}$;
Eigen-decomposition of covariance as per (8.11);
Initialize kernel k, regularization λ, and step size η;
for $t = 1, \ldots, T$ **do**
    Input: a query point $\mathbf{x}_t$;
    Calculate the weights $\alpha(\mathbf{x}_t)$;
    repeat
        $\mathfrak{v} = \nabla_{\mathscr{M}} \sum_{n=1}^{N} \alpha_n(\mathbf{x}_t)\, \mathtt{dist}^2(\mu_n, \mu)$;
        $\mu \leftarrow R_\mu(\eta\, \mathfrak{v})$;
    until *convergence*;
    Output: mean μ;
    Compute the parallel transport covariance $\Sigma_{\|n}$ as per (8.10);
    Output: covariance Σ as per (8.8);

where $\mathtt{dist}(\cdot,\cdot)$ denotes the geodesic distance between two manifold points. Finally, our estimator for mean prediction is obtained as

$$\widehat{\mathbf{s}}_m(\mathbf{x}) = \underset{\mu \in \mathscr{Y}}{\mathtt{argmin}} \sum_{n=1}^{N} \alpha_n(\mathbf{x})\, \mathtt{dist}^2(\mu_n, \mu). \tag{8.6}$$

It shall be noted that the loss function used in (8.6) is the squared geodesic distance, which is SELF as shown by Rudi et al. (2018). To perform the estimation at a new test point $\mathbf{x}$, a geometric optimization algorithm is required. In particular, we employ the Riemannian gradient descent, which extends the usual gradient descent method to manifolds with the guarantee that the computed value is still an element of the manifold. By denoting the minimization objective of (8.6) as $\mathtt{F}(\mu)$, the iterative optimization process takes the form

$$\mu_{i+1} = \mathtt{Exp}_{\mu_i}(\eta_i \nabla_{\mathscr{M}} \mathtt{F}(\mu_i)), \tag{8.7}$$

where $\nabla_{\mathscr{M}}$ is the Riemannian gradient operator and $\eta_i \in \mathbb{R}$ is a step size. In general, the exponential map $\mathtt{Exp}$ could be difficult to compute. A computationally cheaper alternative is the retraction map $R_\mu : \mathscr{T}_\mu \mathscr{M} \to \mathscr{M}$, which is a first-order

approximation to the exponential map. In addition to faster computation, the retraction map also preserves convergence guarantees.

The procedure for the covariance matrix prediction can be achieved similarly to (7.5) and (7.6):

$$\Sigma = \frac{\sum_{n=1}^{N} \alpha_n(\mathbf{x}) \left(\mathtt{Log}_{\mu}(\mu_n) \mathtt{Log}_{\mu}(\mu_n)^{\top} + \Sigma_{\|n} \right)}{\sum_{n=1}^{N} \alpha_n(\mathbf{x})}, \tag{8.8}$$

$$\approx \frac{\sum_{n=1}^{N} \alpha_n(\mathbf{x}) \Sigma_{\|n}}{\sum_{n=1}^{N} \alpha_n(\mathbf{x})}, \tag{8.9}$$

where a parallel transported covariance matrix $\Sigma_{\|n}$ is used in place of Σ_n. Parallel transport is necessary here as it can transfer the information from one point to another by considering the rotation of the coordinate systems along the geodesic curve. The transported covariance matrix can be computed by

$$\Sigma_{\|n} = \sum_{j=1}^{d} \Gamma_{\mu_n \to \mu}(\mathfrak{u}_j) \Gamma_{\mu_n \to \mu}(\mathfrak{u}_j)^{\top}, \tag{8.10}$$

where $\mathfrak{u}_j$ is obtained through an eigen-decomposition:

$$\Sigma_n = \sum_{j=1}^{d} \mathfrak{u}_j \mathfrak{u}_j^{\top}. \tag{8.11}$$

Likewise, manifold-adaptive behavior, such as passing through some desired via-point, is realized by modifying the weight α according to (6.6).

Table 8.1 provides the manifold operations required for two problems of interest considered in this thesis: The sphere $\mathbb{S}^2(r)$ with radius r and the circular generalized cylinder $\mathbb{R}^2 \times \mathbb{S}$. An algorithm for imitation with manifold-valued output is summarized in Algorithm 3.

Table 8.1 List of operations of common Riemannian manifolds considered in this thesis.

	Sphere with radius r: $\mathbb{S}^2(r)$	Circular generalized cylinder: $\mathbb{R}^2 \times \mathbb{S}^1$
Distance metric $\texttt{dist}(\mu_n, \mu)$	$r \arccos\left(\frac{\mu_n^\top \mu}{r^2}\right)$	$\sqrt{\left\|\mu_n^{\mathbb{R}} - \mu^{\mathbb{R}}\right\|^2 + \arccos\left(\mu_n^{\mathscr{S}\top} \mu^{\mathscr{S}}\right)^2}$
Minimization objective $\texttt{F}(\mu)$	$\sum_{n=1}^N \alpha_n r^2 \arccos\left(\frac{\mu_n^\top \mu}{r^2}\right)^2$	$\sum_{n=1}^N \alpha_n \left(\left\|\mu_n^{\mathbb{R}} - \mu^{\mathbb{R}}\right\|^2 + \arccos\left(\mu_n^{\mathscr{S}\top} \mu^{\mathscr{S}}\right)^2\right)$
Riemannian gradient $\nabla_{\mathscr{M}} \texttt{F}(\mu)$	$2\sum_{n=1}^N \alpha_n \left(\frac{\mu\mu^\top}{r^2} - \mathbf{I}\right) \frac{\arccos\left(\frac{\mu_n^\top \mu}{r^2}\right)}{\sqrt{1 - \left(\frac{\mu_n^\top \mu}{r^2}\right)^2}} \mu_n$	$2\sum_{n=1}^N \alpha_n \left[\left[\mu^{\mathbb{R}} - \mu_n^{\mathbb{R}}\right]^\top \frac{\arccos\left(\mu_n^{\mathscr{S}\top} \mu^{\mathscr{S}}\right)}{\sqrt{1 - \left(\mu_n^{\mathscr{S}\top} \mu^{\mathscr{S}}\right)^2}} \times \left[(\mu^{\mathscr{S}} \mu^{\mathscr{S}\top} - \mathbf{I}) \mu_n^{\mathscr{S}}\right]^\top\right]^\top$
Retraction $R_\mu(\mathfrak{p})$	$r \frac{\mu + \mathfrak{p}}{\|\mu + \mathfrak{p}\|}$	$\left[\left[\mu^{\mathbb{R}} + \mathfrak{p}^{\mathbb{R}}\right]^\top \left[\frac{\mu^{\mathscr{S}} + \mathfrak{p}^{\mathscr{S}}}{\|\mu^{\mathscr{S}} + \mathfrak{p}^{\mathscr{S}}\|}\right]^\top\right]^\top$
Logarithmic map $\texttt{Log}_{\mu_n}(\mu)$	$\texttt{dist}(\mu_n, \mu) \frac{r^2 \mu - \mu_n^\top \mu \mu_n}{\|r^2 \mu - \mu_n^\top \mu \mu_n\|}$	$\left[\left[\mu^{\mathbb{R}} - \mu_n^{\mathbb{R}}\right]^\top \texttt{dist}(\mu_n^{\mathscr{S}}, \mu^{\mathscr{S}}) \frac{\mu^{\mathscr{S}\top} - \mu_n^\top \mu \mu_n^{\mathscr{S}\top}}{\|\mu^{\mathscr{S}} - \mu_n^\top \mu \mu_n^{\mathscr{S}}\|}\right]^\top$
Parallel transport $\Gamma_{\mu_n \to \mu}(\mathfrak{u})$	$\mathfrak{u} - \frac{\texttt{Log}_{\mu_n}(\mu)^\top \mathfrak{u}}{\texttt{dist}^2(\mu_n, \mu)} \left(\texttt{Log}_{\mu_n} \mu + \texttt{Log}_\mu \mu_n\right)$	$\left[\mathfrak{u}^{\mathbb{R}\top} \mathfrak{u}^{\mathscr{S}\top} - \frac{\texttt{Log}_{\mu_n^{\mathscr{S}}}(\mu^{\mathscr{S}})^\top \mathfrak{u}^{\mathscr{S}}}{\texttt{dist}^2(\mu_n^{\mathscr{S}}, \mu^{\mathscr{S}})} \left[\texttt{Log}_{\mu_n^{\mathscr{S}}} \mu^{\mathscr{S}} + \texttt{Log}_{\mu^{\mathscr{S}}} \mu_n^{\mathscr{S}}\right]^\top\right]^\top$

Chapter 9

Evaluations

We evaluate the effectiveness of our proposed approach with both simulations and real experiments in this chapter. Our experimental set-ups manifest several advantages of our approach by making comparisons with state-of-the-art methods, namely ProMP, KMP, and LPV-DS[1]. Specifically, we conduct simulation experiments on trajectory reproduction as well as trajectory adaptation including both via-position and velocity points.

9.1 Simulation Results

In this part, simulation results are provided. The effectiveness of our approach to imitation in Euclidean space is shown by learning trajectories of different 2D letters. For imitation of trajectories with manifold constraints, two cases, namely a sphere and a generalized cylinder, are studied. For other types of manifolds, our approach can be applied similarly.

9.1.1 Trajectory Reproduction

We first evaluate the effectiveness of our algorithm in a trajectory reproduction scenario under probabilistic imitation learning framework, i.e. learning a policy

[1]Source codes used for comparison are from Calinon (2020), Huang et al. (2019), and Figueroa and Billard (2018)

from multiple demonstrations. Our goal is to compare with state-of-the-art algorithms in terms of imitation fidelity by learning the trajectories of four illustrative letters, namely 'N', 'S', 'W', and 'P', with time as the input. It should be noted that trajectory covariance learning is usually not considered by autonomous approaches like LPV-DS as they focus more on global stability and robustness to disturbances.

The collected data from multiple demonstrations for the letter trajectories is first fit with Gaussian mixture models (GMM), subsequently a probabilistic reference trajectory is extracted by Gaussian mixture regression (GMR) (Billard et al., 2008), serving as the baseline for different imitation algorithms to compare against.

The learning results are shown in Figure 9.1. In the same row, the same multiple demonstrations for a letter are fed to different imitation learning algorithms in order to learn mean and covariance of a probabilistic trajectory. The first column in yellow represents ProMP. As ProMP is based on a parametric method, a set of basis functions are needed for fitting demonstration trajectories. We employ 30 Gaussian radial basis functions (RBF) for learning reference trajectories. The second column in blue represents KMP, whose hyperparameters, such as the regularization coefficients, are selected according to Huang et al. (2019). The third column in red denotes LPV-DS. It should be noted that in order for LPV-DS to generate a trajectory, an initial point needs to be manually selected. Here, we set it to coincide with the reference trajectory's initial point. The last two columns in green represent our approach. Wherein, the fourth column denotes the KL divergence imitation mode and the fifth column denotes the reverse KL divergence imitation mode. To apply our algorithm, we choose the Gaussian kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\kappa \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2)$ with hyper-parameter $\kappa = 6$ when training the input-dependent weights α.

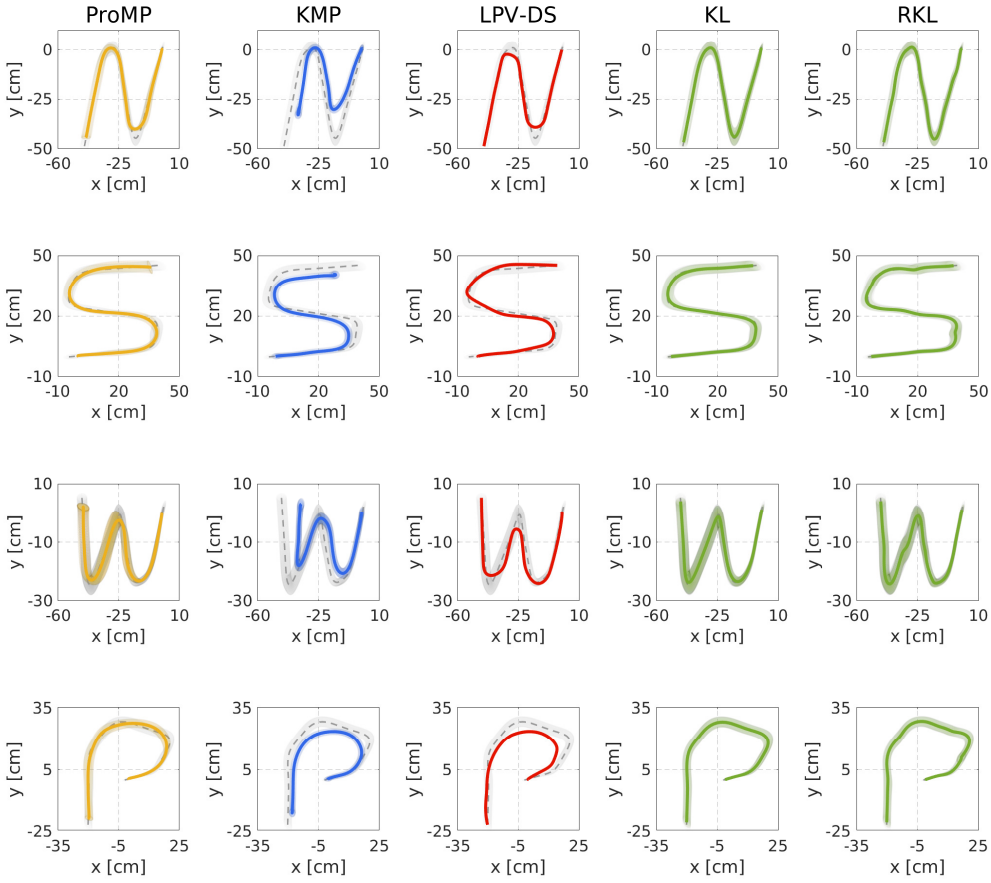


Figure 9.1 Imitation fidelity comparison among ProMP (yellow), KMP (blue), LPV-DS (red) and our approach (green, with the fourth column KL imitation mode and the fifth column reverse KL imitation mode) for reproducing the trajectories of different letters. Solid curve and shallow area are used to represent trajectory mean and standard deviation, respectively. The reference baseline used here is the probabilistic trajectory retrieved by GMR (gray).

