UNIVERSITA' DEGLI STUDI DI VERONA

DEPARTMENT OF

Computer Science

GRADUATE SCHOOL OF

Natural Sciences and Engineering

DOCTORAL PROGRAM IN

Computer Science

XXXIII cycle (2017)

Learning of Surgical Gestures for Robotic Minimally Invasive Surgery Using Dynamic Movement Primitives and Latent Variable Models

S.S.D. INF/01

Coordinator: Prof. Paolo Fiorini

Signature _____

Doctoral Student: Dott. Michele Ginesi

Signature _____

Learning of Surgical Gestures for Minimally Invasive Surgery using Dynamic Movement Primitives and Latent Variable Models

Michele Ginesi
PhD thesis
Verona, 30 September 2021

Università degli Studi di Verona

Department of Computer Science

Doctoral thesis

Learning of Surgical Gestures for Robotic Minimally Invasive Surgery using Dynamic Movement Primitives and Latent Variable Models

Candidate:
Michele Ginesi

Thesis advisor:
Prof. Paolo Fiorini

XXXIII cycle

Abstract

Full and partial automation of Robotic Minimally Invasive Surgery holds significant promise to improve patient treatment, reduce recovery time, and reduce the fatigue of the surgeons. However, to accomplish this ambitious goal, a mathematical model of the intervention is needed.

In this thesis, we propose to use Dynamic Movement Primitives (DMPs) to encode the *gestures* a surgeon has to perform to achieve a task. DMPs allow to learn a trajectory, thus imitating the dexterity of the surgeon, and to execute it while allowing to generalize it both spatially (to new starting and goal positions) and temporally (to different speeds of executions). Moreover, they have other desirable properties that make them well suited for surgical applications, such as online adaptability, robustness to perturbations, and the possibility to implement obstacle avoidance. We propose various modifications to improve the state-of-the-art of the framework, as well as novel methods to handle obstacles. Moreover, we validate the usage of DMPs to model gestures by automating a surgical-related task and using DMPs as the low-level trajectory generator.

In the second part of the thesis, we introduce the problem of unsupervised segmentation of tasks' execution in gestures. We will introduce latent variable models to tackle the problem, proposing further developments to combine such models with the DMP theory. We will review the *Auto-Regressive Hidden Markov Model* (AR-HMM) and test it on surgical-related datasets. Then, we will propose a generalization of the AR-HMM to general, non-linear, dynamics, showing that this results in a more accurate segmentation, with a less severe over-segmentation. Finally, we propose a further generalization of the AR-HMM that aims at integrating a DMP-like dynamic into the latent variable model.

ACKNOWLEDGMENTS

Many people supported me during my Ph.D. Their help was fundamental to reach this goal.
A big thank goes to my advisor, prof. Paolo Fiorini, for his support. During these three years, he always trusted me and my ideas while providing numerous hints and suggestions. This allowed me to carry out my research in the best way I could imagine.
My regards to my colleagues at the ALTAIR Robotics Laboratory. They provided an environment in which the sharing of opinions and experiences never lacked. Many of the ideas contained in this thesis were born in front of a whiteboard while discussing with them.
I wish to thank my parents and sister, who always supported my ambitions and never doubted my capabilities.
Finally, I send my greatest regards to my wife Sara, who always believed in me and was always there to help me face any challenge. Her infinite support and patience allowed me to overcome any obstacle I had to face during my Ph.D.

Published Content and Contributions

[43] M. Ginesi, D. Meli, A. Calanca, D. Dall'Alba, N. Sansonetto, and P. Fiorini. Dynamic movement primitives: Volumetric obstacle avoidance. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 234–239, Dec 2019

[44] M. Ginesi, D. Meli, H. C. Nakawala, A. Roberti, and P. Fiorini. A knowledge-based framework for task automation in surgery. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 37–42, Dec 2019

[45] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, and P. Fiorini. Autonomous task planning and situation awareness in robotic surgery. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3144–3150, 2020

[46] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, and P. Fiorini. Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions. *Journal of Intelligent & Robotic Systems*, 101(4):79, Mar 2021

[47] M. Ginesi, N. Sansonetto, and P. Fiorini. Overcoming some drawbacks of dynamic movement primitives. *Robotics and Autonomous Systems*, 144:103844, 2021

Contents

List of Figures

List of Tables

List of Algorithms

Mathematical Notation

Scalars	a, α	Lowerase letters
Sets	A, Γ	Uppercase letters
Vectors	$\mathbf{v}, \boldsymbol{\omega}$	Lowercase bold letters
Matrices	$\mathbf{A}, \boldsymbol{\Gamma}$	Uppercase bold letters
$d \times d$ Identity matrix	$\mathbf{I}\mathrm{d}_d$	
Transposed of a vector $\mathbf{v}$ or of a matrix $\mathbf{A}$	$\mathbf{v}^\top$, $\mathbf{A}^\top$	
Scalar product between vectors $\mathbf{v}$ and $\mathbf{u}$	$\langle \mathbf{v}, \mathbf{u} \rangle$, $\mathbf{v} \cdot \mathbf{u}$	
Vector (cross) product between vectors $\mathbf{v}$ and $\mathbf{u}$ in $\mathbb{R}^2$ or $\mathbb{R}^3$	$\mathbf{v} \times \mathbf{u}$	
Determinant of matrix $\mathbf{A}$	$\det(\mathbf{A})$	
Kroneker product between matrices $\mathbf{A}$ and $\mathbf{B}$	$\mathbf{A} \otimes \mathbf{B}$	
Component-wise product between vectors	$\mathbf{v} \odot \mathbf{w}$	
Normalized vector	$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$	

I

INTRODUCTION

CHAPTER 1

INTRODUCTION AND THESIS OUTLINE

Minimally Invasive Surgery (MIS) has revolutionized surgical care, significantly reducing postoperative pain, recovery time, and hospital stays with marked improvements in health and cosmetic outcome, and overall cost-effectiveness [25].
In recent years, MIS has drawn the attention of both surgeons and researchers [92]. An important example is the daVinci surgical robot [50] from Intuitive Surgical Inc. (Sunnyvale, California), which provides a complete telemanipulator system for minimally invasive interventions. The daVinci system includes three components (see Figure 1.1):

- The *surgeon console* is the control center of the system that allows the surgeon to view the surgical field and controls the instruments and endoscope movements;
- The *patient-side cart* with three articulated mechanical arms that support the surgical instrument arms and camera during surgical procedures;
- The *electronic cart* contains supporting hardware and software components.

The introduction of robots in MIS has further improved the precision of gestures and the recovery time for patients [84, 24, 130]. Moreover, the surgeon's fatigue is greatly reduced. However, these advantages come at the price of reduced immersiveness for the surgeon, since the surgeon can not feel any force exerted onto the situs [92], even though methods to incorporate haptic feedbacks in the robot have been studied [93].

At present, Robotic MIS is performed only in a tele-operative fashion, i.e. the surgeon controls the patient-side robot via the master console. One of the long-term goals of surgical robotic research is the development of a cognitive robotic

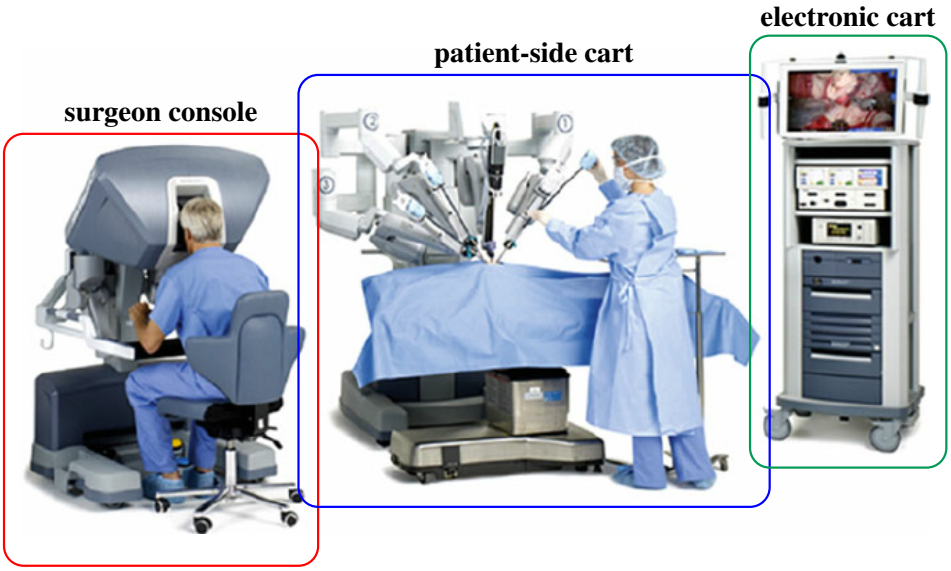


Figure 1.1: The daVinci surgical robot with its three major components highlighted.

system able to automatically execute an operation, or, at least, a part of it [18, 96]. Indeed, increasing the level of surgical procedure automation could further improve the quality of an intervention, in terms of safety and recovery time for the patient while allowing to optimize the use of the operating room, reducing both surgeons' fatigue and hospital costs [140].

The challenges towards autonomous robotic surgery have been investigated also in Artificial Intelligence (AI) research and they include situation awareness, scene understanding to monitor and adapt the surgical workflow in real-time, explainable plan generation for safety, dexterous trajectory generation, and adaptation even in small workspaces.

In this thesis, we discuss how to model surgical *gestures*, that is all those movements that a surgeon has to perform in order to achieve a task. For instance, a gesture is 'performing a loop' during a suturing task.