Table 9.1 Performance comparison of different imitation learning algorithms

	'N'			'S'			'W'			'P'		
	C_m	C_{cov}	Time (s)	C_m	C_{cov}	Time (s)	C_m	C_{cov}	Time (s)	C_m	C_{cov}	Time (s)
ProMP	17.5	16.5	0.01	15.8	19.4	0.02	14.2	17.7	0.02	14.6	14.1	0.01
KMP	40.3	19.3	1.84	30.6	17.5	1.83	35.2	18.4	1.88	29.2	18.8	2.10
LPV-DS	60.8	N/A	12.9	65.3	N/A	11.1	61.0	N/A	13.5	56.9	N/A	9.5
KL	7.4	5.7	0.11	8.9	6.5	0.09	7.8	6.0	0.11	5.9	5.4	0.09
RKL	7.3	6.4	1.47	9.3	5.7	1.51	8.9	6.2	1.55	5.0	5.1	1.52

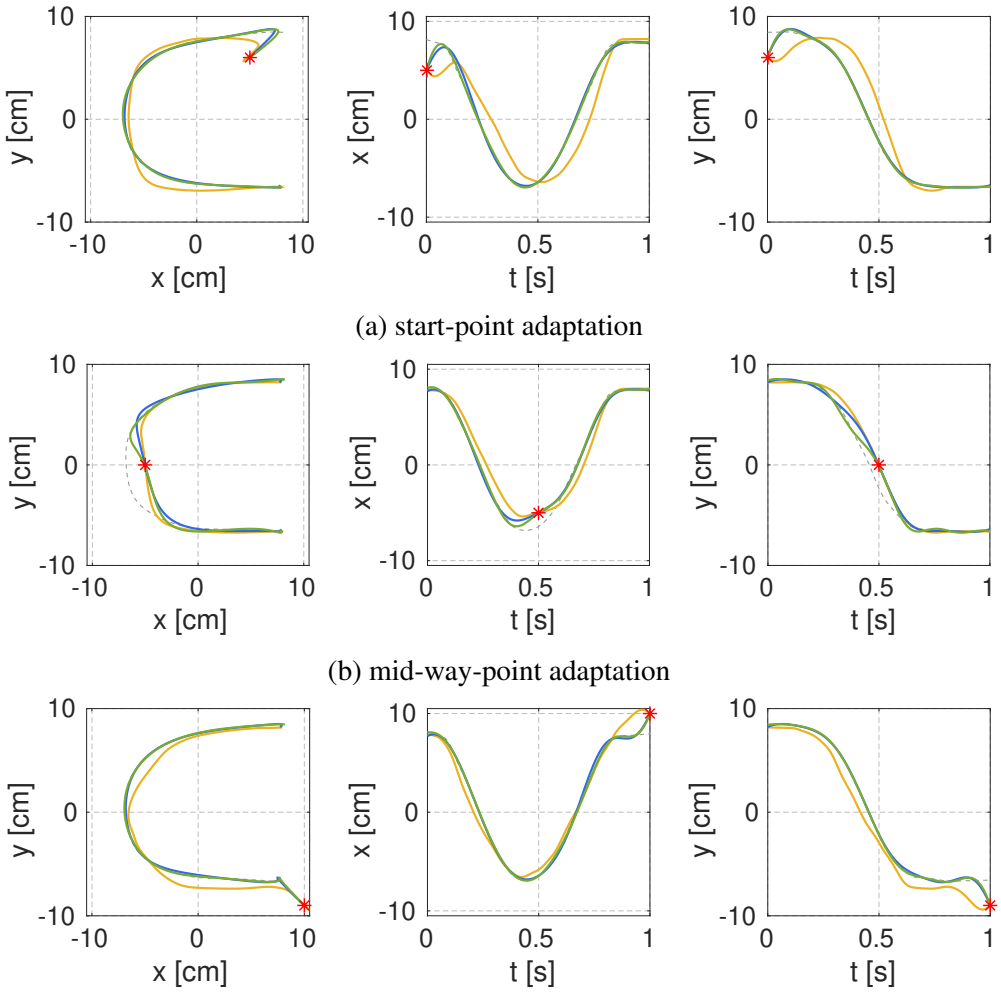
In general, all the imitation learning algorithms can accurately reproduce the original demonstrated trajectories. Both learned trajectory mean and covariance coincide with the reference probabilistic trajectory to some extent. In Figure 9.1, it can be observed that our proposed methods most closely reproduce the original demonstrations. Especially, when it comes to abrupt turns, as in the case of the letter 'W', our algorithm can still imitate the demonstrated trajectory closely, while other methods exhibit a larger mismatch. In order to quantitatively compare the reproduction performances of each algorithm, we report the cumulative error for trajectory mean and covariance, as well as the training time. For evaluating the trajectory mean imitation, we propose to compute the Root Mean Square Error (RMSE) between the estimated value and the demonstrated one:

$$C_m = \sqrt{\sum_{n=1}^{N} \| \widehat{\mathbf{s}}_m(\mathbf{x}_n) - \mu_n \|^2}$$

. Likewise, for the evaluation of trajectory covariance imitation, we define the error as

$$C_{cov} = \sqrt{\sum_{n=1}^{N} \| \log(\Sigma_n^{-\frac{1}{2}} \widehat{\mathbf{s}}_c(\mathbf{x}_n) \Sigma_n^{-\frac{1}{2}}) \|_{\mathrm{F}}^2}$$

, where the notation of geodesic distance of positive definite matrices is utilized. Concerning computational complexity, ProMP is expected to be more efficient since it grows as $\mathcal{O}(m^3 d^3)$, where m is the dimension of the basis function and d is the dimension of output value. Our algorithm and KMP both have a $\mathcal{O}(n^3)$ time complexity, where n denotes the number of trajectory data points. LPV-DS is expected to be the most computationally expensive, since a non-linear constrained optimization problem has to be solved. The obtained numerical results are summarized in Table 9.1, where training time is averaged over five trials. To conclude, our algorithm shows significant performance improvements with respect to the selected state-of-the-art methods on a diverse set of target trajectories, in terms of both trajectory mean and covariance imitation quality.



(a) start-point adaptation

(b) mid-way-point adaptation

(c) end-point adaptation

Figure 9.2 Comparison of position adaptation among ProMP (yellow), KMP (blue), and our approach (green) for a C-shape trajectory with time as the input. The desired points to go through are marked with red stars.

9.1.2 Trajectory Adaptation

In this experiment, we study the problem of trajectory adaptation considering two scenarios: Position only and both position and velocity. Specifically, in position adaptation, three cases are studied, namely start-point, mid-way-point, and end-point adaptation. Similarly to Section 9.1.1, ProMP and KMP are employed for comparison purposes. LPV-DS is not considered here as it does

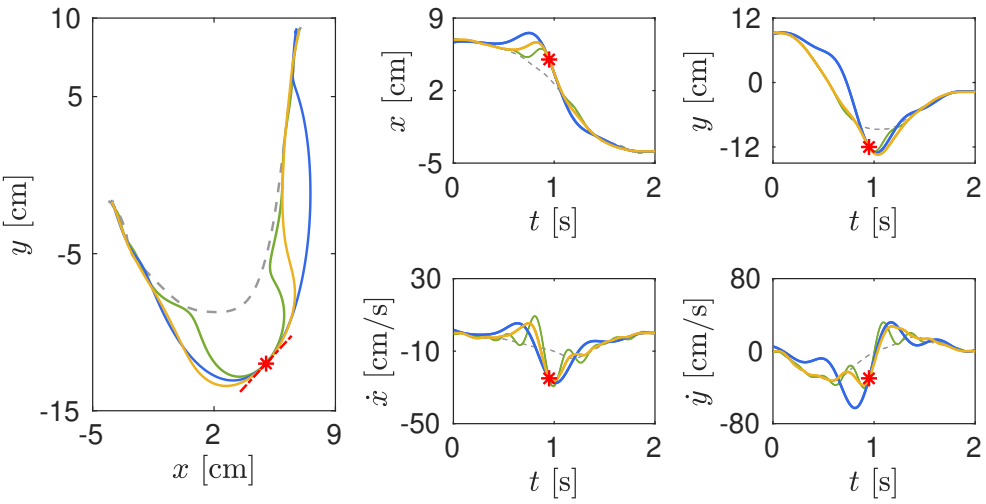
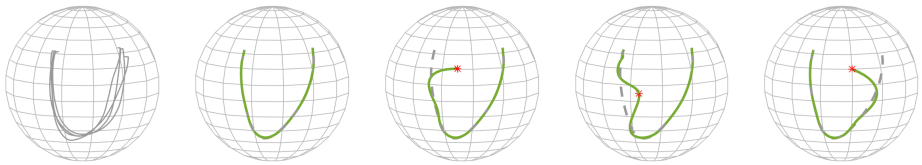


Figure 9.3 Comparison of both position and velocity adaptation among ProMP (yellow), KMP (blue), and our approach (green) for a J-shape trajectory (dashed gray) with time as the input. The desired position points to go through are marked with red stars and the desired velocity is depicted with a dashed red line.

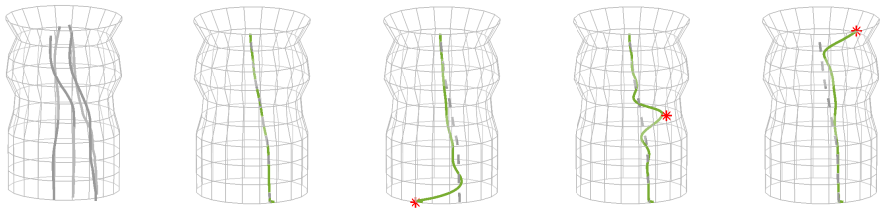
not provide an straightforward manner for trajectory adaption. For simplicity, here we select the KL divergence as imitation mode for our approach.

For position adaptation, we consider to adapt an imitated C-shape trajectory to go through variously positioned via-points with time as the input. We first evaluate start-point adaptation. To this end, we set a desired point at $[5\ 6]^\top$cm at time step $t = 0$s. Next, we move the desired point to $[-5\ 0]^\top$cm at time step $t = 0.5$s to study mid-way-point adaptation. Finally, the issue of end-point adaptation is studied by setting the desired point at $[10\ -9]^\top$cm at time step $t = 1.0$s. Different trajectory position adaptation cases are illustrated in Figure 9.2. It shall be observed that all algorithms are capable of respecting the additional via-point constraint with high accuracy. Our algorithm tends to converge to the original demonstrated trajectory closer compared with other methods and thus preserves higher imitation fidelity.

In the next experiment, we study the performance of adapting both position and velocity of an imitated trajectory. All three algorithms are applied to



(a) Trajectory imitation on a sphere



(b) Trajectory imitation on a generalized cylinder

Figure 9.4 Illustration of trajectory reproduction and adaptation on a manifold where multiple demonstrations are depicted in gray, trajectories generated by our algorithm are plotted in green and the desired via-point is marked with a red star.

adapt a J-shape trajectory whose total duration is 2 s. It shall be noted that additional first-order derivatives of basis functions are needed for ProMP to encode the dynamics of the motion, since it is based on parametric regression. By contrast, non-parametric methods like KMP and our approach do not depend on explicit basis functions. To temporally adapt the trajectory, at time step $t = 1\,\mathrm{s}$ we set a desired location at $[5\ \ -12]^\top \mathrm{cm}$ associated with a desired velocity $[-25\ \ -30]^\top \mathrm{cm/s}$. The obtained results are shown in Figure 9.3, where all the algorithms successfully generate the trajectory that goes through the desired position and desired velocity. The adaptation errors at the via point are negligible. It can be observed that the trajectory generated by our algorithm is affected the least by the additional adaptation requirements, while the other algorithms end up with larger imitation error due to larger deviation.

9.1.3 Manifold Trajectory Learning

In this experiment, we demonstrate a powerful feature unique to our approach, i.e. learning and adapting trajectories lying on manifolds. To show the effectiveness of our approach, we study the problem of imitation on two common manifolds, namely a sphere ($\mathbb{S}^2$) and a circular generalized cylinder ($\mathbb{R}^2 \times \mathbb{S}$). The performance of both reproduction from multiple demonstrations as well as adaptation towards a via point are evaluated. Since adapting trajectories on manifolds has never been addressed before in the context of robot imitation learning, we hereby report the experimental results of our approach alone.

For learning on a sphere, we first draw four U-shaped trajectories on a sphere, each having time duration $t = 1\,\text{s}$, as shown in the first column of Figure 9.4a. The radius of the sphere is 1 cm and the center is located at the origin of the coordinate system. The base point for conducting exponential and logarithmic mappings is chosen as $[0\ 0\ 1]^\top$cm. Likewise, the collected demonstration data is processed using GMM and a reference probabilistic trajectory is subsequently extracted by GMR. It should be noted that the collected data is defined on a Riemannian manifold, therefore, extended versions of GMM and GMR for Riemannian manifolds are employed (Zeestraten et al., 2017b). To test adaptive behavior, a via-point at $[-0.199\ -0.98\ 0]^\top$cm is set on the surface of the sphere for the trajectory to pass through at $t_v = 0.35\,\text{s}$. The second and third columns of Figure 9.4a show reproduction and adaptation trajectories from our algorithm, respectively. During each point prediction, the step size of Algorithm 3 is set to $\eta = 0.01$. We can observe that our algorithm is capable of accurately meeting the requirement of via-point (marked with a red star) adaptation while preserving the shape of the original demonstrated trajectory.

The other experiment we carry out is trajectory imitation on a circular generalized cylinder, which has a smoothly varying circular cross-section. We use the cylindrical coordinate system to express a data point (r, z, φ), where r is the radial distance from the z-axis, z denotes height and φ is the azimuth. Since a data point lying on a generalized cylinder consists of a two-dimensional Euclidean component and a circular component, we formulate its Riemannian representation with the help of the Cartesian product: $\mathbb{R}^2 \times \mathbb{S}^1$. The center of the

bottom of the cylinder is positioned at the origin of the cylindrical coordinate system and the cylinder center line is aligned with the z-axis. We first draw four demonstrations along the cylinder's central line, each one of 1 cm length and duration $t = 1$ s, as shown in the first column of Figure 9.4b. The second column of Figure 9.4b compares the reference retrieved by GMR and reproduction trajectory by our approach with step size again set to $\eta = 0.01$. It can be observed that our algorithm is able to reproduce the reference trajectories with high imitation fidelity. At time step $t_v = 0.5$ s, the trajectory is required to adapt towards a via-point located at $(1.51\,\text{cm}, 0.5\,\text{cm}, 0.863\,\text{rad})$. The learning results are shown in the third column of Figure 9.4b, where the adapted trajectory exactly passes through the via-point (marked with a red star).

9.2 Real Experiments

In this part, real experimental results to illustrate the effectiveness of our approach are provided. We test our approaches with two scenarios. The first one is a writing task, where the robot learns how to write a letter on a tabula rasa. The second one is a polishing task, where the robot learns how to polish on a burst-resistant exercise ball. To transfer the desired skills from the human to the robot, we demonstrate each individual task for several times.

Experimental setup

The robot used for accomplishing the polishing task is the KUKA LWR IV+, a 7-DoF robotic arm. With the joint torque sensors mounted at the actuators, it can be torque-controlled and the torque commands are sent to KUKA using the Fast Research Interface (FRI) at 500 Hz. There is also a 6-axis force-torque sensor installed on the end-effector. We choose to attach a white board marker onto the robot as our end-effector for the writing task and a 3D printed finger tool for the polishing task.

During the demonstration, the robot is switched to the gravity compensation mode to make the robot light to interact. The Cartesian trajectory of the robot end-effector is recorded to train our algorithm.