A surgical procedure can be subdivided at different *granularity* or *hierarchical* levels [85, 129]. This hierarchical decomposition could be useful to structure the different interactions between the surgical team and the new technologies (e.g., communication, surgical activities) [28]. Usually, a procedure is decomposed into steps, sub-steps, task, and sub-tasks [85], see Figure 1.2 for a scheme. Each sub-task can be thought of as a gesture. For instance, the *suture* is a task, whose sub-tasks are *position jaws*, *bite tissue*, *pull needle through*, etc. Some gesture can be further subdivided into smaller components, called *primitives* or *surgemes*. The granularity level defines the level of abstraction at which the surgical procedure is described. Levels that are higher in the hierarchy are the most abstract (for instance 'prepare

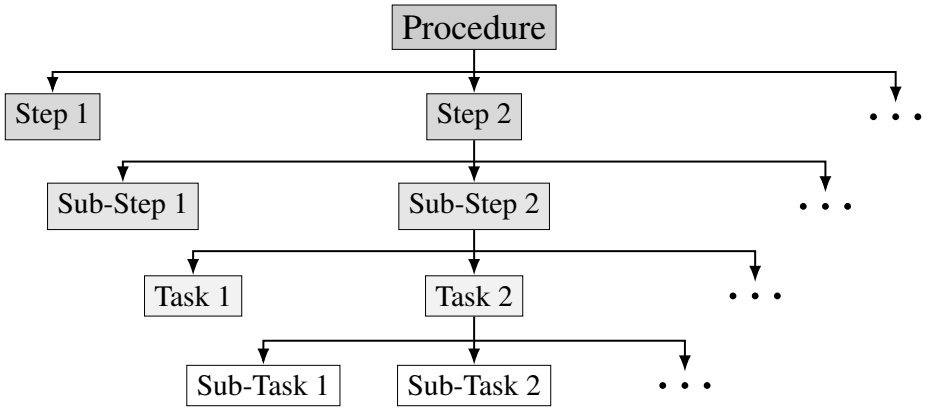


Figure 1.2: Example of hierarchical subdivision of a surgical procedure

the patient') while lower levels have a lower abstraction level (for instance 'push the needle').

Due to these different abstraction levels, we have that automation of surgical procedures must be treated from two directions. In particular, the higher level of abstraction can be implemented in a top-down fashion by explicitly dividing the operation into its steps. On the other hand, modeling gestures and surgemes requires a bottom-up approach. Indeed, the different surgemes have to be learned and later combined into sub-tasks. Robot trajectory planning can easily be done using splines [19, 86, 27, 81] or potentials [114]. These methods work well when an exact representation of the environment is available, and when the aim is a type of optimization (e.g. minimization of the execution time, of the energy consumption, etc.). However, in surgical domains, these assumptions are no longer true: the environment is stochastic and dynamic, and a trajectory has the unique goal to accomplish a task. Thus, in this case, a *Learning from Demonstration* (LfD) approach in which the learned trajectory can be generalized to new environments is preferable. Two main aspects are being studied: the definition of a kinematical model from observations [92, 98, 100]; and the automatic segmentation of surgical tasks into gestures [69, 97, 36, 28].

In this thesis, we present a full treatment of the former aspect and present some preliminary studies for the latter.

In the last years, the growing attention to Machine Learning (ML) and in particular to Deep Learning (DL) led various teams to the application of these techniques to multiple robotic tasks, including, for example, door opening [139] and grasping [105, 78, 64, 11].

While being empirically proven to handle good results, these methods have some shortcomings that make them non-suitable for surgical applications. Firstly, most of the methods are of Reinforcement Learning (RL) type, which means that the robot

learns how to accomplish a task with a trial-and-error strategy, while in medical applications the goal is to extract a model by observing expert surgeons perform the task. Secondly, these methods usually require an enormous quantity of data to being trained. For instance, to learn a grasping task using a Neural Network, in [105] a database with fifty thousand trials was used, while in [78] eight hundred thousand grasps attempts were collected. Unfortunately, it is difficult to obtain data from the executions of real interventions. Thus, for our purposes, there is the need for a method to learn a trajectory with few (or even a single) demonstrations, but which can also be generalized to new situations.

In Part II of the thesis, we discuss the definition of the surgical gestures kinematical model. We will present the Dynamic Movement Primitives framework as a modeling strategy for the automatic execution of movements. Dynamic Movement Primitives (DMPs) [61, 62, 119, 120, 60, 103, 54] is a well-known and widely used framework for trajectory learning and control in robotics. Three main advantages make DMPs well suited for Robotic MIS.

Firstly, a DMP can be learned from a single demonstration. This allows the framework to be used in fields where there is a lack of data. Secondly, the learned behavior can be generalized both spatially and temporally. To be more precise, the DMP will always converge to any desired goal position, while maintaining a trajectory of similar shape to the learned one. Moreover, by modifying a single scalar parameter, it is possible to change the speed of execution of the DMP. Finally, obstacle avoidance can be implemented within the DMP framework. This is particularly useful in medical scenarios. Indeed, the anatomy of each patient is different. Thus, the framework should be able to adapt to unsees scenarios. In particular, the robot's end-effector must not collide with organs and blood vessels, to avoid harming the patient.

We firstly present the DMP framework, focusing on all the aspects that make them useful in Robotic MIS. Then, we validate the usage of DMPs in surgical-related scenarios. In particular, we propose two automation frameworks, both of which use DMPs as the low-level controller, to automate a Peg & Ring task. This task is widely used to train surgeons since it presents several challenges in common with real surgery, such as grasping and transferring [44].

Part II is structured as follows. In Chapter 2 we present the DMP formulation in Cartesian space. This formulation can be used both to control the robot's end-effector position (as we will do) or to control the robot in joint space. In Chapter 3 we present various obstacle avoidance methods for DMPs. This is a crucial aspect of surgical task automation. Indeed, colliding with the environment in surgical scenarios could harm the patient. In Chapter 4 we present the DMP formulation in unit quaternion space. This formulation allows learning and controlling the orientation

of the robot's end-effector. Multiple gestures in MIS require precise wrist movement, such as in a knot-tying task. In Chapter 5 we introduce a modification of the DMP framework to automatically adapt the trajectory execution to the joint limits of the robots. In Chapter 6 we present two frameworks to automate surgical-related tasks. Both frameworks are composed of a high-level task reasoner which decides which gesture should be performed to accomplish the task. DMPs are used as the low-level controller, to prove the effectiveness of using DMPs to learn gestures and surgemes in surgical scenarios.

In order to facilitate the automation of robotic surgery, detailed and exhaustive comprehension of the surgical procedure is needed. A key aspect in this regard is to develop techniques that segment and recognize surgical tasks. Many segmentation methods, both supervised and unsupervised, have been proposed in the literature to segment surgical procedures, as well as other robotic operations, in tasks. This methods comprehend *Transition State Clustering* [97, 70], *Skip-Chain Conditional Random Field* [75], *Soft-Boundary approach* [36], and *Convolutional Neural Networks* [74, 76]. Segmentation in sub-tasks is often performed using primitive libraries that can be given a priori [94], or be extracted from probabilistic segmentation [88, 71, 82, 83] or by detecting dynamics switches and repetitions [21, 3].
In Part III of the thesis, we discuss our preliminary work in the context of unsupervised trajectory segmentation. Our proposed approach can be classified as a probabilistic segmentation one. In particular, we rely on the theory of *Latent Variable models*. This family of models assumes that the observed behavior (in our case, the executed trajectory) evolution is determined by a latent (or hidden, or un-observed) variable. In the unsupervised segmentation problem, such a latent variable tells us which gesture is currently being executed.
In Chapter 7 we present the widely known Auto-Regressive Hidden Markov Model (AR-HMM). This model is composed of a finite set of latent variables, related by a Markov chain process, and of a continuous set of observations. The model assumes that the latent variable describes the linear auto-regressive dynamics that model the evolution of the observed continuous state. The main advantage of this model lies in two aspects. First, the model is fully determined by a finite set of parameters: the initial mode probability, the transition probabilities describing the Markov chain, the parameters of the linear dynamics, and the covariance matrix of the observation noise. Moreover, these parameters can be computed directly from the observed data stream via maximum likelihood estimation. Second, given the model parameters and an observed sequence, it is possible to extract the most probable sequence of latent variables that generated the observed trajectory, thus providing a segmentation of the trajectory itself.
In Chapter 8 we propose two ways to generalize the AR-HMM. In particular, we

present a way to extend the auto-regressive dynamics to be not limited to a linear case. To do this, we show how generic non-linear dynamics can be expressed as a linear combination of non-linear basis functions. In this way, the learning algorithm of classical AR-HMM can be easily extended to handle this more general family of models. Moreover, we will introduce a model able to combine the AR-HMM model with a DMP based dynamic. This last part is currently a work in progress. In particular, while analogies between DMP-HMM and classical AR-HMM will be shown, the learning procedures are still incomplete. Future works comprehend the full development and testing of this new model.

In Part IV of the thesis, we present the conclusions of the thesis.

Finally, in Part V, we present the Appendices, in which the mathematical background is presented.

II
Dynamic Movement Primitives

CHAPTER 2

Dynamic Movement Primitives in Cartesian Coordinates

Dynamic Movement Primitives (DMPs) [61, 62, 119, 120, 60] is a framework for trajectory learning based on a second-order Ordinary Differential Equation (ODE) of spring-mass-damper type in which a *forcing* (also called *perturbation*) term is "learned" in such a way to encode the shape of the trajectory of the solution.

During the *learning phase* a trajectory (or a set of trajectories) is recorded, and the forcing term is modeled so that the learned behavior resembles as close as possible the observation(s). Then, during the *execution phase*, the learned forcing term is used to generate new trajectories, allowing to change initial and final position, and speed of execution, while maintaining a trajectory of similar shape to the learned one, and guaranteeing convergence to the goal position.

DMPs have been proven effective in many robotic scenarios, such as cloth manufacturing [66], reproduction of human walk for exoskeletons [56], and collaborative bimanual tasks [38].

We decided to use DMPs to encode surgical gestures because they are able to accurately imitate and replicate human movements, which is essential to encode the surgeon's dexterity.

This Chapter is structured as follows. In Section 2.1 we introduce the original DMP formulation from [61], highlighting its strengths and drawbacks. In Section 2.2 we present the modified DMP formulation from [103], which solves some of the drawbacks of the original one. In Section 2.3 we describe the learning process that allows encoding a desired behavior within the DMP system. In Section 2.4 we present the execution phase that allows executing the learned behavior. In Sec-

tion 2.5 we define and discuss different sets of basis functions that can be used to encode the desired behavior, discussing their numerical properties. In Section 2.6 we describe our novel formulation that allows for a robust adaptation of the DMP to any desired goal configuration. In Section 2.7 we present our proposed algorithm to learn a unique DMP from a set of multiple observations. Finally, in Section 2.8 we present the conclusions.