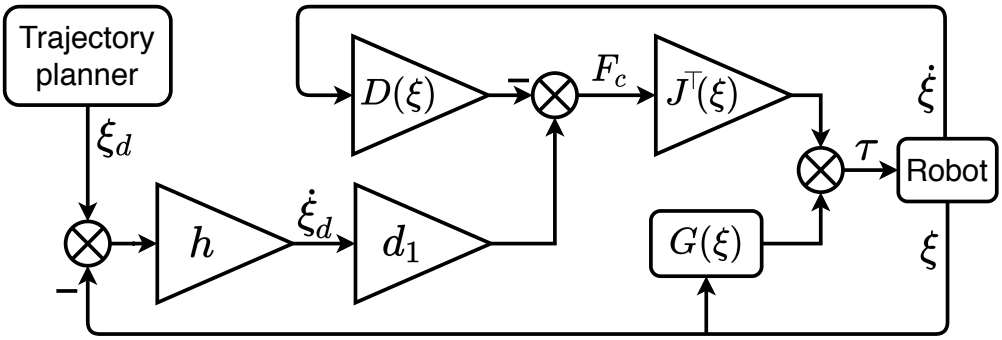


Figure 9.5 Block illustration of the employed passive controller. The algorithm is composed of a velocity-based control law and a gravity-compensation term.

Robot control

Tracking time-indexed position profiles usually poses a challenge for robots to achieve fast reactivity in response to external disturbances. To this end, we propose to employ a novel passive controller developed by Kronander and Billard (2015) to enhance the robot's robustness to large real-time disturbances. We consider the control of the robot's end effector in Cartesian space as in (Amanhoud et al., 2019); the corresponding translational control force $\mathbf{F}_c$ takes the form

$$\mathbf{F}_c = \mathbf{D}(\xi)(\dot{\xi}_d - \dot{\xi}) = d_1\dot{\xi}_d - \mathbf{D}(\xi)\dot{\xi}, \tag{9.1}$$

where ξ represents the end-effector position and $\mathbf{D}(\xi)$ is a state-varying damping matrix with the first eigen-vector $d_1 > 0$ aligned with the desired velocity $\dot{\xi}_d$. For our time-invariant task representation, we design the desired velocity profile as

$$\dot{\xi}_d = h(\xi_d - \xi), \tag{9.2}$$

where $h > 0$ weighs the tracking deviation.

In our experiment, the passive controller is only applied to the translational direction. For orientation control, a control moment is computed by setting the end-effector roughly pointing towards the center of the ball using spherical linear interpolation with a PD control law (Ghadami et al., 2015). The desired

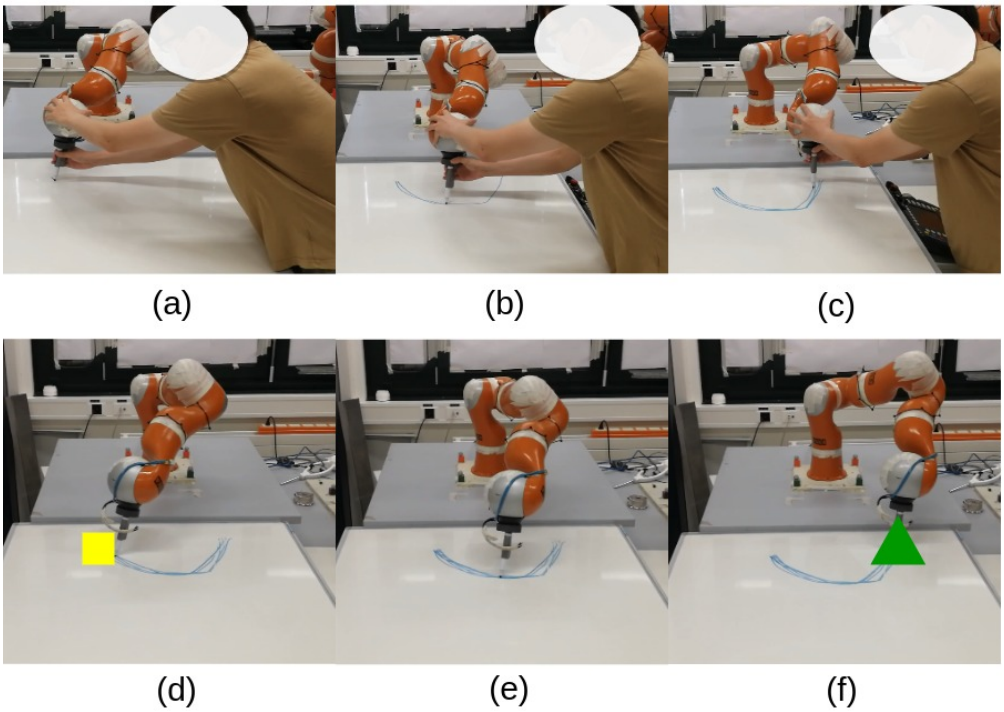


Figure 9.6 Snapshots of demonstration (*top row*) and reproduction (*bottom row*) of the writing task with KUKA LWR IV+ where the yellow square denotes the starting point and the green triangle denotes the end point.

quaternion q_d can be interpolated between q_0 and q_1 as

$$q_d = \frac{\sin(\Omega(1-w))q_0 + \sin(\Omega w)q_1}{\sin(\Omega)}$$

, where $\Omega = q_0^\top q_1$ and $w \in [0,1]$ is an interpolation parameter.

Finally, the control commands are computed by mapping the control wrench, consisting of control moment and force, to joint torques using corresponding robot Jacobian matrix. Figure 9.5 presents a block representation of our proposed controller.

9.2.1 Writing Task

In the writing task, the robot is taught to write on a tabula rasa with a white board marker. In order to make the mark leave a trace on the board, a small

constant velocity along the z-direction is set so that there is contact established between marker tip and the board. The procedure of kinesthetic teaching and reproduction is shown in Figure 9.6.

To evaluate the adaptation capability of our approach, we set different via-points for the robot end-effector marker to pass through. Figure 9.7 shows that the robot end-effector movement is modulated properly in order to respect the constraint incurred by different additional desired via-points.

9.2.2 Polishing Task

The purpose of the polishing task is to teach a robot by kinesthetic guidance so that it learns how to polish on the surface of a sphere manifold. Once reproduction of the skill is achieved, the robot is later required to polish a user-defined via-point on the surface to show adaptive behavior thanks to our approach.

Results and analysis

The illustration of the kinesthetic demonstration procedure as well as skill reproduction is shown in Figure 9.8. A teacher demonstrates the polishing task multiple times to the robot on a sphere whose radius is 0.3m and center position is identified as $[-0.7631\ -0.0271\ 0.0698]^\top$m in a least-squares sense. We then evaluate the reproduction capabilities of our approach on the robot. It shall be observed that the robot is able to reproduce the demonstrated task along the surface of the sphere. In another reproduction experiment, we apply external perturbations to the robot to test its working robustness. By lifting the robot arm, the robot is still able to converge to the original planned trajectory. The satisfactory compliant behavior makes it safe for the robot to interact with a human user. To evaluate the adaptation capability of our approach, we set different via-points for the robot end-effector to pass through. Figure 9.9 shows that the robot end-effector movement is modulated in order to respect the constraint incurred by different additional desired via-points. The real robot trajectories are plotted in Figure 9.10, where the relative position between the

trajectories and the ball as well as the zoomed-in trajectories are provided for clarity.

It should be noted that the trajectory generated by our approach exactly lies on the manifold in theory, yet sometimes there is small interference or deviation between the end-effector and the ball. This could be a result of the measuring error of the sphere information as well as the tracking error between the sent reference trajectory and the real robot end-effector position. In the future, this issue could be resolved by introducing contact perception signals at the level of the control architecture design.

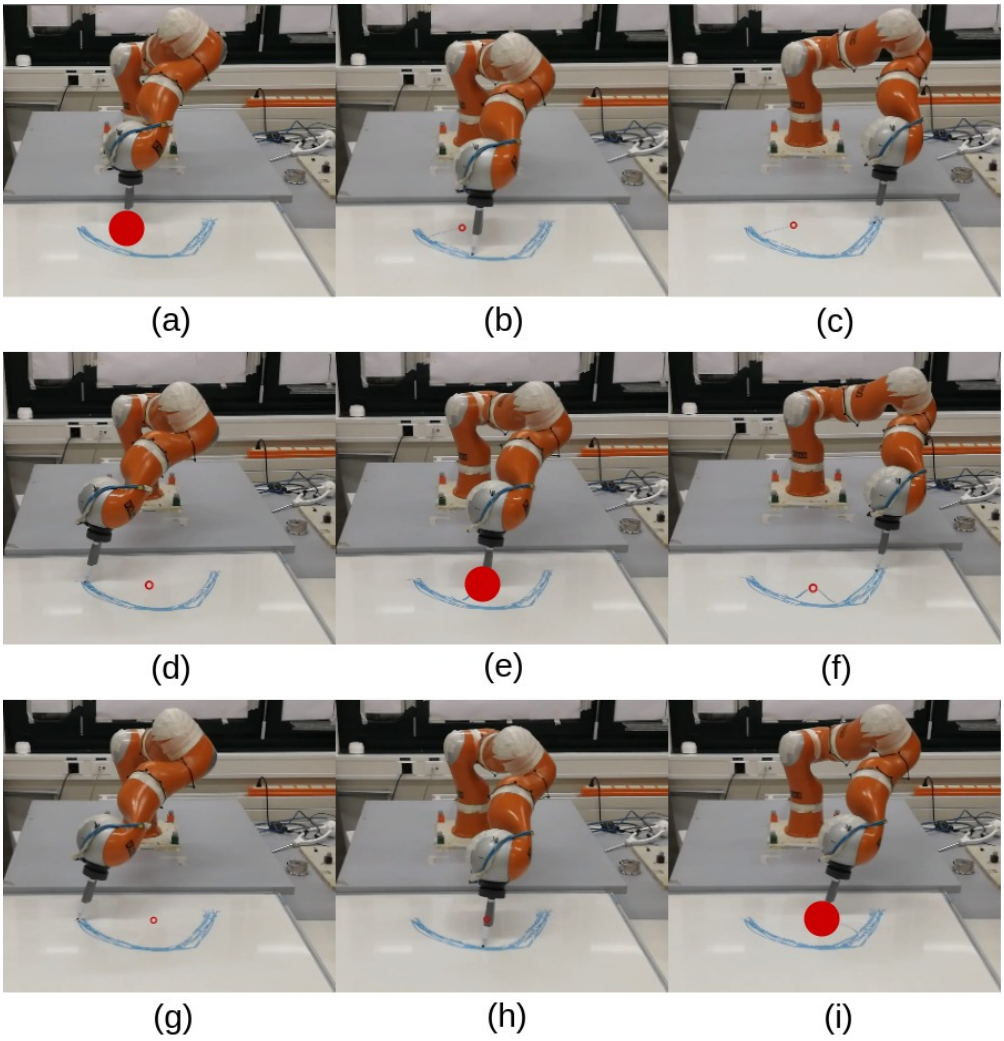


Figure 9.7 Snapshots of the trajectory adaption on a tabula rasa with KUKA LWR IV+, in terms of a new desired start point((a)-(c)), a new mid-way point ((d)-(f)) as well as a new desired end point. The desired point is denoted with a red round area.

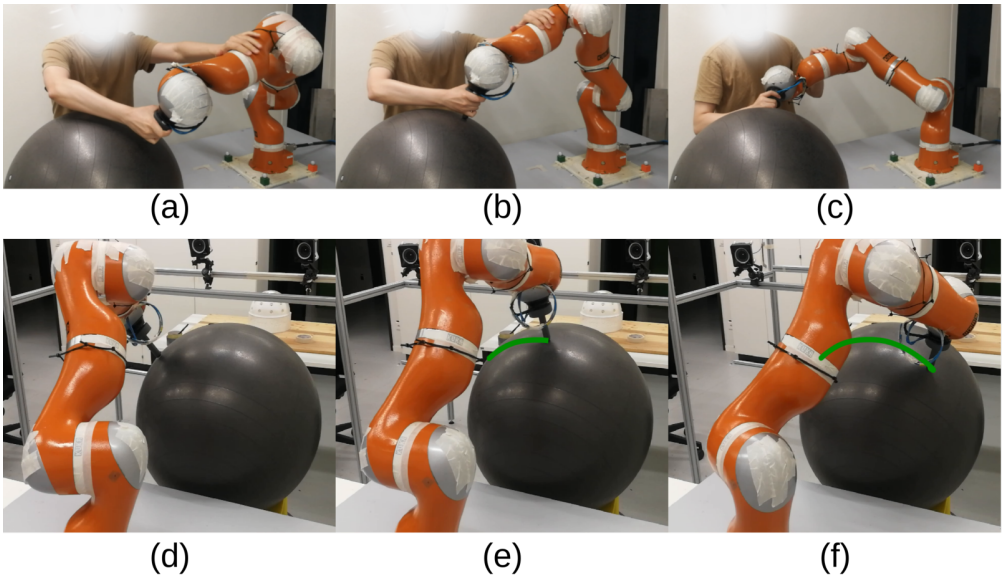


Figure 9.8 Snapshots of demonstration (*top row*) and reproduction (*bottom row*) of the polishing task with KUKA LWR IV+.

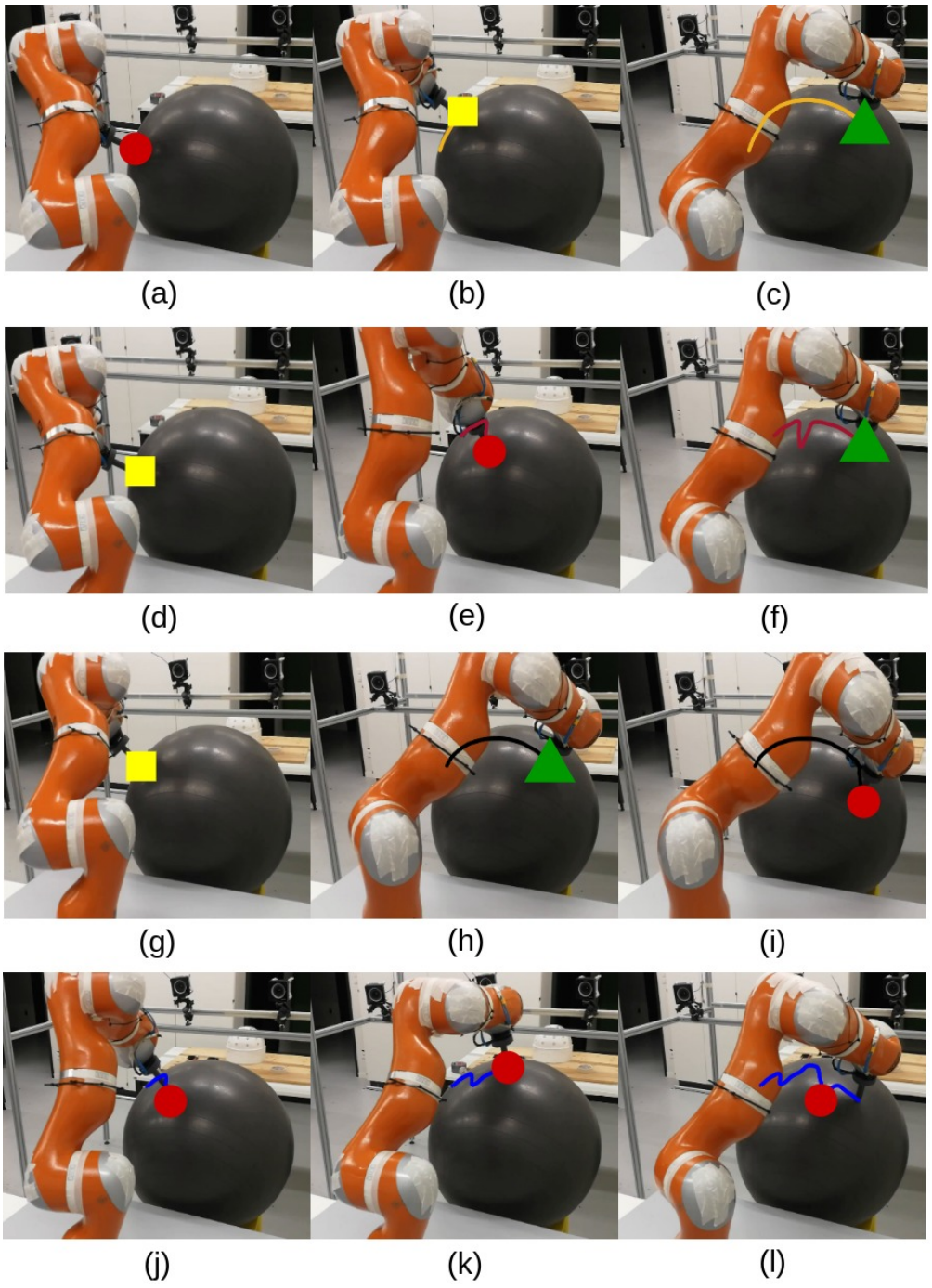


Figure 9.9 Snapshots of the trajectory adaption on a sphere with KUKA LWR IV+, in terms of a new desired start point((a)-(c)), a new mid-way point ((d)-(f)), a new desired end point ((g)-(i)) as well as multiple via-points((j)-(l)). Start points and end points of the original demonstration trajectories and new desired points are marked with squares, triangles, and circles.

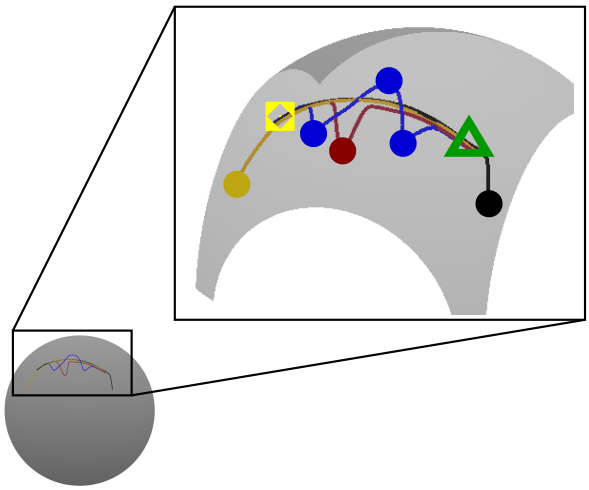


Figure 9.10 Plot of the robot end-effector adaptive trajectories in the task of polishing on the sphere, where the yellow, red, black, and blue curves represent start-point, mid-way-point, end-point, and multi-via-point adaption, respectively.

Chapter 10

Trajectory Improvement via Policy Search

In the previous part we have presented a structured prediction approach to probabilistic trajectory generation under the context of robot programming by demonstration. Although robot is able to learn policies from human demonstrations, it is sometimes however insufficient to accomplish a task using the learned policy only. Therefore, the learned trajectory shall be refined further to meet task requirements. Here, we consider the problem of multi-tasks sequencing in robot Cartesian space. Usually, in order to continuously combine and blend multiple movement primitives which are probabilistically encoded into a single movement, a common technique is to employ the *Gaussian product* Rasmussen (2004). By taking the product of trajectory distributions at each time step, the resulting trajectory again satisfies the Gaussian distribution. An important advantage of the Gaussian product is that it can capture the overlapping area of the activated trajectories. In general, the shape of the obtained trajectory shares higher similarity to the movement primitive that has higher probability density.

Suppose that there are in total K tasks that the robot has previously learned and for each task k the trajectory distribution $\mathbf{x}_k(t) \sim \mathcal{N}\left(\mu_k(t), \Sigma_k(t)\right)$ is retrieved using GMR. To allow for multi-tasks sequencing, we propose to modulate the activation status of the trajectories so that the tasks are executed one by one smoothly. Let us write the user-defined time-varying activation functions of

each task k at time t as $\pi_k(t)$ with $\sum_k \pi_k(t) = 1$. The resulting trajectory $\bar{\mathbf{x}}$ can be obtained by co-activating the movement primitives:

$$p(\bar{\mathbf{x}}) \propto \prod_k p(\mathbf{x}_k)^{\pi_k}. \tag{10.1}$$

At each time step, the resulting distribution obeys Gaussian $\bar{\mathbf{x}}(t) \sim \mathcal{N}\left(\bar{\mu}(t), \bar{\Sigma}(t)\right)$, with its mean and covariance matrix given by Rasmussen (2004)

$$\begin{aligned} \bar{\mu}(t) &= \bar{\Sigma}(t)\Big(\sum_k \big(\Sigma_k(t)/\pi_k(t)\big)^{-1}\mu_k(t)\Big), \\ \bar{\Sigma}(t) &= \Big(\sum_k \big(\Sigma_k(t)/\pi_k(t)\big)^{-1}\Big)^{-1}. \end{aligned} \tag{10.2}$$

Thus, multi-tasks sequencing in Cartesian space is accomplished by following the obtained trajectory.

10.1 Transformation from Cartesian Space to Joint Space

We need to transform Cartesian trajectories into joint space in order to control the robot. To do so, the Jacobian-based inverse kinematics technique $\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$ is employed, where $\mathbf{J} \in \mathbb{R}^{3\times d}$ is the robot Jacobian matrix Duan et al. (2018). In general, the corresponding discrete implementation incorporating null-space exploration is given by

$$\begin{aligned} \mathbf{q}_t &= \mathbf{q}_{t-1} + \mathbf{J}_{t-1}^{\dagger}(\mathbf{x}_t - \mathbf{x}_{t-1}) + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{N}(\Theta)\Delta t, \\ \mathbf{N}(\Theta) &= \Theta^{\top}\Phi(t), \end{aligned} \tag{10.3}$$

where $\mathbf{J}^{\dagger} = \mathbf{J}^{\top}(\mathbf{J}\mathbf{J}^{\top})^{-1}$ represents the Moore-Penrose pseudoinverse of $\mathbf{J}$ when $\mathbf{J}\mathbf{J}^{\top}$ is invertible, $\Delta t > 0$ is the time step, $\mathbf{I} \in \mathbb{R}^{d\times d}$ denotes an identity matrix, $\mathbf{N}$ corresponds to joint movement in null space, which is parameterized by $\Theta \in \mathbb{R}^{\mathscr{O}\times d}$, and $\Phi(t) \in \mathbb{R}^{\mathscr{O}\times 1}$ are basis functions with the total number $\mathscr{O}$ and

the i-th element defined using the normalized Gaussian kernel function

$$\mathrm{e}^{-h_i(t-c_i)^2} / \sum_j \mathrm{e}^{-h_j(t-c_j)^2}$$

with $h_i > 0$ and c_i equally spaced in the execution time interval.

When transforming the sequenced probabilistic Cartesian trajectory $\bar{\mathbf{x}}$ into joint space using (10.3), the obtained joint trajectory $\bar{\mathbf{q}}_t^C$ also satisfies the probabilistic distribution $\mathcal{N}(\bar{\mu}^C(t), \bar{\Sigma}^C(t))$ with mean $\bar{\mu}^C(t)$ and covariance matrix $\bar{\Sigma}^C(t)$ given by Huang et al. (2018b)

$$\begin{aligned} \bar{\mu}^C(t) &= \bar{\mathbf{q}}_{t-1}^C + \mathbf{J}^\dagger(\bar{\mu}_t - \bar{\mu}_{t-1}) + (\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})\mathbf{N}(\Theta)\Delta t, \\ \bar{\Sigma}^C(t) &= \mathbf{J}^\dagger \bar{\Sigma}_t \mathbf{J}^{\dagger\top}. \end{aligned} \tag{10.4}$$

It should be noted that the obtained joint trajectory upon transformation can only guarantee that the Cartesian constraint is respected. In order to take into account joint level imitation, the parameter Θ needs to be optimized according to the criteria proposed in the next section.

10.2 Optimization of Null-Space Parameters

The demonstrated joint trajectories can also be retrieved using GMR in a similar way to Cartesian trajectories. To represent different tasks, we propose to fuse the joint trajectories by GMM. It should be noted that joint trajectories are formulated in a different way from Cartesian ones, which are treated using the Gaussian product. This choice has been made since GMMs can preserve complete information on the robot's configuration for each sub-task together with their corresponding activation degree at each time step. Such information is very important, since one of the main goals for joint-level imitation is that the robot posture should be altered accordingly during the transition between consecutive tasks. In contrast, the Gaussian product only results in a single Gaussian with its mean value calculated by the summation of weighted sub-task means. In this way, the information on the original robot joint configuration for each sub-task is lost. Therefore, GMMs are a more reasonable choice for

modeling multi-tasks joint trajectories, in that it can capture richer information than the Gaussian product.

For each task k, we denote the corresponding joint trajectory distribution as $\mathbf{q}_k^J(t) \sim \mathcal{N}\big(\mu_k^J(t), \Sigma_k^J(t)\big)$. As mentioned above, the reference joint trajectory $\bar{\mathbf{q}}^J(t)$ is formulated in terms of GMM as

$$p(\bar{\mathbf{q}}^J(t)) = \sum_k \pi_k(t) \mathcal{N}\big(\mu_k^J(t), \Sigma_k^J(t)\big). \tag{10.5}$$

One can observe that there are concurrent conflicts between Cartesian trajectory $\bar{\mathbf{q}}_t^C$ and joint trajectory $\bar{\mathbf{q}}_t^J$, which are usually referred to as *competing constraints*[1] Calinon and Billard (2009). In order to minimize such conflicts, we propose to formulate the optimization objective from an information-theoretic perspective. A widely used technique in statistics and pattern recognition to measure the distance between two probability distributions is the *Kullback-Leibler (KL) divergence*, also known as *relative entropy* Kullback and Leibler (1951). Its usage in addressing competing constraints has been shown in Huang et al. (2018a), where the main purpose was to design a minimal intervention controller. By contrast, our focus is on sequencing multiple tasks. In order to minimize the inconsistency between $\bar{\mathbf{q}}_t^C$ and $\bar{\mathbf{q}}_t^J$, the null space will be explored by optimizing the null-space parameter Θ according to the KL-divergence-based objective:

$$J_{\mathrm{KL}}(\Theta) = \sum_t D_{\mathrm{KL}}\big(p(\bar{\mathbf{q}}_t^J) \parallel p(\bar{\mathbf{q}}_t^C; \Theta)\big), \tag{10.6}$$

where the expression for the KL divergence is defined as

$$D_{\mathrm{KL}}\big(p(\bar{\mathbf{q}}_t^J) \parallel p(\bar{\mathbf{q}}_t^C; \Theta)\big) \triangleq \int p(\bar{\mathbf{q}}_t^J) \log \frac{p(\bar{\mathbf{q}}_t^J)}{p(\bar{\mathbf{q}}_t^C; \Theta)} d\mathbf{q}_t. \tag{10.7}$$

It should be noted that the KL divergence is asymmetric. The choice of the direction of the KL divergence is problem-dependent. In our case, the moment projection form rather than the information projection form is employed, since during the task transition period when $p(\bar{\mathbf{q}}_t^J)$ has multiple modes, $p(\bar{\mathbf{q}}_t^C; \Theta)$ will

[1] Time dependence is moved to subscript for simplicity.

choose to blur different modes together so as to yield more natural action for tasks switch.

One issue arising from the optimization objective in (10.6) is that the calculation of the KL divergence between GMMs and a Gaussian is not analytically tractable. Therefore, an approximation method for (10.7) is necessary to facilitate solving the optimization problem. Theoretically, there are several approaches to the calculation of the KL divergence between GMMs, such as Monte Carlo sampling, unscented transformation, matched bound approximation, etc. Here, the one based on the variational lower bound to the likelihood is chosen because of its simple closed-form expression as well as its high accuracy Hershey and Olsen (2007). Formally, we approximate (10.7) as

$$\begin{aligned} &D_{\mathrm{KL}}\big(p(\bar{\mathbf{q}}_t^J) \parallel p(\bar{\mathbf{q}}_t^C;\Theta)\big) \approx \\ &\sum_k \pi_k(t) \log \frac{\sum_{k'} \pi_{k'}(t) \mathrm{e}^{-D_{\mathrm{KL}}\left(p(\mathbf{q}_{k,t}^J) \| p(\mathbf{q}_{k',t}^J)\right)}}{\mathrm{e}^{-D_{\mathrm{KL}}\left(p(\mathbf{q}_{k,t}^J) \| p(\bar{\mathbf{q}}_t^C;\Theta)\right)}}. \end{aligned} \tag{10.8}$$

First let us denote $\mathbb{E}_{p(\bar{\mathbf{q}}_t^J)}[\log\left(p(\bar{\mathbf{q}}_t^C;\Theta)\right)]$ as $L_J(C)$. The KL-divergence from (10.7) can be decomposed as[2]

$$D_{\mathrm{KL}}\big(p(\bar{\mathbf{q}}^J) \,\|\, p(\bar{\mathbf{q}}^C;\Theta)\big) = L_J(J) - L_J(C). \tag{10.9}$$

We define variational parameters $\phi_{k'|k} > 0$ with $\sum_{k'} \phi_{k'|k} = 1$. The lower bound to $L_J(J)$ can be derived based on the Jensen's inequality Hershey and Olsen

[2]Without ambiguity, time dependence is dropped out here for simplicity.

(2007):

$$\begin{aligned} L_J(J) &= \mathbb{E}_{p(\bar{\mathbf{q}}^J)}[\log\left(p(\bar{\mathbf{q}}^J)\right)] \\ &= \mathbb{E}_{p(\bar{\mathbf{q}}^J)}[\log \sum_{k'} \pi_{k'} p(\mathbf{q}^J_{k'})] \\ &= \mathbb{E}_{p(\bar{\mathbf{q}}^J)}[\log \sum_{k'} \phi_{k'|k} \frac{\pi_{k'} p(\mathbf{q}^J_{k'})}{\phi_{k'|k}}] \\ &\geq \mathbb{E}_{p(\bar{\mathbf{q}}^J)}[\sum_{k'} \phi_{k'|k} \log \frac{\pi_{k'} p(\mathbf{q}^J_{k'})}{\phi_{k'|k}}] \\ &\triangleq \mathscr{L}_J(J, \phi). \end{aligned} \tag{10.10}$$

The obtained lower bound is maximized with respect to the variational parameters ϕ_k. Such constrained optimization problem can be solved by using Lagrangian multiplier. The maximum value is achieved when Durrieu et al. (2012)

$$\phi^*_{k'|k} = \frac{\pi_{k'} \mathrm{e}^{-D_{\mathrm{KL}}\left(p(\mathbf{q}^J_k) \| p(\mathbf{q}^J_{k'})\right)}}{\sum_{\hat{k}} \pi_{\hat{k}} \mathrm{e}^{-D_{\mathrm{KL}}\left(p(\mathbf{q}^J_k) \| p(\mathbf{q}^J_{\hat{k}})\right)}}. \tag{10.11}$$

By substituting (10.11) into (10.10), the lower bound now becomes (Durrieu et al., 2012)

$$\begin{aligned} \mathscr{L}_J(J, \phi^*) = \sum_k \pi_k \log \sum_{k'} \pi_{k'} \mathrm{e}^{-D_{\mathrm{KL}}\left(p(\mathbf{q}^J_k) \| p(\mathbf{q}^J_{k'})\right)} \\ - \sum_k \pi_k H\left(p(\mathbf{q}^J_k)\right), \end{aligned} \tag{10.12}$$

where $H(p(\mathbf{q}^J_k))$ is the entropy of $p(\mathbf{q}^J_k)$ with its expression given as (Bishop, 2006)

$$H\left(p(\mathbf{q}^J_k)\right) \triangleq -\int p(\mathbf{q}^J_k) \log p(\mathbf{q}^J_k) d\mathbf{q}. \tag{10.13}$$

For the calculation of $L_J(C)$, the exact expression is available immediately since $p(\bar{\mathbf{q}}^C)$ has only one component:

$$L_J(C) = \sum_k \pi_k \log \mathrm{e}^{-D_{\mathrm{KL}}\left(p(\mathbf{q}_k^J) \| p(\bar{\mathbf{q}}^C)\right)} - \sum_k \pi_k H\left(p(\mathbf{q}_k^J)\right). \tag{10.14}$$

By substituting (10.12) and (10.14) into (10.9), we can finally obtain the approximation as (10.8).