2.1. Original Formulation

DMPs [61, 62, 119, 120, 60] are used to model both rhythmic and discrete (or stroke-based) movements.
For Minimally Invasive Surgery (MIS), only the discrete formulation is useful. Indeed, even movements that seem rhythmic, such as performing multiple suturing loops, are actually a single gesture that has to be repeated a finite amount of times in slightly different locations. Thus, it is convenient to model these seemingly rhythmic gestures as multiple iterations of discrete movements.

DMPs consist of a system of second-order ODEs, one for each dimension of the ambient space, of mass-spring-damper type with a forcing term.
The framework aims at modeling the forcing term in such a way to be able to generalize the trajectory to new start and goal positions while maintaining the shape of the learned trajectory.

The one-dimensional formulation of DMPs is [61, 62, 119]

$$\begin{cases} \tau \dot{v} = K(g - x) - Dv + (g - x_0) f(s) & \text{(2.1a)} \\ \tau \dot{x} = v & \text{(2.1b)} \end{cases}$$

where $x, v \in \mathbb{R}$ are respectively position and velocity of a prescribed point of the system.
Scalar $\tau \in \mathbb{R}^+$ is a temporal scaling factor that can be used to change the speed of execution of the system. Doubling the value of τ halves the speed of execution, thus doubling the duration.
Scalar $s \in (0, 1]$ is a re-parametrization of time $t \in [0, T]$, whose evolution is described by the so-called *canonical system*:

$$\tau \dot{s} = -\alpha s, \tag{2.2}$$

where $\alpha \in \mathbb{R}_+$ is a parameter controlling the speed of decay of s. The canonical system's initial condition is $s(0) = 1$. The importance of the canonical system (2.2) relies in making the dynamical system (2.1) with (2.2) an autonomous system, i.e. non directly dependent on time t. This allows to speed up or slow down the evolution of the system without any difficulty by simply changing the value of temporal scaling factor τ.

Scalar $x_0 \in \mathbb{R}$ is the initial position, and $g \in \mathbb{R}$ is the *goal* position.
Constants $K, D \in \mathbb{R}_+$ are, respectively, the spring and damping terms. They are chosen in such a way that the associated homogeneous system is critically damped: $D = 2\sqrt{K}$.
Function $f : \mathbb{R} \to \mathbb{R}$ is a real-valued, non-linear *forcing* (also called *perturbation*) *term*. It is defined in terms of basis functions as

$$f(s) = \frac{\sum_{i=0}^{N} \omega_i \psi_i(s)}{\sum_{i=0}^{N} \psi_i(s)} s, \tag{2.3}$$

where functions

$$\psi_i(s) = \exp\left(-h_i(s - c_i)^2\right), \quad i = 0, 1, \ldots, N \tag{2.4}$$

are *Gaussian Radial Basis Functions* (GRBFs) with centers c_i and widths h_i. Centers c_i are defined as

$$c_i = \exp\left(-\alpha \, i \, \frac{T}{N}\right), \quad i = 0, 1, \ldots, N, \tag{2.5}$$

while widths h_i are defined as [122]:

$$\begin{aligned} h_i &= \frac{\tilde{h}}{(c_{i+1} - c_i)^2}, \quad i = 0, 1, \ldots, N-1, \\ h_N &= h_{N-1}, \end{aligned} \tag{2.6}$$

where the value of $\tilde{h} \in \mathbb{R}_+$ determines the overlap between basis functions. This value is usually set to one [127, 2]: $\tilde{h} = 1$. Scalars $\omega_i \in \mathbb{R}$, for $i = 0, 1, \ldots, N$ are called *weights*. The *learning process*, which we will present in details in Section 2.3, focuses on the computation of the weights ω_i that best approximate the desired forcing term, obtained by solving (2.1a) for f.

Figure 2.1 shows an example of such basis functions ψ_i. We remark the following properties:

B1 Being defined as Gaussian function with respect to s, each basis function is symmetric in s, i.e. identity

$$\psi_i(c_i + \varsigma) = \psi_i(c_i - \varsigma)$$

holds true for any value of $\varsigma \in \mathbb{R}$;

B2 The set of centers c_i is equispaced in t. Indeed, since the solution of the canonical system (2.2) is $s(t) = \exp(-\alpha/\tau \, t)$, we have $t = -\tau/\alpha \log(s)$. Thus, definition (2.5) reads, in t,

$$-\frac{\tau}{\alpha} \log\left(\exp\left(-\alpha \, \frac{T}{N} \, i\right)\right) = \frac{T\,\tau}{N} \, i.$$

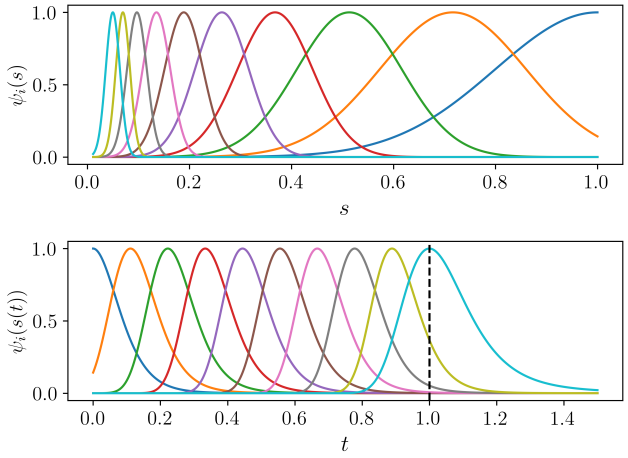


Figure 2.1: Example of Gaussian Basis Functions. The upper plot shows the evolution of the functions w.r.t. the parameter s, while the lower plot shows them as a function of t. The parameters are: $T = 1$, $\alpha = 3$, and $N = 9$. Time $T = 1$ is marked by the dashed black line.

B3 Since s is a decreasing function of t, the order of the basis functions change from s to t: the first (left-most) basis function in t is the last (right-most) in s.

One of the main advantages, and the main reason for their wide usage in robotics, of DMPs (2.1) is that convergence to the goal position is ensured.

Proposition 2.1. *Dynamical system* (2.1) *with* s *governed from* (2.2) *converges to the unique equilibrium* $(x, v) = (g, 0)$.

Proof. Let us firstly prove the result in the case $f(s) \equiv 0$. With this assumption, dynamical system (2.1) reads

$$\begin{cases} \tau \dot{v} = K(g - x) - Dv & \text{(2.7a)} \\ \tau \dot{x} = v & \text{(2.7b)} \end{cases}$$

To compute the equilibria we must solve

$$\begin{cases} K(g - x) - Dv = 0 \\ v = 0 \end{cases}$$

Substituting the second identity into the first, we get

$$K(g - x) = 0,$$

which yields $x = g$. Thus the unique equilibrium is $x = g, v = 0$.
In order to prove the stability of the equilibrium let us re-write the dynamical system (2.7) as

$$\tau^2 \ddot{x} + \tau D \dot{x} - K(g - x) = 0. \quad \text{(2.8)}$$

Now, let us substitute $z = x - g$. By assuming that g is fixed, we have that $\dot{z} = \dot{x}$ and $\ddot{z} = \ddot{x}$. Thus, (2.8) reads

$$\tau^2 \ddot{z} + \tau D \dot{z} + Kz = 0.$$

Using the eigenvalues method we get that the general solution is:

$$z = C_1 \exp(\lambda_1 t) + C_2 \exp(\lambda_2 t)$$

with

$$\lambda_{1,2} = \frac{-\tau D \pm \sqrt{\tau^2 D^2 - 4\tau^2 K}}{2\tau^2}.$$

From condition $D = 2\sqrt{K}$ we obtain $\lambda_1 = \lambda_2 = \frac{-D}{2\tau}$, from which $z = C \exp(-\lambda_1 t)$, for a particular $C \in \mathbb{R}$, depending on the initial condition. Since λ_1 is negative, we have that $\dot{z}$ is negative for any time t. Thus the equilibrium is stable.

The result for the non-homogeneous system (2.1) follows from the fact that $f(s)$ goes to 0 as $t \to \infty$, [89]. □

A second, important, property of dynamical system (2.1) is that it is invariant under translations.

Proposition 2.2. *Dynamical System* (2.1) *is invariant under translations.*

Proof. To prove the result, let us consider a translation $x' = x - x^\star$ with fixed $x^\star$. From this, identities $x'_0 = x_0 - x^\star$ and $g' = g - x^\star$ follow. By substituting in (2.1b) we obtain

$$v' = \tau \dot{x}' = \tau \frac{d(x - x^\star)}{dt} = \tau \left(\frac{dx}{dt} - \frac{dx^\star}{dt} \right) = \tau \dot{x} = v,$$

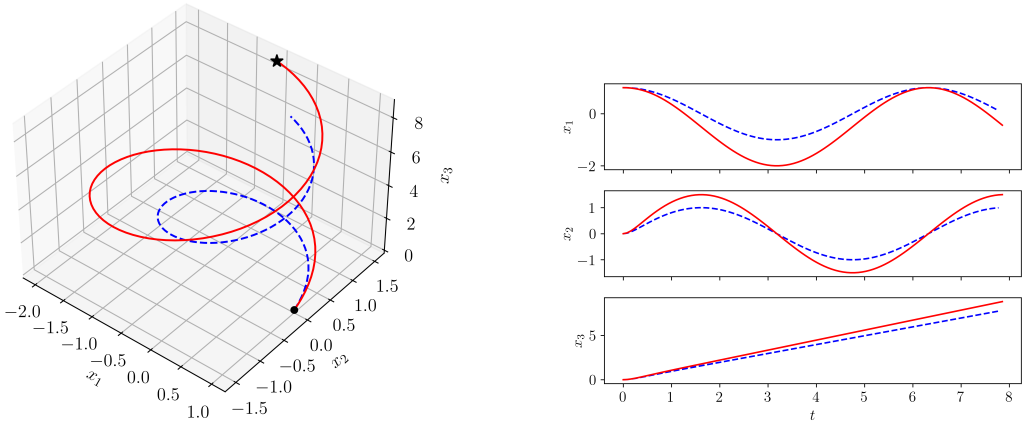
where we used the fact that $x^\star$ is fixed (thus $dx^\star/dt = 0$).
Similarly, by substituting in (2.1a) we obtain

$$\begin{aligned} \tau \dot{v} &= K(g' - x') - Dv + (g' - x'_0) f(s) \\ &= K(g - x^\star - x + x^\star) - Dv + (g - x^\star - x_0 + x^\star) f(s) \\ &= K(g - x) - Dv + (g - x_0) f(s). \end{aligned}$$

Thus, the vector field governing the evolution of the solution is the same of (2.1), hence the thesis. □

Up to now, we considered only 1−dimensional DMP systems. When a d−dimensional trajectory has to be modeled, d decoupled copies of system (2.1) have to be used. This results in the model

$$\begin{cases} \tau \dot{\mathbf{v}} = \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{D}\mathbf{v} + (\mathbf{g} - \mathbf{x}_0) \odot \mathbf{f}(s) & \text{(2.9a)} \\ \tau \dot{\mathbf{x}} = \mathbf{v} & \text{(2.9b)} \end{cases}$$



(a) Trajectory.