Now the involved KL divergence calculation in (10.8) is only between multivariate Gaussian distributions, which has an analytical solution. Taking the KL divergence between $\mathbf{q}_{k,t}^J$ and $\mathbf{q}_t^C$ as an example, its form is given by Hershey and Olsen (2007)

$$\begin{aligned} D_{\mathrm{KL}}\left(p(\mathbf{q}_{k,t}^J) \parallel p(\bar{\mathbf{q}}_t^C;\Theta)\right) = \frac{1}{2}\Big(&\mathrm{Tr}(\bar{\Sigma}_t^{C-1}\Sigma_{k,t}^J) + \\ &(\bar{\mu}_t^C - \mu_{k,t}^J)^\top \bar{\Sigma}_t^C - 1(\bar{\mu}_t^C - \mu_{k,t}^J) + \log\frac{|\bar{\Sigma}_t^C|}{|\Sigma_{k,t}^J|} - d\Big), \end{aligned} \tag{10.15}$$

where $\mathrm{Tr}(\cdot)$ and $|\cdot|$ denote the trace and the determinant of a matrix, respectively. The KL divergence between other terms can be calculated similarly.

Given the complexity of the formulated optimization problem, it can be addressed with the help of RL algorithms. In this work, we apply the simplified path improvement with path integrals in search for the optimal null-space parameter Θ Stulp and Sigaud (2013). The update rule for Θ is given as

$$\Theta_{i+1} \leftarrow \Theta_i + \frac{\sum_{l=1}^L \varepsilon_l \mathrm{e}^{-\frac{1}{\lambda}J(\Theta_i+\varepsilon_l)}}{\sum_{l=1}^L \mathrm{e}^{-\frac{1}{\lambda}J(\Theta_i+\varepsilon_l)}}, \tag{10.16}$$

where Θ_i is the initial parameter value for the i-th iteration, $\lambda > 0$ is a constant which can be considered as a factor controlling the learning rate and $\varepsilon_l \sim \mathcal{N}(\mathbf{0}, \Sigma_\varepsilon)$ is the exploration noise for the l-th trial. The update rule keeps being executed until no further improvement on Θ is observed.

Algorithm 4: Multi-tasks sequencing with competing constraints

Collect the dataset of K tasks, each containing M demonstrations having length N: $\{\{\{t^k_{n,m}, \mathbf{x}^k_{n,m}, \mathbf{q}^k_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}\}_{k=1}^{K}$;
Retrieve Cartesian and joint trajectory distribution $\{\mathbf{x}_k(t), \mathbf{q}_k(t)\}$ from demonstrations using GMR ;
Specify task sequencing schedule $\pi_k(t)$;
Initialize RL hyperparameters Θ_0, Σ_ε, λ, h_i, c_i, L ;
Repeat;
for $l = 1$ *to* L **do**
    Sample $\varepsilon_l \sim \mathcal{N}(\mathbf{0}, \Sigma_\varepsilon)$ $\Theta \leftarrow \Theta + \varepsilon_l$;
    for $t = 1$ *to* N **do**
        Blend Cartesian trajectory using (10.2) ;
        Transform into joint trajectory using (10.4);
        Estimate time step cost by (10.8) and (10.15)
    Compute trial cost as (10.6)
Update Θ according to (10.16);
Until Θ converge;
Calculate the optimal trajectory $\{\mathbf{q}^*\}_{t=1}^{N}$ from (10.3) $\{\mathbf{q}^*\}_{t=1}^{N}$;

The complete proposed method for multi-tasks sequencing with competing constraints is summarized in Algorithm 4.

10.3 Experimental Evaluations

The proposed method is validated on the iCub humanoid robot Natale et al. (2017), both in simulation (Coumans and Bai, 2019) as well as in real experiments. The case study we choose for concept proof is to sequence a pick-and-place task as well as a cleaning task. In this section, we will show that by teaching the robot these two tasks separately, the robot can learn by itself to sequence them together.

Firstly, the robot is taught two tasks individually with each one demonstrated for 5 times. A GMM with 5 states is used for modeling the joint probabilistic distribution between time and the corresponding output. The demonstrated Cartesian and joint trajectories with GMM modeling are shown in Fig. 10.1

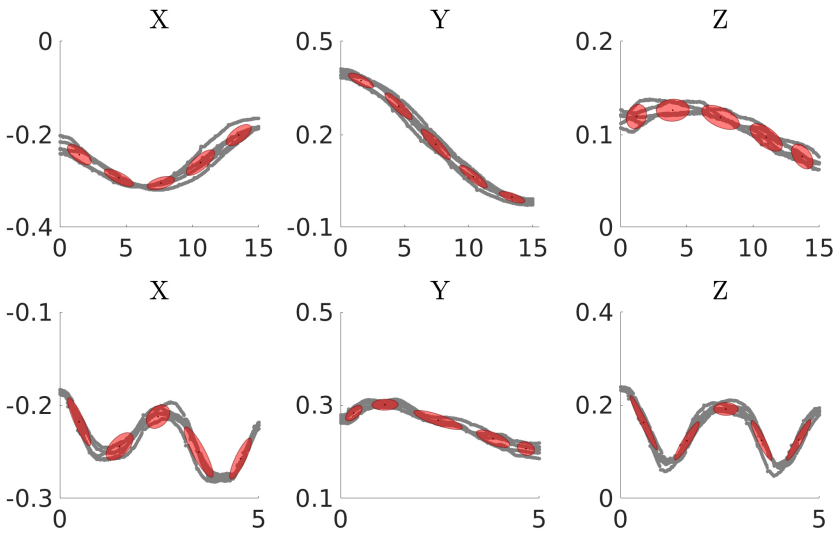


Figure 10.1 GMM modeling of the demonstrated Cartesian trajectories for the pick-and-place task (*top row*) and the cleaning task (*bottom row*). The grey trajectories represent multiple demonstrations and the red ellipses are Gaussian components in GMM.

and Fig. 10.2, respectively. For robot control, GMR is employed to extract the reference trajectory. The reproduction of the demonstrated tasks on the real iCub humanoid robot is shown in Fig. 10.3.

Next, iCub is required to sequence two demonstrated tasks together. To this end, the experimental set-up is conceived as follows: iCub will firstly pick a kitchen sponge handed over by a user, then perform the cleaning task, and, lastly, place the kitchen sponge at the destination. To start with, Cartesian trajectories of the two demonstrated tasks will be blended by Gaussian product. The extracted Cartesian trajectories for each task as well as the resulting blended trajectory are shown in the top row of Fig. 10.4. The corresponding weights used in the Gaussian product are given by a time-varying activation function, as shown in the bottom row of Fig. 10.4. According to the given activation function, the pick-and-place task will last for 15 s, during which the cleaning task will be triggered and last for 5 s.

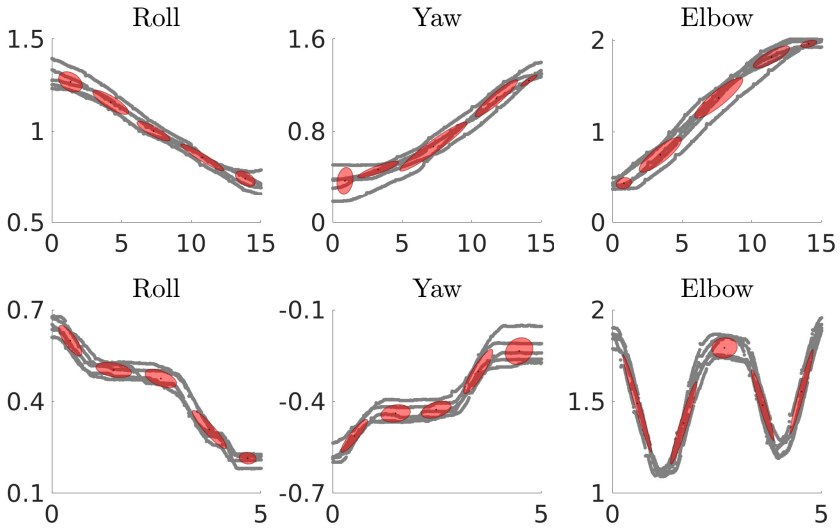


Figure 10.2 GMM modeling of the demonstrated joint trajectories (shoulder roll, yaw, and elbow) for the pick-and-place task (*top row*) and the cleaning task (*bottom row*). The grey trajectories represent multiple demonstrations and the red ellipses are Gaussian components in GMM.

Upon the availability of the blended Cartesian trajectories, the corresponding joint trajectories shall be obtained by Jacobian-based inverse kinematics (IK) with the null-space parameters exposed. These parameters will be optimized to drive the robot configuration towards the demonstrated joint trajectories, which are modeled using GMM. The optimization is tackled by the reward-weighted RL algorithm Stulp and Sigaud (2013) with the hyperparameters empirically set as $\Theta_0 = \mathbf{0}$, $\Sigma_\varepsilon = 10^{-2}\mathbf{I}$, $\lambda = 0.3$, and $L = 5$. The resulting optimized joint trajectories and their comparison with the trajectories from Jacobian-based inverse kinematics are shown in Fig. 10.5. The trajectories from inverse kinematics deviate from the demonstrated joint trajectories very much and therefore the imitation of the joint trajectories is broken. Instead, the trajectories with the optimal null-space parameters tend to match the corresponding demonstrated joint trajectories. The robot is trying to maintain the configuration of each demonstrated task during the process of sequencing the two tasks together.

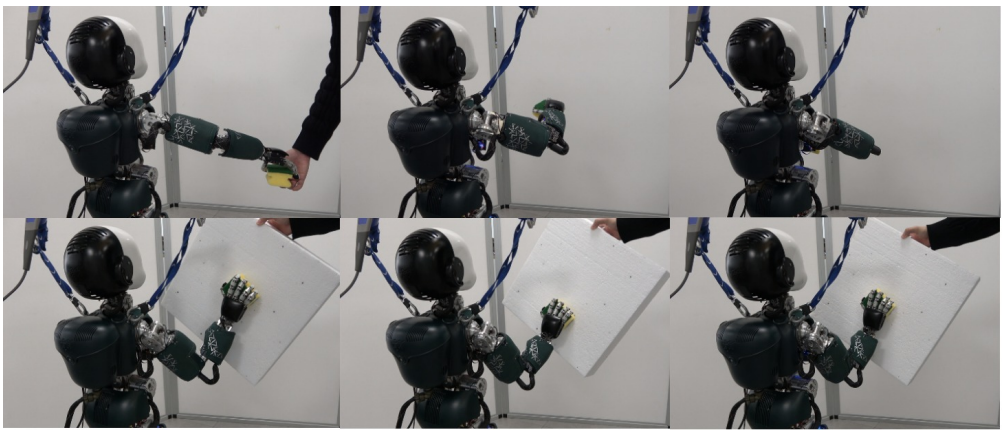


Figure 10.3 Snapshots of reproduction for the pick-and-place task (*top row*) and the cleaning task (*bottom row*).

The RL algorithm performance is reported in Fig. 10.6. The parameters to be optimized are updated for 20 times with each one running 5 trials. The error bar is obtained by repeating the learning process for 5 times.

Finally, we evaluate our proposed method on the real iCub humanoid robot. The comparison between the optimized joint trajectories and the inverse-kinematics-based solution is provided in Fig. 10.7. It can be easily observed that the robot's movement appears more natural by running the optimized joint trajectories. This is evidenced by the fact that when the robot is undergoing task switch, the corresponding imitation emphasis also shifts from one task to another gradually.

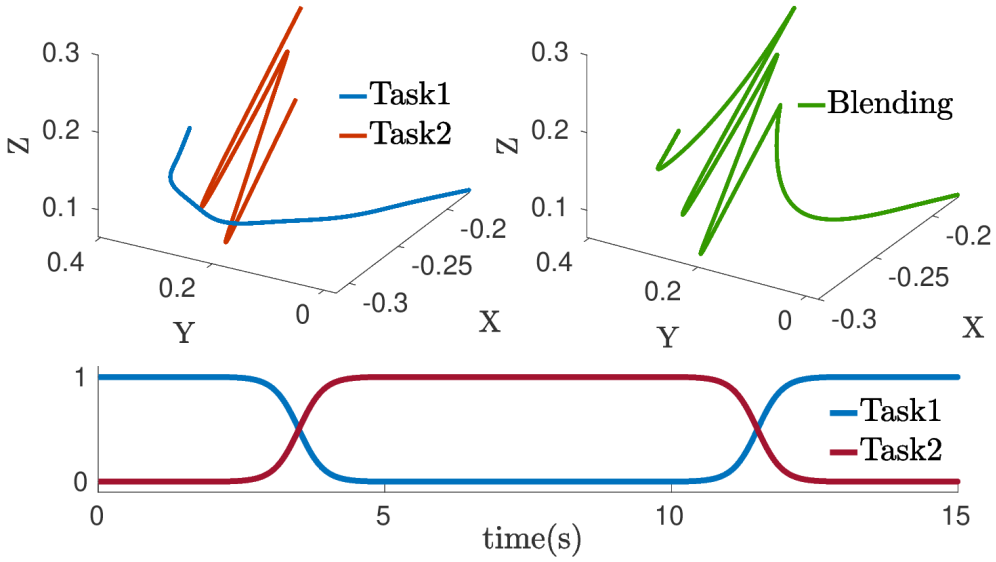


Figure 10.4 Cartesian trajectories sequencing (*top row*) with their corresponding activation functions (*bottom row*). The blue trajectory represents the pick-and-place task, the red trajectory represents the cleaning task and the green trajectory shows the result of sequencing.

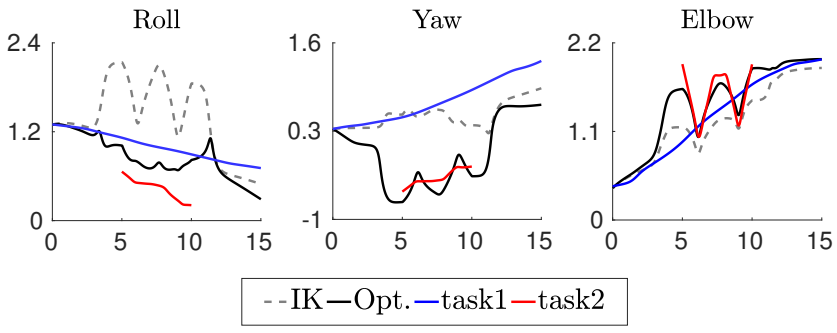


Figure 10.5 Comparison of the joint trajectories with the optimal null-space parameters and inverse kinematics. The joint trajectories for two tasks are also included for reference.

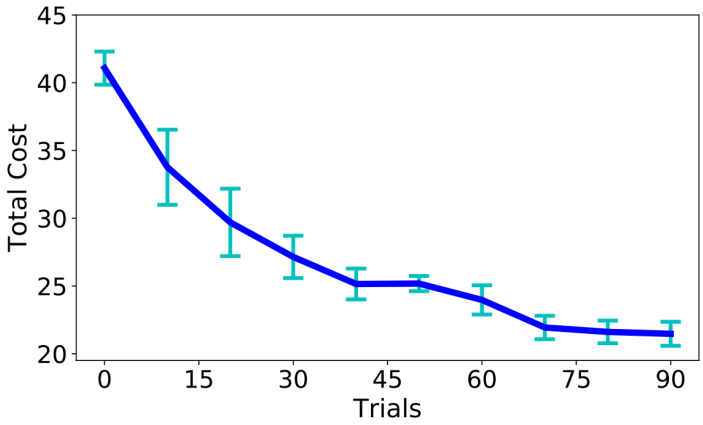


Figure 10.6 The error-bar curve of the KL-divergence based cost during learning of the optimal null-space parameters. The vertical bars denote the standard deviations

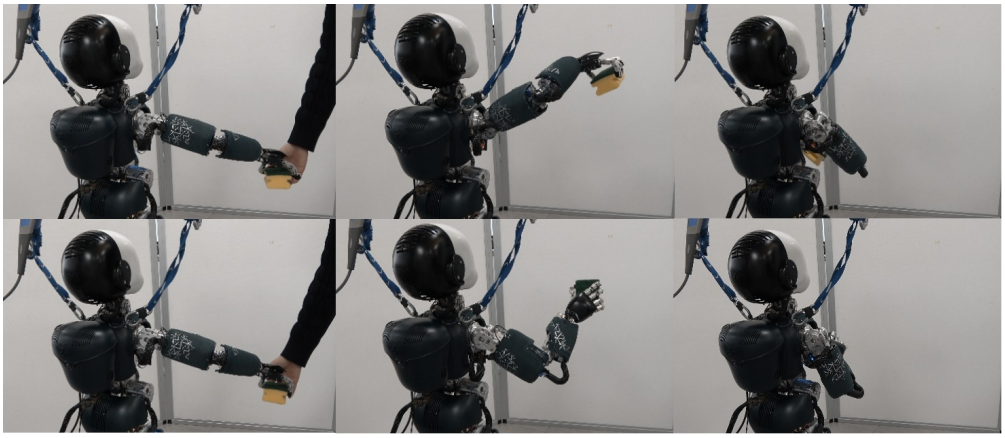


Figure 10.7 Snapshots of the results of sequencing two tasks. The joint trajectories with the optimal null-space parameters (*bottom row*) result in more natural behavior than inverse kinematics solution (*top row*).

Chapter 11

Constrained Imitation Learning

As policy search is able to refine trajectory further to meet certain requirements, one possible issue is that random search during the exploration process could make the robot violate the joint limits. Also, it can be conceived that the modified trajectories under external obstacles are very susceptible to the strength of the potential field which could drive joint trajectories out of the allowable range. In this chapter, we address the joint limits violation problem in order to guarantee that joint limits are always respected. We develop our joint limits avoidance strategy, called Constrained Dynamic Movement Primitives (CDMPs) (Duan et al., 2018). Although our approach is based on DMP, the idea can be readily applied to other types of movement primitives. In short, CDMPs are derived by parameterizing the original trajectory using the hyperbolic tangent function

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}.$$

We make modifications on the hyperbolic tangent space such that the joint trajectories will always evolve within the given bound. Formally, assume the joint limits are determined by $\mathbf{q}_{\min}$ and $\mathbf{q}_{\max}$. The feasible joint space in terms of the exogenous variable ξ is as follows

$$\mathbf{q}(\xi) = \mathbf{q}_{\mathrm{e}} \tanh(\xi) + \mathbf{q}_{\mathrm{o}}, \tag{11.1}$$

where $\mathbf{q}_\mathrm{e} = \mathrm{diag}\big(\frac{1}{2}(\mathbf{q}_\mathrm{max} - \mathbf{q}_\mathrm{min})\big)$ and $\mathbf{q}_\mathrm{o} = \frac{1}{2}(\mathbf{q}_\mathrm{max} + \mathbf{q}_\mathrm{min})$. Therefore, given the desired reference trajectory $\mathbf{q}_\mathrm{d}$, its transformation into tanh-space is given as

$$\xi_\mathrm{d} = \mathrm{arctanh}\big(\mathbf{q}_\mathrm{e}^{-1}(\mathbf{q}_\mathrm{d} - \mathbf{q}_\mathrm{o})\big). \tag{11.2}$$

Here, we briefly introduce DMPs (Ijspeert et al., 2013). DMPs are a flexible representation for motion primitives. It can be used to generate either discrete or rhythmic movements. Here, only discrete movements are considered.

Consisting of a set of linear differential equations, DMPs can be interpreted as a damped spring model with the following transformation system:

$$\tau\ddot{y} = \alpha(\beta(g-y) - \dot{y}) + f + P(y,\dot{y}), \tag{11.3}$$

$$\tau\dot{g} = \alpha_g(g_o - g), \tag{11.4}$$

where y, $\dot{y}$, g are system states; τ, α, β and α_g are positive constants; g_o is the final goal; P is called *coupling term* and its form is dependent on the choice of the potential field. Although the goal parameter is usually a constant, here we formulate it in a differential equation as Eq. (11.4) to allow for goal learning, as described later. Theoretically, we can obtain arbitrary continuous shapes by disturbing the dynamical system using the nonlinear forcing term f, defined as:

$$f = \mathbf{g}_t^T \Theta, \tag{11.5}$$

$$[\mathbf{g}_t]_j = \frac{\Psi_j(s_t) \cdot s_t}{\sum_{k=1}^{p} \Psi_k(s_t)}(g_t - y_0), \tag{11.6}$$

$$\Psi_j = \exp(-0.5h_j(s_t - c_j)^2), \tag{11.7}$$

where $\mathbf{g}_t$ are called basis functions, composed by the initial position y_0 and the Gaussian kernel Ψ_j with parameters $h_j > 0$ and $c_j \in [0,1]$. Θ is the vector of basis function weights or shape vector since it decides the general shape of the trajectory. Usually, it is Θ that represents a specific policy in RL algorithms. Given a trajectory from demonstration, Θ can be efficiently calculated by weighted linear regression. Phase variable s_t is calculated from a canonical

system that drives the whole dynamic system:

$$\tau \dot{s}_t = -\alpha s_t. \tag{11.8}$$

Since s_t moves from one to zero, convergence to the goal g_o is guaranteed as f vanishes at the end of the movement. It should be noted that although each DoF typically requires one transformation system, only one single canonical system is enough to coordinate all DoFs.

It should be noted that the training of DMP is the same as the usual procedure, i.e. fitting the acceleration profile with Gaussian radial basis functions. Here the only difference is that it happens in the tanh-space. The working flow of CDMPs is illustrated in Fig. 11.1. As the trajectory modification events such as additive exploratory noise from policy search and obstacle-avoidance modifications happen in the tanh-space, the learned trajectory is guaranteed to evolve within the specified safety range as illustrated in Fig. 11.2.

All the experiments are conducted in the Pybullet simulation environment with the external off-the-shelf kinematics and dynamics library iDynTree (Nori et al., 2015).

11.1 Experiments for Comparison of CDMPs and DMPs

We evaluate the proposed method by performing two toy examples.

1) Task description: In this task, the effectiveness of joint limit avoidance of CDMPs is demonstrated by comparing the evolution of the associated joint states with the ones of DMPs, whose parameters are the same ones used in Ijspeert et al. (2013). Usually there are two cases that have high chance to hit the joint limits, namely goal adaption and improper acceleration. Goal adaption refers to modifying DMPs dynamics by changing the goal parameter only. Because a simple change of the goal state automatically creates a complex rescaling of the entire movement, the maximum value of the rescaled system is difficult to be bounded in DMPs. In addition, very large acceleration from the

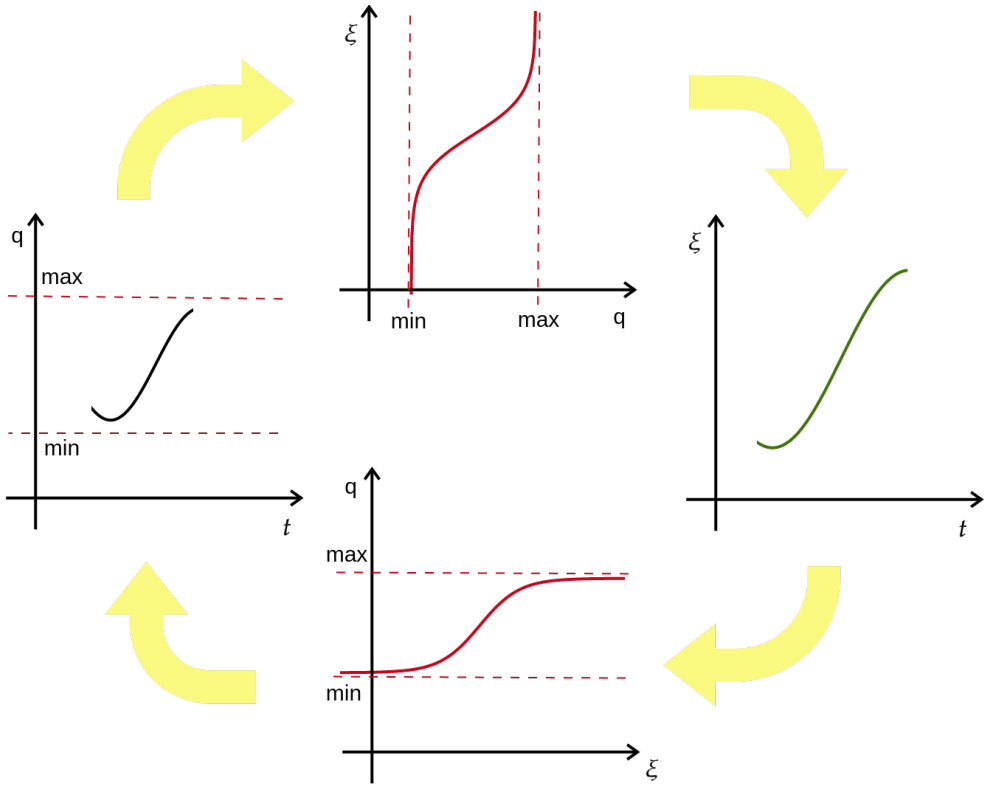


Figure 11.1 Illustration of the principle of CDMPs. Joint trajectory q is first transformed into the exogenous state ξ using arctangent hyperbolic function and then transformed back using tangent hyperbolic function. Trajectory modifications are done in *tanh*-space.

dramatic change of a trajectory could also contribute to joint limit violation as a result of overshoot. In this task, however, it will be shown that the allowable joint limit value can be easily satisfied by using CDMPs.

2) Experimental results and discussion: It can be seen in Fig. 11.3a that given an eligible demonstrated trajectory, both DMPs and CDMPs can accurately imitate a given trajectory. However, when adapting the final goal to a higher value, the obtained DMPs trajectory has a maximum value of 48.5 deg and it exceeds the maximum limit 35 deg. On the other hand, by using CDMPs, the range of the obtained trajectory is bounded all the time with the maximum value 34.4 deg. Moreover, when the imitated trajectory has a steep change, as

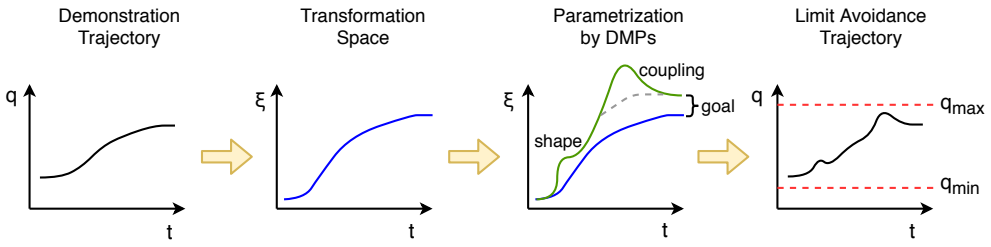


Figure 11.2 Illustration of the usage of CDMPs for safety-limits avoidance in face of exploration noises and obstacle-avoidance modifications.

shown in Fig. 11.3b, DMPs cannot be guaranteed to evolve within the specified limit (55 deg in the example). The maximum value due to the overshoot of DMPs is 58.9 deg, while for CDMPs it is bounded to 52.8 deg.

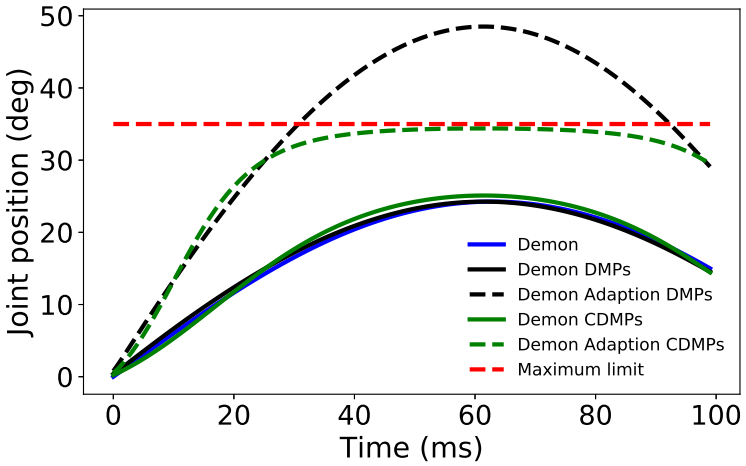
It can be concluded from the above toy examples that the proposed CDMPs can overcome the problem of joint limit violation, which is very common in the traditional DMPs.

It is also possible to apply CDMP with reinforcement learning algorithm such as policy search. Like the usual treatment of the DMP and policy search, the additive noise is added to the parameters of CDMP, thus allowing the obtained exploratory trajectory to search for the best trajectory with the maximum rewards. As the trajectory generated by CDMP is determined to be bounded as previously discussed, the exploratory trajectory will never exceed the allowed safety region. Such characteristic allows safe exploration, which is a common issue in the field of reinforcement learning.