(b) Solution.

Figure 2.2: Example of execution of a DMP (2.9). In both plots, the blue dashed line shows the learned behavior, while the red solid line shows the generalization to the new goal position (marked with a black star in Figure 2.2a).

Vectors $\mathbf{x}, \mathbf{v} \in \mathbb{R}^d$ are the position and the velocity of the system. Matrices $\mathbf{K} = \text{diag}(K_1, K_2, \dots, K_d)$ and $\mathbf{D} = \text{diag}(D_1, D_2, \dots, D_d) \in \mathbb{R}_+^{d \times d}$ are diagonal matrices. To maintain the stability of dynamical system (2.9), the following relation must holds: $D_i = 2\sqrt{K_i}$, for all $i = 1, 2, \dots, d$. Vectors $\mathbf{x}_0, \mathbf{g} \in \mathbb{R}^d$ are, respectively, the starting and goal positions. Scalar $\tau \in \mathbb{R}_+$ is the temporal scaling factor (as in the 1−dimensional formulation (2.1)).

Scalar $s \in (0, 1]$ is still governed by the canonical system (2.2). In the vector formulation (2.9), s has an additional role than the scalar formulation (2.1). It is still used to make the system autonomous, making it easier to temporally scale the system evolution. Moreover, in the vector formulation (2.9) it is used also to synchronize all the components of the system.

Operator $\odot$ denotes the component-by-component product: given $\mathbf{v} = [v_i]_{i=1,2,\dots,d}, \mathbf{w} = [w_i]_{i=1,2,\dots,d}$, then

$$\mathbf{v} \odot \mathbf{w} \stackrel{\text{def}}{=} [v_i w_i]_{i=1,2,\dots,d}.$$

Since formulation (2.9) is obtained by making d decoupled copies of dynamical system (2.1), we have that both the existence of a unique stable equilibrium (Proposition 2.1) and the invariance under translations (Proposition 2.2) hold true also in the d−dimensional case. Indeed, to prove these results, it is sufficient to decouple the system and prove the result for each direction $p = 1, 2, \dots, d$. The proof for each component is the same as the proofs of the scalar case.

Example 2.1. We show an example of DMPs' evolution. Let us consider the desired

curve $\mathbf{x}(t) \in \mathbb{R}^3$:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} \cos(t) \\ \sin(t) \\ t \end{bmatrix}, \qquad t \in \left[0, \frac{5}{2}\pi\right].$$

In Figure 2.2 we show the desired trajectory and the resulting DMP when changing the goal position. As it can be seen, the shape of both the trajectory and solution resembles the learned behavior, while generalizing to the new desired behavior (i.e. convergence to the new, desired, goal position).

2.2. New Formulation

Three main drawbacks characterize DMP formulation (2.9) (see [104]):

D1 If the goal position g coincides, in any direction, with the initial position x_0, $g = x_0$, the perturbation term $f(s)$ does not contribute to the evolution of the solution, since it is multiplied by zero;

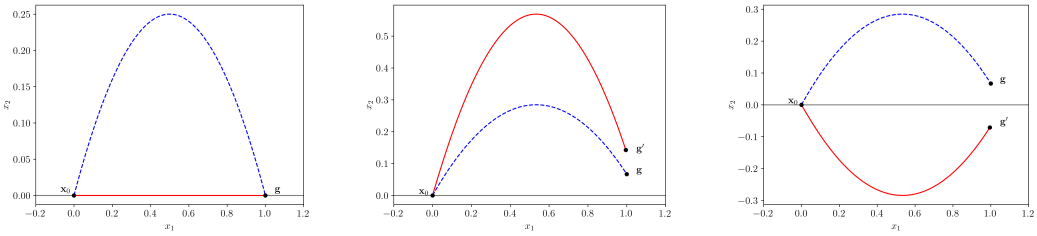
D2 If $g - x_0$ is "small" the scaling of the perturbation term f by $g - x_0$ may produce unexpected (and undesired) behaviors when the relative position $g - x_0$ changes from the learned trajectory to the new desired behavior;

D3 If the scaling factor $g - x_0$ changes sign from the learned trajectory to the new one, the trajectory will result mirrored, since the product $(g - x_0)f(s)$ changes sign.

An example of each of these three drawbacks is shown in Figure 2.3. In particular, Figure 2.3a shows drawback D1: in the vertical component, $\mathbf{x}_0$ and $\mathbf{g}$ coincide. Thus, there is no evolution in the vertical component of the resulting DMP. Figure 2.3b shows drawback D2: the vertical component of $\mathbf{g} - \mathbf{x}_0$ is 'small'. When the learned DMP is executed by slightly changing the goal position, the vertical component of the trajectory results excessively distorted. Finally, Figure 2.3c shows drawback D3: in the learned trajectory, the vertical component of $\mathbf{g} - \mathbf{x}_0$ is positive. On the other hand, the vertical component of $\mathbf{g}' - \mathbf{x}_0$ is negative, and the desired trajectory results in a mirrored version (w.r.t. the horizontal axis $x_2 = 0$) of the learned one.

To overcome these disadvantages, an updated formulation has been proposed in [103, 104, 54] by considering, instaead of (2.9), the system

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g} - \mathbf{x}_0)s + \mathbf{K}\mathbf{f}(s) & \text{(2.10a)} \\ \tau\dot{\mathbf{x}} = \mathbf{v} & \text{(2.10b)} \end{cases}$$



(a) Drawback D1: the vertical component of $\mathbf{g} - \mathbf{x}_0$ is null.

(b) Drawback D2: the vertical component of $\mathbf{g} - \mathbf{x}_0$ is doubled when changing it by a little amount.

(c) Drawback D3: the vertical component of $\mathbf{g} - \mathbf{x}_0$ changes sign.

Figure 2.3: Example of the three drawbacks that characterize DMP formulation (2.9). In all three plots, the blue dashed line shows the desired (and learned) trajectory, the solid red line the execution of the obtained DMP, and the gray solid line marks the horizontal axis $x_2 = 0$. In the last two plots, $\mathbf{g}'$ denotes the new goal position.

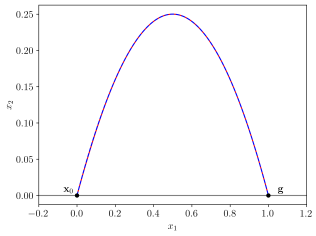
Vectors $\mathbf{x}, \mathbf{v} \in \mathbb{R}^d$ are the position and the velocity of the system.
Matrices $\mathbf{K} = \text{diag}(K_1, K_2, \ldots, K_d), \mathbf{D} = \text{diag}(D_1, D_2, \ldots, D_d) \in \mathbb{R}_+^{d \times d}$ are diagonal matrices, to decouple each direction, satisfying $D_i = 2\sqrt{K_i}$, to maintain the system critically damped. Vectors $\mathbf{x}_0, \mathbf{g} \in \mathbb{R}^d$ are, respectively, the starting and goal positions. Scalar $\tau \in \mathbb{R}_+$ is the temporal scaling factor. Scalar $s \in [0, 1)$ is still governed by the canonical system (2.2). Its usefulness is the same as in formulation (2.9): it allows easy temporal scalability of the evolution of the system and synchronizes each, decoupled, direction.
The forcing term is still written in terms of Gaussian Basis Functions as in (2.3).

We remark that the difference between formulation (2.10) and (2.9) is that the forcing term $\mathbf{f}(s)$ no longer scale with quantity $\mathbf{g} - \mathbf{x}_0$, and it is multiplied by the elastic constant $\mathbf{K}$. Moreover, an initial perturbation $\mathbf{K}(\mathbf{g} - \mathbf{x}_0)s$, that decades exponentially, is added to the acceleration term. This term is required to avoid jumps at the beginning of the movement [104].

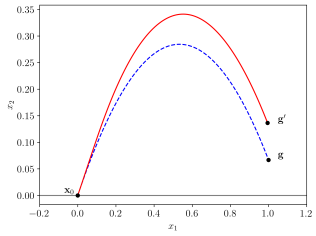
This new formulation maintains the desirable properties of the original DMP formulation.

Proposition 2.3. *Dynamical system* (2.10) *with* s *governed by* (2.2) *converges to* $(\mathbf{x}, \mathbf{v}) = (\mathbf{g}, \mathbf{0})$.

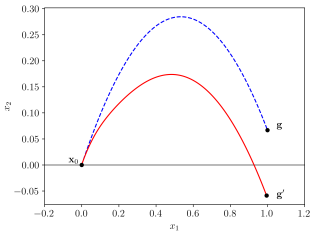
Proof. The proof follows the same argument of Proposition 2.1. We firstly prove the result for the case $s = 0$ and $f(s) \equiv 0$. The proof of this step is identical to the proof of Proposition 2.1. The proof for the general case follows from the fact that, as $t \to +\infty$, $s \to 0$ and $f(s) \to 0$ [89]. □



(a) Behavior of DMP formulation (2.10) in the situation of drawback D1.