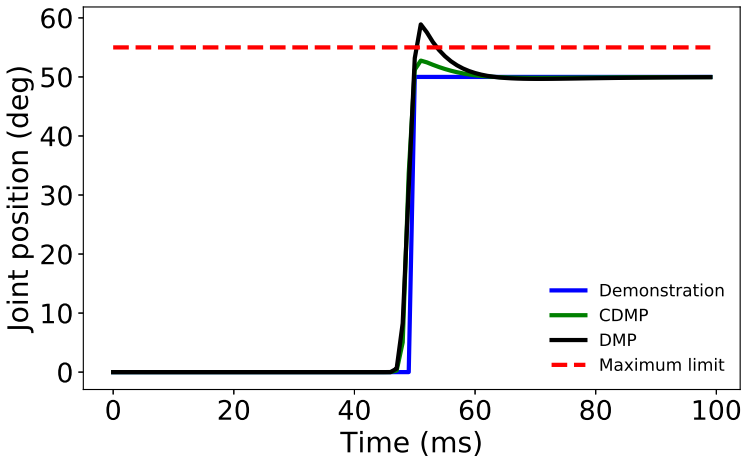
Limits avoidance is an extensively studied topic, there is large literature on algorithms of joint limit avoidance. Mostly used methods include squash function (Deisenroth and Rasmussen, 2011), barrier function (Ames et al., 2019), gradient projection method (Marey and Chaumette, 2010) etc. Therefore, future direction of this work would be comparison among different joint limit avoidance methods. Especially, it will be interesting to identify which method is preferred under certain situation. Also, as implementation of the proposed approach requires pre-specified joint limits range, it will be interesting to investigate how to incorporate dynamical changing joint limit range as it is usually

the case in a non-static environment or human-robot interaction scenario. Such real-time adaptation of joint limit range could help the robot operate more safely in a dynamical environment.

Further, as the proposed approach is designed only for the case of joint or task space position. Extension to the case of orientation will be interesting. Generating orientation-constrained trajectories will be helpful for the case of orientation-sensitive applications such as the peg-in-pole task.



(a) Goal adaption



(b) Improper acceleration

Figure 11.3 Toy examples for CDMPs and DMPs comparison.

Chapter 12

Probabilistic Trajectory Transfer

For some cases, trajectory transfer will be needed as it can adapt a learned trajectory from the geometry at demonstration time to the geometry at reproduction time (Schulman et al., 2016). This is especially desired when robot needs to reproduce the learned trajectory by interacting with flexible materials. Typically, a common technique to find the matching relationship between two sets of features is nonrigid registration (Wahba, 1990). It has already witnessed applications in a couple of fields, such as medical image analysis (Maintz and Viergever, 1996), object recognition (Belongie et al., 2002), 3D scans (Brown and Rusinkiewicz, 2007), etc.

Nonrigid registration has also received attention in imitation learning (Ahmadzadeh and Chernova, 2018; Schulman et al., 2016), yet the application is only focused on the deterministic case. In this part, we show how to realize probabilistic trajectory transfer with nonrigid registration. The basic idea is that the points with higher variances shall be allowed to deviate more after the transformation.

Formally, assume that the chosen P landmark points are $\{\hat{\mu}_{l_i}\}_{i=1}^{P}$ with the points index stored as $\{l_1, \ldots, l_P\}$ and the corresponding target geometry consists of the points $\{\hat{\mu}_i'\}_{i=1}^{P}$. We want to find a mapping function $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ such that each original source point can be mapped to its corresponding target point. The problem of determining the transformation function f can be

formulated as an optimization problem:

$$E(f) = \sum_{i=1}^{P} \left(\hat{\mu}_i' - f(\hat{\mu}_{l_i})\right)^\top \hat{\Sigma}_{l_i}^{-1} \left(\hat{\mu}_i' - f(\hat{\mu}_{l_i})\right) + \lambda \|f\|_{\text{tps}}^2, \tag{12.1}$$

where $E(\cdot)$ represents the bending energy, $\lambda > 0$ is a smoothing parameter that controls the tradeoff between smoothness and goodness-of-fit, and $\|f\|_{\text{tps}}^2$ denotes the thin plate spline regularizer term. The thin plate spline regularizer is used as smoothness constraints in order to control the behavior of the mapping. It is defined as the space integral of the square of the second order derivatives of the mapping function (Wahba, 1990):

$$\|f\|_{\text{tps}}^2 = \int_{\mathbb{R}^3} \|D^2 f\|_{\text{Frob}}^2 d\mathbf{x}, \tag{12.2}$$

where the matrix $D^2 f$ represents the second partial derivatives of f.

Remarkably, there exists a unique minimizer and the analytical solution to f composes an affine or 'rigid' part and a nonlinear or 'nonrigid' part:

$$f(\hat{\mu}_{l_i}) = \underbrace{\mathbf{B}\hat{\mu}_{l_i}}_{\text{Affine part}} + \underbrace{\omega \mathbf{g}(\hat{\mu}_{l_i})}_{\text{Nonlinear part}}, \tag{12.3}$$

where $\mathbf{B} \in \mathbb{R}^{(D+1)\times(D+1)}$ represents the affine transformation and $\omega \in \mathbb{R}^{(D+1)\times K}$ called warping coefficient represents the non-affine deformation. The vector $\mathbf{g}(\hat{\mu}_{l_i}) \in \mathbb{R}^K$ is constructed by the TPS kernel. Each entry is defined as $\mathbf{g}_j(\hat{\mu}_{l_i}) = \|\hat{\mu}_{l_j} - \hat{\mu}_{l_i}\|^2$. It should be noted that we update each point with an intercept term, i.e. $\hat{\mu}_{l_i} \leftarrow [\hat{\mu}_{l_i}^\top 1]^\top$ in order to include data offset conveniently. The analytic expression for $\mathbf{B}$ and ω is provided as follows.

First we concatenate the probabilistic landmark trajectory with $\widetilde{\mu}_l = [\hat{\mu}_{l_1}, \ldots, \hat{\mu}_{l_P}]^\top$, $\widetilde{\mu}' = [\hat{\mu}_1', \ldots, \hat{\mu}_P']$, and $\widetilde{\mathbf{Q}} = [\text{diag}(\hat{\Sigma}_{l_1}^{-1}), \ldots, \text{diag}(\hat{\Sigma}_{l_P}^{-1})]^\top$. By substituting the

solution (12.3) into (12.1), the bending energy function now becomes

$$\begin{aligned} E &= \mathrm{sum}\big((\widetilde{\mu}' - \mathbf{G}\omega^\top - \widetilde{\mu}_l\mathbf{B}^\top) \circ \widetilde{\mathbf{Q}} \circ (\widetilde{\mu}' - \mathbf{G}\omega^\top - \widetilde{\mu}_l\mathbf{B}^\top)\big) \\ &\quad + \lambda \mathrm{tr}(\omega\mathbf{G}\omega^\top) \\ &= \|\mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ (\widetilde{\mu}' - \mathbf{G}\omega^\top - \widetilde{\mu}_l\mathbf{B}^\top)\|^2_{\mathrm{Frob}} + \lambda \mathrm{tr}(\omega\mathbf{G}\omega^\top) \end{aligned} \tag{12.4}$$

In view of the optimization objective, it is not straightforward to calculate $\mathbf{B}$ and ω. Therefore, the QR decomposition is employed to separate the affine and non-affine warping space (Wahba, 1990):

$$\mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ \widetilde{\mu}_l = [\mathbf{Q}_1 \ \mathbf{Q}_2] \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}. \tag{12.5}$$

By observing that $(\mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ \widetilde{\mu}_l)^\top \omega^\top = 0$, we set $\omega^\top = \mathbf{Q}_2\gamma$. Subsequently, the bending energy can be expressed as

$$\begin{aligned} E =& \|\mathbf{Q}_2^\top \big(\mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ (\widetilde{\mu}' - \mathbf{G}\mathbf{Q}_2\gamma)\big)\|^2_{\mathrm{Frob}} \\ &+ \|\mathbf{Q}_1^\top \big(\mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ (\widetilde{\mu}' - \mathbf{G}\mathbf{Q}_2\gamma) - \mathbf{R}\mathbf{B}^\top\big)\|^2_{\mathrm{Frob}} \\ &+ \lambda \mathrm{tr}(\gamma^\top \mathbf{Q}_2^\top \mathbf{G}\mathbf{Q}_2\gamma). \end{aligned} \tag{12.6}$$

Finally, the solution for $\mathbf{B}$ and ω are

$$\begin{aligned} \mathbf{B}^\top &= \mathbf{R}^{-1}\mathbf{Q}_1^\top \big(\mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ (\widetilde{\mu}' - \mathbf{G}\mathbf{Q}_2\gamma)\big) \\ \omega^\top &= \mathbf{Q}_2\big(\mathbf{Q}_2^\top \mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ \mathbf{G}\mathbf{Q}_2 + \lambda_\omega \mathbf{I}\big)^{-1}\mathbf{Q}_2^\top \mathrm{sqrt}(\widetilde{\mathbf{Q}}) \circ \widetilde{\mu}' \end{aligned} \tag{12.7}$$

The evaluation of proposed method is shown in Figure 12.1. The original demonstrated trajectory mean is bend with respect to four control points.

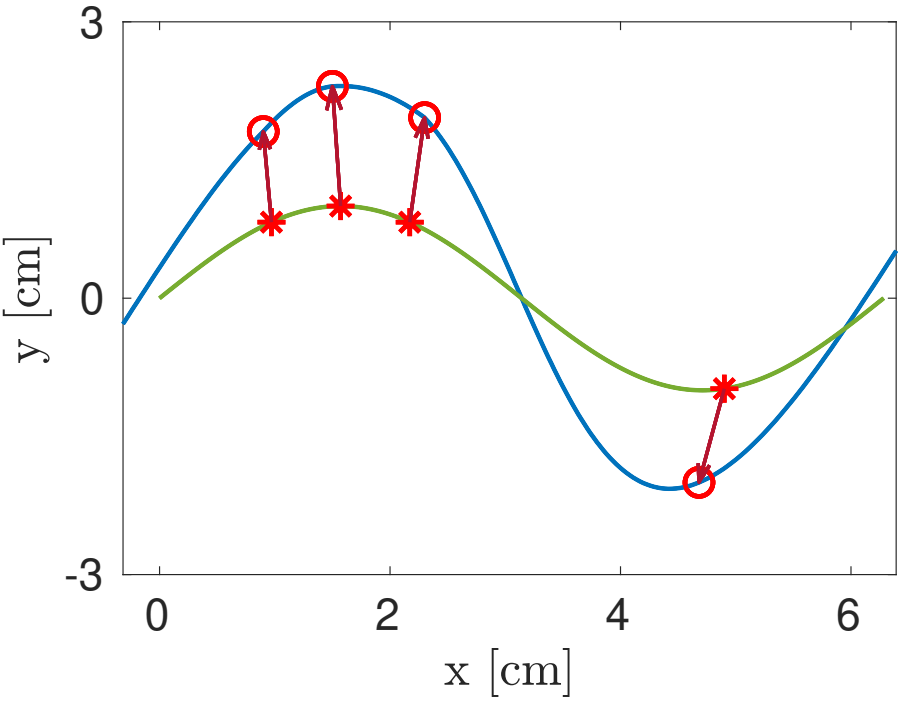


Figure 12.1 Illustration of probabilistic trajectory transfer where the green curve denotes the demonstration trajectory and the blue curve denotes the transferred trajectory with the old and new control points depicted with red stars and circles, respectively. The red arrows point from old control points to the new corresponding ones.

Epilogue

In this thesis, we have presented a structured prediction approach to robot programming by demonstration based on the probabilistic imitation learning framework. At the core of our approach, we construct the f-divergence based loss function between the expert's trajectory policy and the learner's trajectory policy. Such perspective is gradually receiving attention as many imitation learning algorithms have been shown to fall into this scope. Due to the generality of the structured prediction, the proposed method is able to imitate trajectories not only in classic Euclidean space, but also with manifold constraints. Especially, the merit of adapting trajectories with manifold constraints distinguishes our method from other state-of-the-art robot imitation learning algorithms. In the meanwhile, the adapted trajectory is guaranteed to evolve in the same manifold as the demonstration trajectory.

As we develop our approach under the probabilistic imitation learning framework, the procedure of deriving a policy from multiple demonstrations reminds us of the general approach of reinforcement learning, which continuously updates policies based on past trials and their associated rewards (Sutton and Barto, 2018). In this regard, when each demonstration is labeled with human preference (Brown et al., 2019), the derived imitation learning algorithm could resemble one step reinforcement learning. It is worth further investigation to bridge imitation learning and reinforcement learning for underlying consistency. In addition, rather than use f-divergence for probability measures, it could be also interesting to consider other techniques such as Hilbertian metrics resp. positive definite kernels (Hein and Bousquet, 2004) or the the Wasserstein distance (Villani, 2003).

In this thesis, we have shown a few extensions to our approach including trajectory refinement with policy search, constrained imitation learning, and probabilistic trajectory transfer. In addition, we conceive that our approach could be applied in a variety of robotic applications that require human-guided trajectory generation such as humanoid whole-body tele-operation (Darvish et al., 2019) or periodic movements such as bipedal robot locomotion (Romualdi et al., 2018).

References

Abbeel, P. (2012). Lecture notes in inverse reinforcement learning.

Abbeel, P., Coates, A., and Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639.

Absil, P.-A., Mahony, R., and Sepulchre, R. (2009). *Optimization algorithms on matrix manifolds*. Princeton University Press.

Ahmadzadeh, S. R. and Chernova, S. (2018). Trajectory-based skill learning using generalized cylinders. *Frontiers in Robotics and AI*, 5.

Álvarez, M. A., Rosasco, L., and Lawrence, N. D. (2012). Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266.

Amanhoud, W., Khoramshahi, M., and Billard, A. (2019). A dynamical system approach to motion and force generation in contact tasks. Robotics: Science and Systems (RSS).

Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., and Tabuada, P. (2019). Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE.

Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):509–522.

Bertsekas, D. P. et al. (2000). *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot programming by demonstration. *Springer handbook of robotics*, pages 1371–1394.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York.

Bonalli, R., Bylard, A., Cauligi, A., Lew, T., and Pavone, M. (2019). Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach. In *Robotics: Science and Systems*.

Bossi, F., Willemse, C., Cavazza, J., Marchesi, S., Murino, V., and Wykowska, A. (2020). The human brain reveals resting state activity patterns that are predictive of biases in attitudes toward robots. *Science robotics*, 5(46).

Brown, B. J. and Rusinkiewicz, S. (2007). Global non-rigid alignment of 3-d scans. In *ACM Transactions on Graphics (TOG)*, volume 26, page 21. ACM.

Brown, D., Goo, W., Nagarajan, P., and Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, pages 783–792.

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

Calinon, S. (2019). Lecture notes in motion primitives.

Calinon, S. (2020). Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control. *IEEE Robotics and Automation Magazine (RAM)*.

Calinon, S. and Billard, A. (2009). Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 23:2059–2076.

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.

Calinon, S. and Lee, D. (2016). Learning control. *Humanoid Robotics: a Reference*, pages 1–52.

Calinon, S. and Lee, D. (2019). Learning control. In Vadakkepat, P. and Goswami, A., editors, *Humanoid Robotics: a Reference*, pages 1261–1312. Springer.

Calinon, S., Li, Z., Alizadeh, T., Tsagarakis, N. G., and Caldwell, D. G. (2012). Statistical dynamical systems for skills acquisition in humanoids. In *12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 323–329.

Cheng, C.-A., Yan, X., Wagener, N., and Boots, B. (2018). Fast policy learning through imitation and reinforcement. In *Uncertainty in artificial intelligence*.

Ciliberto, C., Rosasco, L., and Rudi, A. (2016). A consistent regularization approach for structured prediction. In *Advances in neural information processing systems*, pages 4412–4420.

Ciliberto, C., Rosasco, L., and Rudi, A. (2020). A general framework for consistent structured prediction with implicit loss embeddings. *arXiv preprint arXiv:2002.05424*.

Ciliberto, C., Rudi, A., Rosasco, L., and Pontil, M. (2017). Consistent multitask learning with nonlinear output relations. In *Advances in Neural Information Processing Systems*, pages 1986–1996.

Coumans, E. and Bai, Y. (2016–2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Darvish, K., Tirupachuri, Y., Romualdi, G., Rapetti, L., Ferigo, D., Chavez, F. J. A., and Pucci, D. (2019). Whole-body geometric retargeting for humanoid robots. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 679–686.

Daumé, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

Duan, A., Camoriano, R., Ferigo, D., Calandriello, D., Rosasco, L., and Pucci, D. (2018). Constrained dmps for feasible skill learning on humanoid robots. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–6. IEEE.

Duan, A., Camoriano, R., Ferigo, D., Huang, Y., Calandriello, D., Rosasco, L., and Pucci, D. (2019). Learning to sequence multiple tasks with competing constraints. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2672–2678. IEEE.

Duchi, J. C., Mackey, L. W., and Jordan, M. I. (2010). On the consistency of ranking algorithms. In *ICML*.

Durrieu, J.-L., Thiran, J.-P., and Kelly, F. (2012). Lower and upper bounds for approximation of the kullback-leibler divergence between gaussian mixture models. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4833–4836. IEEE.

Elobaid, M., Hu, Y., Romualdi, G., Dafarra, S., Babic, J., and Pucci, D. (2019). Telexistence and teleoperation for walking humanoid robots. In *Proceedings of SAI Intelligent Systems Conference*, pages 1106–1121. Springer.

Figueroa, N. and Billard, A. (2018). A physically-consistent bayesian non-parametric mixture model for dynamical system learning. In *CoRL*, pages 927–946.

Gao, W. and Zhou, Z.-H. (2011). On the consistency of multi-label learning. In *Proceedings of the 24th annual conference on learning theory*, pages 341–358.

Ghadami, R., Rahebi, J., and Yayly, Y. (2015). Fast methods for spherical linear interpolation in minkowski space. *Advances in Applied Clifford Algebras*, 25(4):863–873.

Ghasemipour, S. K. S., Zemel, R., and Gu, S. (2019). A divergence minimization perspective on imitation learning methods. *arXiv preprint arXiv:1911.02256*.

Ginesi, M., Sansonetto, N., and Fiorini, P. (2019). DMP++: Overcoming some drawbacks of dynamic movement primitives. *arXiv preprint arXiv:1908.10608*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hein, M. and Bousquet, O. (2004). Hilbertian metrics and positive definite kernels on probability measures.

Hersch, M., Guenter, F., Calinon, S., and Billard, A. (2008). Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467.

Hershey, J. R. and Olsen, P. A. (2007). Approximating the Kullback Leibler divergence between Gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–317. IEEE.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573.

Huang, Y., Rozo, L., Silvério, J., and Caldwell, D. G. (2019). Kernelized movement primitives. *The International Journal of Robotics Research*, 38(7):833–852.

Huang, Y., Silvério, J., and Caldwell, D. G. (2018a). Towards minimal intervention control with competing constraints. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 733–738. IEEE.

Huang, Y., Silvério, J., Rozo, L., and Caldwell, D. G. (2018b). Hybrid probabilistic trajectory optimization using null-space exploration. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, pages 7226–7232. IEEE.

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26).

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373.

Ke, L., Barnes, M., Sun, W., Lee, G., Choudhury, S., and Srinivasa, S. (2019). Imitation learning as f-divergence minimization. *arXiv preprint arXiv:1905.12888*.

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.

Khoramshahi, M., Henriks, G., Naef, A., Salehian, S. S. M., Kim, J., and Billard, A. (2020). Arm-hand motion-force coordination for physical interactions with non-flat surfaces using dynamical systems: Toward compliant robotic massage. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE.

Kingston, Z., Moll, M., and Kavraki, L. E. (2019). Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research*, 38(10-11):1151–1178.

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.

Kronander, K. and Billard, A. (2015). Passive interaction control with dynamical systems. *IEEE Robotics and Automation Letters*, 1(1):106–113.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Kulvicius, T., Ning, K., Tamosiunaite, M., and Worgötter, F. (2011). Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics*, 28(1):145–157.

Kumar, P., Verma, J., and Prasad, S. (2012). Hand data glove: a wearable real-time device for human-computer interaction.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations.

Maintz, J. A. and Viergever, M. A. (1996). An overview of medical image registration methods. In *In Symposium of the Belgian hospital physicists association (SBPH-BVZF*.

Marey, M. and Chaumette, F. (2010). New strategies for avoiding robot joint limits: Application to visual servoing using a large projection operator. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6222–6227. IEEE.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1):8.

Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56.

Mroueh, Y., Poggio, T., Rosasco, L., and Slotine, J.-J. (2012). Multiclass learning with simplex coding. In *Advances in Neural Information Processing Systems*, pages 2789–2797.

Natale, L., Bartolozzi, C., Pucci, D., Wykowska, A., and Metta, G. (2017). iCub: The not-yet-finished story of building a robot child. *Science Robotics*, 2(13):eaaq1026.

Nava, G. (2020). *Instantaneous Momentum-Based Control of Floating Base Systems*. PhD thesis, University of Genova.

Nehaniv, C. L., Dautenhahn, K., and Dautenhahn, K. (2002). *Imitation in animals and artifacts*. MIT press.

Nori, F., Traversaro, S., Eljaik, J., Romano, F., Del Prete, A., and Pucci, D. (2015). iCub whole-body control through force regulation on rigid non-coplanar contacts. *Frontiers in Robotics and AI*, 2:6.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179.

Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. (2013). Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624.

Pardo, L. (2018). *Statistical inference based on divergence measures*. Chapman and Hall/CRC.

Pasquale, G., Ciliberto, C., Odone, F., Rosasco, L., and Natale, L. (2019). Are we done with object recognition? the icub robot's perspective. *Robotics and Autonomous Systems*, 112:260–281.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143.

Peng, X. B., Coumans, E., Zhang, T., Lee, T.-W., Tan, J., and Levine, S. (2020). Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*.

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer.

Ratliff, N. D., Issac, J., Kappler, D., Birchfield, S., and Fox, D. (2018). Riemannian motion policies. *arXiv preprint arXiv:1801.02854*.

Ravichandar, H., Polydoros, A. S., Chernova, S., and Billard, A. (2020). Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3.

Rea, F., Metta, G., and Bartolozzi, C. (2013). Event-driven visual attention for the humanoid robot icub. *Frontiers in neuroscience*, 7:234.

Reiner, B., Ertel, W., Posenauer, H., and Schneider, M. (2014). LAT: A simple learning from demonstration method. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4436–4441. IEEE.

Romano, F., Nava, G., Azad, M., Čamernik, J., Dafarra, S., Dermy, O., Latella, C., Lazzaroni, M., Lober, R., Lorenzini, M., et al. (2017). The codyco project achievements and beyond: Toward human aware whole-body controllers for physical human robot interaction. *IEEE Robotics and Automation Letters*, 3(1):516–523.

Romualdi, G., Dafarra, S., Hu, Y., and Pucci, D. (2018). A benchmarking of dcm based architectures for position and velocity controlled walking of humanoid robots. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE.

Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668.

Rudi, A., Ciliberto, C., Marconi, G., and Rosasco, L. (2018). Manifold structured prediction. In *Advances in Neural Information Processing Systems*, pages 5610–5621.

San Juan, I. I., Sloth, C., Kramberger, A., Petersen, H. G., Østergård, E. H., and Savarimuthu, T. R. (2020). Towards reversible dynamic movement primitives. In *IEEE/RSJ International Conference on Intelligent Robots and SystemsIEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5063–5070. IEEE.

Sason, I. (2018). On f-divergences: Integral representations, local behavior, and inequalities. *Entropy*, 20(5):383.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242.

Schneider, M. and Ertel, W. (2010). Robot learning by demonstration with local Gaussian process regression. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 255–260. IEEE.

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., and Abbeel, P. (2014). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270.

Schulman, J., Ho, J., Lee, C., and Abbeel, P. (2016). Learning from demonstrations through the use of non-rigid registration. In *Robotics Research*, pages 339–354. Springer.

Simo-Serra, E., Torras, C., and Moreno-Noguer, F. (2017). 3D human pose tracking priors using geodesic mixture models. *International Journal of Computer Vision*, 122(2):388–408.

Stulp, F. and Sigaud, O. (2013). Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Ti, B., Gao, Y., Li, Q., and Zhao, J. (2019). Dynamic movement primitives for movement generation using gmm-gmr analytical method. In *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, pages 250–254. IEEE.

Villani, C. (2003). *Topics in optimal transportation*. Number 58. American Mathematical Soc.

Wahba, G. (1990). *Spline models for observational data*, volume 59. Siam.

Welling, M. (2002). Lecture notes in kernel ridge regression.

Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and Panne, M. (2020). Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pages 317–329. PMLR.

You, S. and Robert Jr, L. P. (2018). Human-robot similarity and willingness to work with a robotic co-worker. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 251–260.

Zeestraten, M. J., Calinon, S., and Caldwell, D. G. (2016). Variable duration movement encoding with minimal intervention control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 497–503. IEEE.

Zeestraten, M. J., Havoutis, I., Calinon, S., and Caldwell, D. G. (2017a). Learning task-space synergies using riemannian geometry. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 73–78. IEEE.

Zeestraten, M. J., Havoutis, I., Silvério, J., Calinon, S., and Caldwell, D. G. (2017b). An approach for imitation learning on Riemannian manifolds. *IEEE Robotics and Automation Letters*, 2(3):1240–1247.

Chapter 13

Appendix

13.1 Proof Sketch for SELF

Here we provide sketch proof that the loss functions used in the paper are SELF. The basic idea is to illustrate the loss function can be formulated to conform to 4.3.

KL divergence

Consider the following formulation for the loss function when making mean prediction:

$$
\begin{aligned}
&(\mu-\mu_n)^\top\Sigma^{-1}(\mu-\mu_n) \\
&=\underbrace{\begin{bmatrix}\mu^\top\Sigma^{-1}\mu\\ \mu \\ 1\end{bmatrix}}_{\Psi(\mu)}^\top \underbrace{\begin{matrix}\text{blkadiag}(1,\\ -2\Sigma^{-1},1)\end{matrix}}_{V} \underbrace{\begin{bmatrix}\mu_n^\top\Sigma^{-1}\mu_n\\ \mu_n \\ 1\end{bmatrix}}_{\Psi(\mu_n)},
\end{aligned}
\tag{13.1}
$$

where blkadiag$(\cdot)$ denotes the block anti-diagonal matrix.

Consider the following formulation for the loss function when making covariance prediction:

$$
\begin{aligned}
&(\mu-\mu_n)^\top\Sigma^{-1}(\mu-\mu_n)+\log|\Sigma|+\mathtt{Tr}(\Sigma^{-1}\Sigma_n)\\
&=\underbrace{\begin{bmatrix}\mu^\top\Sigma^{-1}\mu\\ \Sigma^{-1}\mu\\ \phi_2(\Sigma^{-\frac{1}{2}})\\ \log|\Sigma|\\ \mathtt{vec}(\Sigma^{-1})\\ \mathtt{vec}(\Sigma)\\ 1\\ \phi_1(\mu)\\ -2\mu\\ 1\end{bmatrix}}_{\Psi(\Sigma)}
\underbrace{\begin{bmatrix}\multicolumn{2}{c}{\multirow{2}{*}{$\mathbf{0}$}} & \text{blkadiag}(1,\\ & & \mathbf{I}_d,\mathbf{I}_q,\\ & & 1,\mathbf{I}_{d^2})\\ \hdashline \mathbf{0} & & \mathbf{0}\end{bmatrix}}_{V}
\underbrace{\begin{bmatrix}\mu_n^\top\Sigma_n^{-1}\mu_n\\ \Sigma_n^{-1}\mu_n\\ \phi_2(\Sigma_n^{-\frac{1}{2}})\\ \log|\Sigma_n|\\ \mathtt{vec}(\Sigma_n^{-1})\\ \mathtt{vec}(\Sigma_n)\\ 1\\ \phi_1(\mu_n)\\ -2\mu_n\\ 1\end{bmatrix}}_{\Psi(\Sigma_n)},
\end{aligned}
\tag{13.2}
$$

where $\mathbf{0}$ is the zeros matrix with proper dimension. The features ϕ_1 and ϕ_2 are designed such that $\phi_2(\Sigma_n^{-\frac{1}{2}})\phi_1(\mu_n)=\mu_n\Sigma^{-1}\mu_n$ and $\mathtt{vec}(\cdot)$ is the vectorization operation for a matrix.

Reverse KL divergence

Similarly, the loss functions used in the reverse KL divergence can be structured as SELF. For loss function of mean, consider the following formulation:

$$\begin{aligned}
&(\mu-\mu_n)^\top\Sigma_n^{-1}(\mu-\mu_n)\\
&=\underbrace{\begin{bmatrix}\mu^\top\Sigma^{-1}\mu\\ -2\mu\\ \phi_1(\mu)\\ \phi_2(\Sigma^{-\frac{1}{2}})\\ \Sigma^{-1}\mu\\ 1\end{bmatrix}^\top}_{\Psi(\mu)}\underbrace{\left[\begin{array}{c:c}\mathbf{0} & \begin{matrix}\text{blkadiag}(1,\\ \mathbf{I}_d,\mathbf{I}_q)\end{matrix}\\ \hdashline \mathbf{0} & \mathbf{0}\end{array}\right]}_{V}\underbrace{\begin{bmatrix}\mu_n^\top\Sigma_n^{-1}\mu_n\\ -2\mu_n\\ \phi_1(\mu_n)\\ \phi_2(\Sigma_n^{-\frac{1}{2}})\\ \Sigma_n^{-1}\mu_n\\ 1\end{bmatrix}}_{\Psi(\mu_n)}
\end{aligned}\tag{13.3}$$

For covariance prediction, consider the following formulation for the loss function:

$$\begin{aligned}
&-\log|\Sigma|+\texttt{Tr}(\Sigma_n^{-1}\Sigma)\\
&=\underbrace{\begin{bmatrix}\log|\Sigma|\\ \texttt{vec}(\Sigma)\\ \texttt{vec}(\Sigma^{-1})\\ 1\end{bmatrix}^\top}_{\Psi(\Sigma)}\underbrace{\left[\begin{array}{c:c}\mathbf{0} & \begin{matrix}\text{blkadiag}(\\ -1,\mathbf{I}_{d^2})\end{matrix}\\ \hdashline \mathbf{0} & \mathbf{0}\end{array}\right]}_{V}\underbrace{\begin{bmatrix}\log|\Sigma_n|\\ \texttt{vec}(\Sigma_n)\\ \texttt{vec}(\Sigma_n^{-1})\\ 1\end{bmatrix}}_{\Psi(\Sigma_n)}
\end{aligned}\tag{13.4}$$