(b) Behavior of DMP formulation (2.10) in the situation of drawback D2.



(c) Behavior of DMP formulation (2.10) in the situation of drawback D3.

Figure 2.4: Example of the improved behaviors from DMP formulation (2.10). In all three plots, the blue dashed line shows the desired (and learned) trajectory, the full red line the execution of the obtained DMP, and the gray solid line marks the horizontal axis $x_2 = 0$. In the last two plots, $\mathbf{g}'$ denotes the new goal position (which is different from the learned one). Both the desired curves and the changes in goal position are the same as in Figure 2.3.

Proposition 2.4. *Dynamical system* (2.10) *is invariant under translations.*

Proof. The proof follows the same argument of Proposition 2.2.
We consider a translation $\mathbf{x}' = \mathbf{x} - \mathbf{x}^\star \in \mathbb{R}^d$ with fixed $\mathbf{x}^\star \in \mathbb{R}^d$. From this, it follows $\mathbf{x}'_0 = \mathbf{x}_0 - \mathbf{x}^\star$ and $\mathbf{g}' = \mathbf{g} - \mathbf{x}^\star$. By substituting in (2.10b) we obtain, since $\mathbf{x}^\star$ is fixed,

$$\mathbf{v}' \overset{\text{def}}{=} \tau \dot{\mathbf{x}}' = \tau \frac{d(\mathbf{x} - \mathbf{x}^\star)}{dt} = \mathbf{v}.$$

Similarly, by substituting in (2.10a) we obtain

$$\begin{aligned}
\tau \dot{\mathbf{v}}' &= \mathbf{K}(\mathbf{g}' - \mathbf{x}') - \mathbf{D}\mathbf{v}' - \mathbf{K}(\mathbf{g}' - \mathbf{x}'_0) + \mathbf{K}\mathbf{f}(s) \\
&= \mathbf{K}((\mathbf{g} - \mathbf{x}^\star) - (\mathbf{x} - \mathbf{x}^\star)) - \mathbf{D}\mathbf{v} - \mathbf{K}((\mathbf{g} - \mathbf{x}^\star) - (\mathbf{x}_0 - \mathbf{x}^\star)) + \mathbf{K}\mathbf{f}(s) \\
&= \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g} - \mathbf{x}_0) + \mathbf{K}\mathbf{f}(s).
\end{aligned}$$

Thus, the vector field governing the evolution of the solution is the same of (2.10), hence the thesis. □

Formulation (2.10) solves all the drawbacks presented at the beginning of the Section, as it can be seen in Figure 2.4.
In particular, Figure 2.4a shows that even when a component of $\mathbf{g} - \mathbf{x}_0$ is zero, DMPs formulation (2.10) is able to learn the desired trajectory, thus solving drawback D1. Figure 2.4b shows that even if the vertical component of $\mathbf{g} - \mathbf{x}_0$ is small, the new trajectory does not result distorted when changing goal position, thus solving drawback D2. Finally, Figure 2.4c shows that even if a component of $\mathbf{g} - \mathbf{x}_0$ changes

sign between the learned and the desired behavior, the trajectory does not result mirrored, thus solving drawback D3.

An additional important property of this new formulation is its invariance under general invertible affine transformations.

Theorem 2.1 (Hoffmann et al. [54])**.** *DMP formulation* (2.10) *is invariant under invertible affine transformations of the coordinate system.*

Proof. Since invariance under translations has already been proved in Proposition 2.4, we need to prove the invariance only under linear invertible transformations.
To do so, let us consider an invertible matrix $\mathbf{S} \in \mathbb{R}^{d\times d}$, and let us perform the following substitutions:

$$\mathbf{x} = \mathbf{S}\mathbf{x}', \quad \mathbf{x}_0 = \mathbf{S}\mathbf{x}'_0, \quad \mathbf{g} = \mathbf{S}\mathbf{g}', \quad \mathbf{v} = \mathbf{S}\mathbf{v}', \quad \dot{\mathbf{v}} = \mathbf{S}\dot{\mathbf{v}}', \quad \mathbf{K} = \mathbf{S}\mathbf{K}'\mathbf{S}^{-1}, \quad \mathbf{D} = \mathbf{S}\mathbf{D}'\mathbf{S}^{-1}. \tag{2.11}$$

With this substitution, (2.10b) reads

$$\tau\mathbf{S}\dot{\mathbf{x}}' = \mathbf{S}\mathbf{v}' \quad \Leftrightarrow \quad \tau\dot{\mathbf{x}}' = \mathbf{v}'.$$

Now, by substituting (2.11) into (2.10a) we obtain

$$\begin{aligned}
\mathbf{f}(s) &= \mathbf{K}^{-1}(\tau\dot{\mathbf{v}} - \mathbf{K}(\mathbf{g}-\mathbf{x}) + \mathbf{D}\mathbf{v} + \mathbf{K}(\mathbf{g}-\mathbf{x}_0)s) \\
&= \tau\mathbf{K}^{-1}\dot{\mathbf{v}} - (\mathbf{g}-\mathbf{x}) + \mathbf{K}^{-1}\mathbf{D}\mathbf{v} + (\mathbf{g}-\mathbf{x}_0)s \\
&= \tau\underbrace{\mathbf{S}\mathbf{K}'^{-1}\mathbf{S}^{-1}}_{\mathbf{K}^{-1}}\underbrace{\mathbf{S}\dot{\mathbf{v}}'}_{\dot{\mathbf{v}}} - \underbrace{\mathbf{S}(\mathbf{g}'-\mathbf{x}')}_{\mathbf{g}-\mathbf{x}} + \underbrace{\mathbf{S}\mathbf{K}'^{-1}\mathbf{S}^{-1}}_{\mathbf{K}^{-1}}\underbrace{\mathbf{S}\mathbf{D}'\mathbf{S}^{-1}}_{\mathbf{D}}\underbrace{\mathbf{S}\mathbf{v}'}_{\mathbf{v}} + \underbrace{\mathbf{S}(\mathbf{g}'-\mathbf{x}'_0)}_{\mathbf{g}-\mathbf{x}_0}s
\end{aligned}$$

Next, we can simplify all occurrences of $\mathbf{S}^{-1}\mathbf{S}$,

$$\begin{aligned}
\mathbf{f}(s) &= \tau\mathbf{S}\mathbf{K}'^{-1}\underbrace{\mathbf{S}^{-1}\mathbf{S}}_{\mathbf{Id}_d}\dot{\mathbf{v}}' - \mathbf{S}(\mathbf{g}'-\mathbf{x}') + \mathbf{S}\mathbf{K}'^{-1}\underbrace{\mathbf{S}^{-1}\mathbf{S}}_{\mathbf{Id}_d}\mathbf{D}'\underbrace{\mathbf{S}^{-1}\mathbf{S}}_{\mathbf{Id}_d}\mathbf{v}' + \mathbf{S}(\mathbf{g}'-\mathbf{x}'_0)s. \\
&= \tau\mathbf{S}\mathbf{K}'^{-1}\dot{\mathbf{v}}' - \mathbf{S}(\mathbf{g}'-\mathbf{x}') + \mathbf{S}\mathbf{K}'^{-1}\mathbf{D}'\mathbf{v}' + \mathbf{S}(\mathbf{g}'-\mathbf{x}'_0)s
\end{aligned}$$

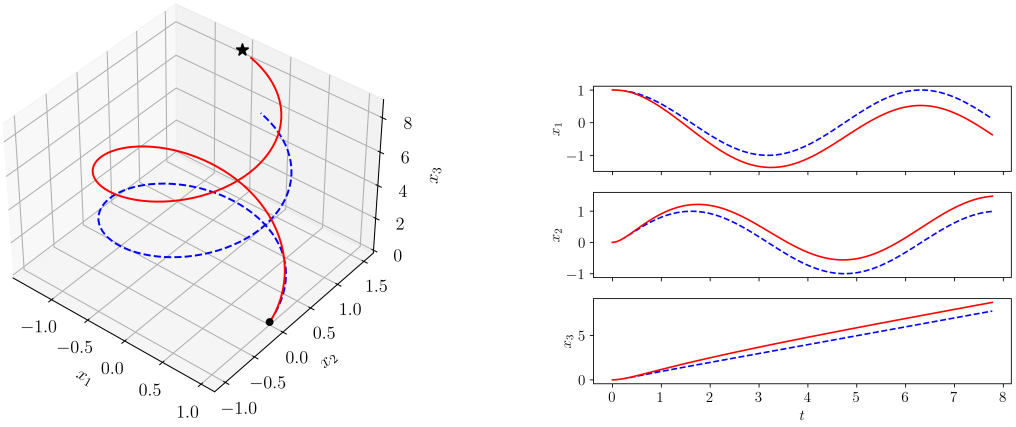
Finally, we can gather term $\mathbf{S}$ obtaining

$$\begin{aligned}
\mathbf{f}(s) &= \mathbf{S}\Big(\tau\mathbf{K}'^{-1}\dot{\mathbf{v}}' - (\mathbf{g}'-\mathbf{x}') + \mathbf{K}'^{-1}\mathbf{D}'\mathbf{v}' + (\mathbf{g}'-\mathbf{x}'_0)s\Big) \\
&= \mathbf{S}\mathbf{f}'(s).
\end{aligned}$$

Thus, if we substitute (2.11) and

$$\mathbf{f} = \mathbf{S}\mathbf{f}' \tag{2.12}$$

in (2.10), we obtain for the prime variables $\mathbf{x}'$ and $\mathbf{v}'$ the same equation (2.10), thus proving the invariance under these transformations. □



(a) Trajectory.

(b) Solution.

Figure 2.5: Example of execution of a DMP (2.10). In both plots, the blue dashed line shows the learned behavior, while the red solid line shows the generalization to the new goal position (marked with a black star in Figure 2.5a).

We will see later in Section 2.6 how this property can be used to increase the robustness of DMPs against any affine transformation of the reference frame.

Example 2.2. We present an example to show the DMP behavior with formulation (2.10). Similarly to Example 2.1, we consider the desired curve $\mathbf{x}(t) \in \mathbb{R}^3$:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} \cos(t) \\ \sin(t) \\ t \end{bmatrix}, \qquad t \in \left[0, \frac{5}{2}\pi\right].$$

In Figure 2.5 we show the desired trajectory and the resulting DMP when changing the goal position. As it can be seen, the shape of both the trajectory and solution resembles the learned behavior, while generalizing to the new desired behavior (i.e. convergence to the new, desired, goal position).
We remark that both formulations (2.9) and (2.10) converge to the new goal position. However, the two behaviors are different. This is clearly evident when comparing components x_1 and x_2 in Figures 2.2b and 2.5b.

2.3. Learning the Forcing Term

In this Section, we present the *learning phase*, which aims at computing the weights ω_i in (2.3) given a desired trajectory. We will refer to DMP formulation (2.10), but we remark that the learning phase can straightforwardly be adapted to formulation (2.9).

Later, in Section 2.7, we will generalize the learning phase to show how a unique DMP (with formulation (2.10)) can be extracted from a set of desired trajectories.

At first, we record a desired trajectory $\tilde{\mathbf{x}}(t)$ and its velocity $\tilde{\mathbf{v}}(t)$, $t \in [0, T]$. Then we set $\tau = 1$.
We remark that s can be expressed analytically. Indeed, since τ and α are constant, the solution to the canonical system (2.2) is (for $\tau = 1$)

$$s = \exp(-\alpha t), \qquad t \in [0, T].$$

The forcing term $\mathbf{f}$ is learned component by component. For each degree-of-freedom $p = 1, 2, \ldots, d$, the forcing term can then be computed using (2.10a) obtaining

$$\tilde{f}_p(s) = \frac{\dot{\tilde{v}}_p + D_p \tilde{v}_p}{K_p} - (g_p - \tilde{x}_p) + (g_p - \tilde{x}_{0p})\, s, \tag{2.13}$$

where the subscript p refers to the $p-$th component for vectors $\mathbf{x}, \mathbf{v}, \mathbf{f}, \mathbf{g}$, and $\mathbf{x}_0$, and to the $(p,p)-$th component of matrices $\mathbf{K}$ and $\mathbf{D}$. We recall that $\mathbf{K}$ and $\mathbf{D}$ are diagonal matrices $\mathbf{K} = \mathrm{diag}(K_1, K_2, \ldots, K_d)$, $\mathbf{D} = \mathrm{diag}(D_1, D_2, \ldots, D_d)$, otherwise it would not be possible to decouple the learning phase along each component.

At this point, we have to compute the weights ω_i that best approximate the desired forcing term $\tilde{f}$ (we drop the index of direction for notation simplicity). Thus, we are searching for the vector $\boldsymbol{\omega}^\star \in \mathbb{R}^{N+1}$ that realizes

$$\begin{aligned}
\boldsymbol{\omega}^\star &= \arg\min_{\boldsymbol{\omega} \in \mathbb{R}^{N+1}} \left\| \frac{\sum_{i=1}^N \omega_i \psi_i(s)}{\sum_{i=0}^N \psi_i(s)}\, s - \tilde{f}(s) \right\|_2^2 \\
&= \arg\min_{\boldsymbol{\omega} \in \mathbb{R}^{N+1}} \underbrace{\int_{s_0}^{s_1} \left(\frac{\sum_{i=0}^N \omega_i \psi_i(s)}{\sum_{i=0}^N \psi_i(s)}\, s - \tilde{f}(s) \right)^2 ds}_{F(\boldsymbol{\omega})}.
\end{aligned}$$

Function $F : \mathbb{R}^{N+1} \to \mathbb{R}^+$ is continuous and strictly convex, which means that the minimum exists and is unique. Moreover, since F is smooth, the minimum can be obtained by nullifying its gradient:

$$\nabla_{\boldsymbol{\omega}} F(\cdot)|_{\boldsymbol{\omega}^\star} = \mathbf{0}.$$

To do so, we start by calling G the gradient of F: $G(\cdot) \doteq \nabla_{\boldsymbol{\omega}} F(\cdot)$. Its h-th component is

$$\begin{aligned}
G_h(\boldsymbol{\omega}) &= \frac{\partial F}{\partial \omega_h}(\boldsymbol{\omega}) \\
&= 2 \int_{s_0}^{s_1} \left(\frac{\sum_{i=0}^N \omega_i \psi_i(s)}{\sum_{i=0}^N \psi_i(s)}\, s - \tilde{f}(s) \right) \left(\frac{\psi_h(s)}{\sum_{i=0}^N \psi_i(s)}\, s \right) ds.
\end{aligned}$$

Function G is linear in $\boldsymbol{\omega}$, thus the minimization problem can be written as a linear system

$$\mathbf{A}\,\boldsymbol{\omega} = \mathbf{b}. \tag{2.14}$$

The component in row h and column k of $\mathbf{A}$, a_{hk}, is, for $h,k \in \{0,1,\dots,N\}$,

$$\begin{aligned} a_{hk} &= \frac{\partial G_h}{\partial \omega_k}(\omega) \\ &= 2\int_{s_0}^{s_1} \frac{\psi_h(s)\psi_k(s)}{\left(\sum_{i=0}^{N}\psi_i(s)\right)^2}\, s^2\, ds, \end{aligned} \tag{2.15a}$$

while the h-th component of the vector $\mathbf{b}$, b_h, is, for $h = 0,1,\dots,N$,

$$b_h = 2\int_{s_0}^{s_1} \left(\frac{\psi_h(s)}{\sum_{i=0}^{N}\psi_i(s)}\, f(s)\, s \right) ds. \tag{2.15b}$$

Using any quadrature formula (e.g. Simpson's rule), the integrals in equations (2.15a) and (2.15b) can be estimated, obtaining matrix $\mathbf{A}$ and vector $\mathbf{b}$ of the linear problem. Finally, we can solve the linear problem $\mathbf{A}\boldsymbol{\omega} = \mathbf{b}$ obtaining $\boldsymbol{\omega}^\star$.
We remark that this procedure has to be repeated for each direction $p = 1,2,\dots,d$.

Algorithm 2.1 summarizes the algorithm to learn the weights relative to the forcing term. We remark that, since the matrix $\mathbf{A}$ does not depend on the forcing term $f(s)$, but only on the set of basis functions $\{\varphi_i(s)\}_{i=0,1,\dots,N}$, we can compute it once for all outside the 'for' loop along the components $p = 1,2,\dots,d$.

2.4. Execution of a Learned Trajectory

To execute a learned trajectory, we simply solve numerically the dynamical system (2.10), using the weights learned previously to compute the forcing term $\mathbf{f}$. Since the goal position can change during time, the system may need "more time" to reach convergence than the learned one. Thus, instead of solving the problem up to a final time, one may decide to let the system evolve until convergence to the goal (which is guaranteed by Proposition 2.3) is achieved within a given tolerance. In such case, the final part of the trajectory may not have an influent forcing term, since $f(s(t)) \approx 0$ when $t \gg T$.

We decided to use an exponential integrator (in particular the *exponential Euler* method [15]) to compute the evolution of the DMP (2.10). We decided to use this numerical integrator since it ensures greater numerical stability than the classical forward Euler scheme.

Algorithm 2.1 Dynamic Movement Primitives: Learning Phase

Input: Desired forcing term $\mathbf{f}(s) = [f_p(s)]_{p=1,2,\dots,d}$, number of basis functions N.

Output: Weights $\mathbf{W} = [\omega_{pk}] \in \mathbb{R}^{d\times(N+1)}$ s.t. ω_{pk} is the k-th weight of the i-th direction of the forcing term

▷ Compute the matrix $\mathbf{A} = [a_{hk}]$ of the minimization problem

for $h = 0, 1, \dots, N$ **do**
  $a_{hh} = 2\int_{s_0}^{s_1} \left(\psi_h^2(s) / \left(\sum_{i=0}^{N} \psi_i(s)\right)^2\right) s^2\, ds$
  for $k = h+1, h+2, \dots, N$ **do**
    $a_{hk} = 2\int_{s_0}^{s_1} \left(\psi_h(s)\psi_k(s) / \left(\sum_{i=0}^{N} \psi_i(s)\right)^2\right) s^2\, ds$
    $a_{kh} = a_{hk}$
  end for
end for

▷ Decouple the problem in each direction

for $p = 1, 2, \dots, d$ **do**

▷ Compute the term $\mathbf{b} = [b_h]$

  for $h = 0, 1, \dots, N$ **do**
    $b_h = 2\int_{s_0}^{s_1} \left(\psi_h(s) / \left(\sum_{i=0}^{N} \psi_i(s)\right) f(s)\, s\, ds\right)$
  end for

▷ Compute the weights $\mathbf{W}_{p,:} = \mathbf{A} \backslash \mathbf{b}$

end for

The canonical system (2.2) can be integrated exactly. For any time interval δ_t, the following relation holds:

$$s(t + \delta_t) = \exp\left(-\frac{\alpha}{\tau}\delta_t\right) s(t), \qquad \forall t \in \mathbb{R},$$

since the exact solution is $s(t) = \exp(-\alpha/\tau\, t)$.

To obtain the numerical scheme for integrating (2.10) let us start by writing the 1 - dimensional formulation in matrix-vector form, expliciting the linear component:

$$\tau \begin{bmatrix} \dot{v} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} -D & -K \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v \\ x \end{bmatrix} + \begin{bmatrix} K(g(1-s) + x_0\, s + f(s)) \\ 0 \end{bmatrix}.$$

for a $d-$dimensional DMP system (2.10), we start by defining the system's state as

$$\mathbf{z} \doteq [v_1, x_1, v_2, x_2, \dots, v_d, x_d]^\intercal. \tag{2.16}$$

Now, vector formulation of DMPs (2.10) can be written as a single system as

$$\tau \dot{\mathbf{z}} = \boldsymbol{\Xi}\, \mathbf{z} + \boldsymbol{\beta}(s). \tag{2.17}$$

Matrix $\boldsymbol{\Xi}$ is a block-diagonal matrix which describes the linear component of the system (2.10) (the blank spaces should be intended as zeros in (2.18)):

$$\boldsymbol{\Xi} = \begin{bmatrix} \boxed{\begin{matrix} -D_1 & -K_1 \\ 1 & 0 \end{matrix}} & & & \\ & \boxed{\begin{matrix} -D_2 & -K_2 \\ 1 & 0 \end{matrix}} & & \\ & & \ddots & \\ & & & \boxed{\begin{matrix} -D_d & -K_d \\ 1 & 0 \end{matrix}} \end{bmatrix}, \tag{2.18}$$

and $\boldsymbol{\beta}(s)$ contains all the remaining terms

$$\boldsymbol{\beta}(s) = \begin{bmatrix} K_1(g_1(1-s) + x_{01}s + f_1(s)) \\ 0 \\ K_2(g_2(1-s) + x_{02}s + f_2(s)) \\ 0 \\ \vdots \\ \vdots \\ K_d(g_d(1-s) + x_{0d}s + f_d(s)) \\ 0 \end{bmatrix}. \tag{2.19}$$

The numerical integration can be carried with any method. We choose to use the *Exponential Euler method*, described in Appendix A.1.

We remark that the numerical scheme shown above require $\mathbf{K}$ and $\mathbf{D}$ to be diagonal.

In the general case in which $\mathbf{K} = [K_{ij}]_{i,j,=1,2,\ldots,d}$ and $\mathbf{D} = [D_{ij}]_{i,j,=1,2,\ldots,d}$ are general matrices, the numerical scheme needs to be changed. Two possible alternatives can be considered.

The first alternative consists in maintaining the state vector $\mathbf{z}$ as in (2.16). In such case, we define

$$\widetilde{\mathbf{K}} = \mathbf{K} \otimes \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

$$\widetilde{\mathbf{D}} = \mathbf{D} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},$$

$$\widetilde{\mathbf{I}\mathrm{d}} = \mathbf{I}\mathrm{d}_d \otimes \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},$$

where $\otimes$ denotes the *Kroneker product* between matrices $A = [a_{ij}]_{i=1,2,\dots,m_A}^{j=1,2,\dots,n_A}$ and $B = [b_{ij}]_{i=1,2,\dots,m_B}^{j=1,2,\dots,n_B}$:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n_A}B \\ a_{21}B & a_{22}B & \cdots & a_{2n_A}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_A1}B & a_{m_A2}B & \cdots & a_{m_An_A}B \end{bmatrix} \in \mathbb{R}^{m_Bm_B \times n_An_B}.$$

With these definitions, matrix $\boldsymbol{\Xi}$ in (2.17) becomes

$$\boldsymbol{\Xi} = -\widetilde{\mathbf{K}} - \widetilde{\mathbf{D}} + \widetilde{\mathbf{Id}_d},$$

while the vector $\boldsymbol{\beta}(s)$ becomes

$$\boldsymbol{\beta}(s) = \left(\mathbf{K} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\right) \cdot \begin{bmatrix} g_1(1-s) + x_{01}s + f_1(s) \\ 0 \\ g_2(1-s) + x_{02}s + f_2(s) \\ 0 \\ \vdots \\ \vdots \\ g_d(1-s) + x_{0d}s + f_d(s) \\ 0 \end{bmatrix}.$$

The second alternative consists in modifying the definition of the state $\mathbf{z}$ in (2.16) as

$$\mathbf{z} \doteq [v_1, v_2, \dots, v_d, x_1, x_2, \dots, x_d]^\intercal.$$

With this new definition of the state, the matrix $\boldsymbol{\Xi}$ becomes the block matrix

$$\boldsymbol{\Xi} = \begin{bmatrix} -\mathbf{D} & -\mathbf{K} \\ \mathbf{Id}_d & \mathbf{0} \end{bmatrix}, \tag{2.20}$$

while the vector $\boldsymbol{\beta}(s)$ becomes

$$\boldsymbol{\beta}(s) = \begin{bmatrix} g_1(1-s) + x_{01}s + f_1(s) \\ g_2(1-s) + x_{02}s + f_2(s) \\ \vdots \\ g_d(1-s) + x_{0d}s + f_d(s) \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

We remark that the first alternative is preferable because the resulting matrix Ξ in (2.19) is three-diagonal. On the other hand, in definition (2.20) matrix Ξ is block matrix. In general, $n-$diagonal matrices guarantee a greater numerical stability and efficiency w.r.t. general matrices.

Independently of the definition of matrix Ξ and vector $\beta(\cdot)$, the numerical scheme takes the same form as in (A.4).

2.5. Other Classes of Basis Functions

As we mentioned in Section 2.1, Gaussian basis functions are adopted in the literature to approximate the forcing term $f(s)$. However, they are not the only possibility. Indeed, any class $\{\psi_i\}_{i=0,1,\dots,N}$ of basis functions satisfying $\lim_{t\to\infty}\psi_i(s(t)) = 0, \forall i \in \{0,1,\dots,N\}$ would be a valid choice. In general, compactly supported basis functions are preferable due to the following advantages:

- Faster convergence is guaranteed since there exists a time $T^\star$ such that $\psi_i(t) = 0$ for any $t > T^\star$;
- When a portion of the trajectory has to be updated, only a subset of the weights has to be updated, since each basis function influences only a finite interval $[t_0, t_1]$ of the time domain $[0, T]$.

In this Section, we proposed different sets of compactly supported basis functions. Later, we will perform convergence and stability analysis to compare the performance of these sets against classical Gaussian Radial Basis functions (2.4).

2.5.1. Truncated Gaussian Basis Functions

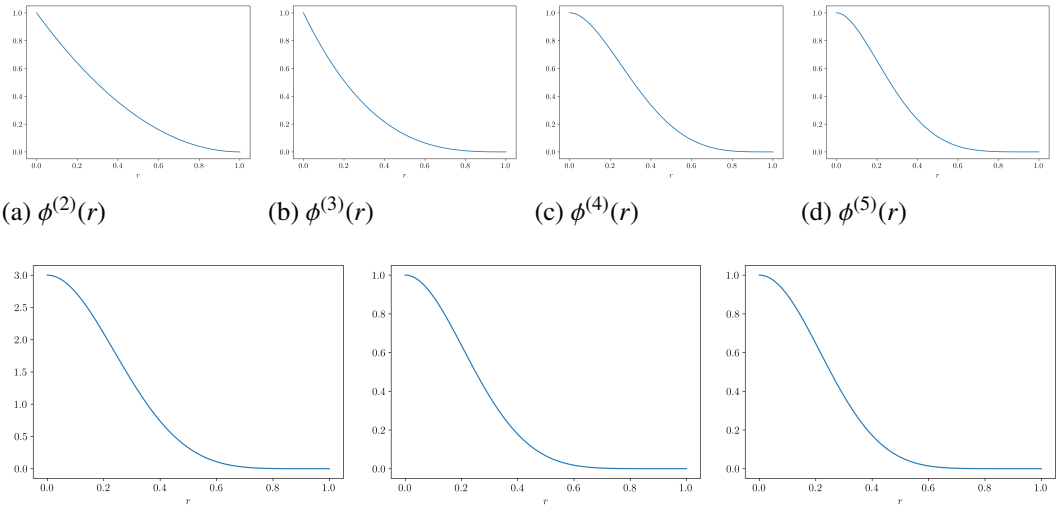
In [134], it was proposed to use truncated Gaussian basis functions

$$\tilde{\psi}_i(s) = \begin{cases} \exp\left(-\frac{h_i}{2}(s-c_i)^2\right) & \text{if } s - c_i \le \mu_i \\ 0 & \text{otherwise} \end{cases}, \tag{2.21}$$

where $\mu_i \in \mathbb{R}$ is a parameter, while c_i and h_i are the centers and widths defined as in (2.5) and (2.6) respectively. This set of basis functions requires a different formulation of the forcing term (2.3) that now reads

$$f(s) = \frac{\sum_{i=1}^{N}(\omega_i s + b_i)\tilde{\psi}_i(s)}{\sum_{i=1}^{N}\tilde{\psi}_i(s) + \varepsilon}, \tag{2.22}$$

where the ω_i's are the weigths and the b_i's are the *biases*. Constant $\varepsilon \in \mathbb{R}_+$ is a small value to avoid division by zero.



(a) $\phi^{(2)}(r)$ (b) $\phi^{(3)}(r)$ (c) $\phi^{(4)}(r)$ (d) $\phi^{(5)}(r)$

(e) $\phi^{(6)}(r)$ (f) $\phi^{(7)}(r)$ (g) $\phi^{(8)}(r)$

Figure 2.6: Wendlan's basis functions (2.24) as functions of r. We omit the dependency on the index i since it change the definition of r, but not the definition of ϕ w.r.t. r.

2.5.2. Compactly Supported Basis Functions

Since truncated Gaussian bassis functions (2.21) formulation introduces discontinuities in the basis functions, in [47] we introduced and compared different classes of compactly supported basis functions. For all of them we maintain the definition of the centers c_i as in (2.5), but we define the widths as

$$\begin{aligned} a_i &= \frac{\tilde{h}}{c_i - c_{i-1}}, \quad i = 1, 2, \ldots, N, \\ a_0 &= a_1, \end{aligned} \tag{2.23}$$

instead of h_i from (2.6). As in (2.6), scalar $\tilde{h} > 0$ determines the overlapping between basis functions. In all our tests, we fix $\tilde{h}$ to value one: $\tilde{h} = 1$.

Well known compactly supported basis functions are *Wendland's functions* [137, 121]. Wendland's functions in one dimension can be defined for any smoothness requirement. Most famous and used Wendland functions are (where the operator $(\cdot)_+$ denotes the *positive part* functions: $(x)_+ = \max\{0, x\}$):

$$\phi_i^{(2)}(r) = (1 - r)_+^{\,2}, \tag{2.24a}$$
$$\phi_i^{(3)}(r) = (1 - r)_+^{\,3}, \tag{2.24b}$$
$$\phi_i^{(4)}(r) = (1 - r)_+^{\,4}(4r + 1), \tag{2.24c}$$
$$\phi_i^{(5)}(r) = (1 - r)_+^{\,5}(5r + 1), \tag{2.24d}$$

$$\phi_i^{(6)}(r) = (1-r)_+^{\,6}(35r^2 + 18r + 3), \tag{2.24e}$$
$$\phi_i^{(7)}(r) = (1-r)_+^{\,7}(16r^2 + 7r + 1), \tag{2.24f}$$
$$\phi_i^{(8)}(r) = (1-r)_+^{\,8}(32r^3 + 25r^2 + 8r + 1), \tag{2.24g}$$

where we defined

$$r = |a_i(s - c_i)|.$$

In Figure 2.6, Wendland basis functions (2.24) are shown.

In [47] we introduced a new class of compactly supported basis functions. To define them, let us first recall the following example of *mollifier* $\varphi : \mathbb{R} \to \mathbb{R}$ defined as

$$\varphi(x) = \begin{cases} \dfrac{1}{I_n} \exp\left(-\dfrac{1}{1-|x|^2}\right) & \text{if } |x| < 1 \\ 0 & \text{otherwise} \end{cases},$$

where I_n is set so that $\int_{\mathbb{R}} \varphi(x)\,dx = 1$. Function φ is smooth, $\varphi \in C^\infty(\mathbb{R})$, and compactly supported.
It is then natural to define the set of *mollifier-like basis functions* $\{\varphi_i(s)\}$ given by

$$\varphi_i(s) = \begin{cases} \exp\left(-\dfrac{1}{1-|a_i(s-c_i)|^2}\right) & \text{if } |a_i(s-c_i)| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

where c_i are the centers as in (2.5), a_i are the widths as in (2.23), and the usual normalization term I_n is omitted since it is not necessary to our purposes.
Both Wendland (2.24) and mollifier-like (2.25) basis functions are compactly supported. The major difference lies in their smoothness. In the Wendland basis functions case, we have that any continuity requirement $C^k(\mathbb{R})$ can be satisfied by a sufficiently high order ℓ of Wendland basis functions. That is, for any $k \in \mathbb{N}$ there exists an $\ell \in \mathbb{N}$ such that $\phi_i^{(\ell)} \in C^k(\mathbb{R})$. However, even for relatively small k's, Wendland functions become quite complicated. On the other hand, mollifier-like basis functions are smooth: $\varphi_i \in C^\infty(\mathbb{R})$.

In Figure 2.7 we plot an example of mollifier-like basis functions both as function of s and of t. We remark that both Wendland (2.24) and mollifier-like (2.25) basis functions satisfy the same properties we highlighted for Gaussian basis functions in Section 2.1, i.e. they are symmetric in s w.r.t. c_i, the centers are equispaced in t and the order of the basis functions is flipped when changing from t to s.

When using compactly supported basis functions, the forcing term formula (2.3) has to be slightly changed to take into account that the denominator may vanish.

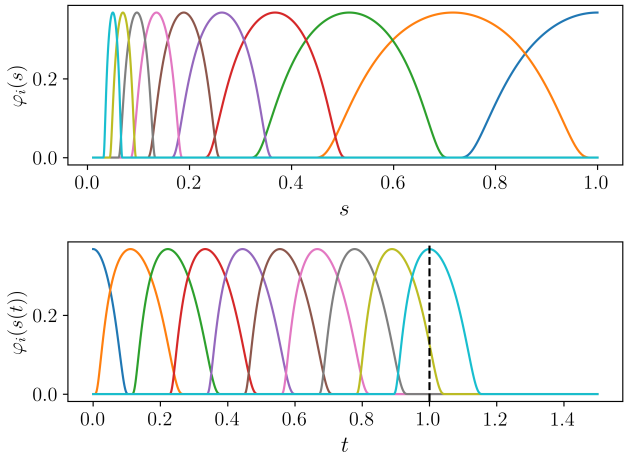


Figure 2.7: Example of Mollifier-like Basis Functions. The upper plot shows the evolution of the functions w.r.t. the parameter s, while the lower plot shows them as a function of t. The parameters are: $T = 1$, $\alpha = 3$, and $N = 9$. Time $T = 1$ is marked by the dashed black line.

Thus, the forcing term reads

$$f(s) = \begin{cases} \dfrac{\sum_{i=0}^{N} \omega_i \varphi_i(s)}{\sum_{i=0}^{N} \varphi_i(s)} s & \text{if } \sum_{i=0}^{N} \varphi_i(s) \neq 0 \\ 0 & \text{otherwise} \end{cases}.$$

In Table 2.1, the properties of various classes of basis functions are summarized. In particular, we remark the regularity and if the support is compact. Gaussians and mollifier-like are the only smooth basis functions, truncated Gaussian are the only discontinuous basis functions, and Wendland basis functions have various continuity orders. Only Wendland and mollifier-like basis functions are compactly supported.

Remark 2.1. Gaussian basis functions are analytic functions, $\psi_i \in C^\omega(\mathbb{R})$, that is they are smooth and are locally given by their Taylor series, $C^\omega(\mathbb{R}) \subset C^\infty(\mathbb{R})$. On the other hand, mollifier-like basis functions are smooth but not analytic, $\varphi_i \in C^\infty(\mathbb{R}) \setminus C^\omega(\mathbb{R})$.

2.5.3. General Considerations

The usage of compactly supported basis functions gives a faster convergence of the DMP system to the goal position since the resulting forcing term is null after a finite amount of time $T^\star > 0$, $f(s(t)) = 0, \forall t > T^\star$. This is not true for (both classical and truncated) Gaussian basis functions (2.4) and (2.21). Indeed, the support of the classical Gaussian basis functions is the whole real line $\mathbb{R}$; while the support for

Function		Regularity	Compactly Supported
Gaussian	$\psi_i(s)(\cdot)$	$\underline{C^\omega(\mathbb{R})}$	No
Truncated Gaussian	$\tilde{\psi}_i(s)(\cdot)$	Discontinuous	No
Wendland	$\phi^{(2)}(\cdot)$	$C^0(\mathbb{R})$	$\underline{\text{Yes}}$
	$\phi^{(3)}(\cdot)$	$C^0(\mathbb{R})$	$\underline{\text{Yes}}$
	$\phi^{(4)}(\cdot)$	$C^2(\mathbb{R})$	$\underline{\text{Yes}}$
	$\phi^{(5)}(\cdot)$	$C^2(\mathbb{R})$	$\underline{\text{Yes}}$
	$\phi^{(6)}(\cdot)$	$C^4(\mathbb{R})$	$\underline{\text{Yes}}$
	$\phi^{(7)}(\cdot)$	$C^4(\mathbb{R})$	$\underline{\text{Yes}}$
	$\phi^{(8)}(\cdot)$	$C^6(\mathbb{R})$	$\underline{\text{Yes}}$
Mollifier-like	$\varphi_i(s)(\cdot)$	$\underline{C^\infty(\mathbb{R})}$	$\underline{\text{Yes}}$

Table 2.1: Summary of the properties (regularity and support compactness) of the different sets of basis functions presented in Section 2.5. The most desirable properties (smoothness and compact support) are underlined.

truncated Gaussian functions is an interval $(-\infty, L]$, $L \in \mathbb{R}$ in s, which is an interval $[L', +\infty)$, $L' \in \mathbb{R}$ in t.

In [134] it was pointed out that if only a portion of the trajectory has to be re-learned, only a subset of weights has to be re-computed. To be more precise, assume that the forcing term $\mathbf{f}(s)$ of a desired trajectory $\mathbf{x}(t)$ has to be updated in the time interval $[t_0, t_1] \subset [0, T]$. Then, not all the weights ω_i have to be re-computed, but only those whose basis function satisfy (where the support has to be intended on the values of t and not s)

$$\text{supp}(\varphi_i) \cap [t_0, t_1] \neq \emptyset.$$

From this, it is easy to notice that for classical Gaussian basis functions this means that all weights have to be re-computed, since their support is the whole real line $\mathbb{R}$: $\text{supp}(\psi_i) = \mathbb{R}$. For compactly supported basis functions, this intersection will usually be smaller than the whole set of indexes $\{0, 1, \ldots, N\}$, and it will depend only on the length $t_1 - t_0$ of the interval.
The support, in s, of each basis function φ_i is the interval $(c_i - 1/a_i, c_i + 1/a_i)$. Therefore, the set of weights $\{\omega_i\}_{i \in I}$ that has to be updated are those for which

$$c_i - \frac{1}{a_i} \leq s_0 \quad \text{and} \quad c_i + \frac{1}{a_i} \geq s_1, \tag{2.26}$$

where $s_0 = \exp(-\alpha t_0)$ and $s_1 = \exp(-\alpha t_1)$.

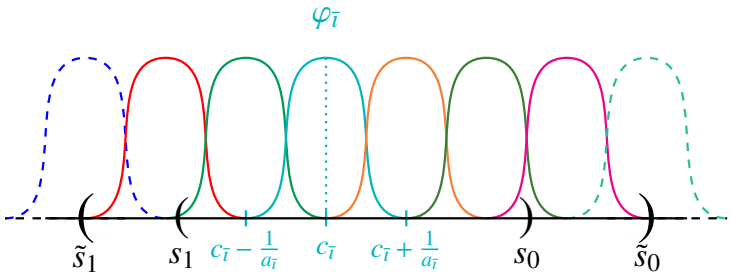


Figure 2.8: Depiction of compactly supported basis functions whose support intersects an interval (s_1, s_0). The basis functions drawn using solid lines are those whose weights have to be updated. As one can observe, they satisfy condition (2.26). Interval $(\tilde{s}_1, \tilde{s}_0)$ shows the domain in which the integrals in (2.15) have to be computed.

Once the set of indexes I has been identified using (2.26), one must solve a linear problem as in (2.14), where $\mathbf{A}$ is a $|I|\times|I|$ matrix, $\mathbf{A} \in \mathbb{R}^{|I|\times|I|}$, and $\mathbf{b}$ is a vector with $|I|$ components, $\mathbf{b} \in \mathbb{R}^{|I|}$. The components of matrix $\mathbf{A}$ and vector $\mathbf{b}$ are given in (2.15). We remark that the integrals in (2.15) have to be evaluated on the interval

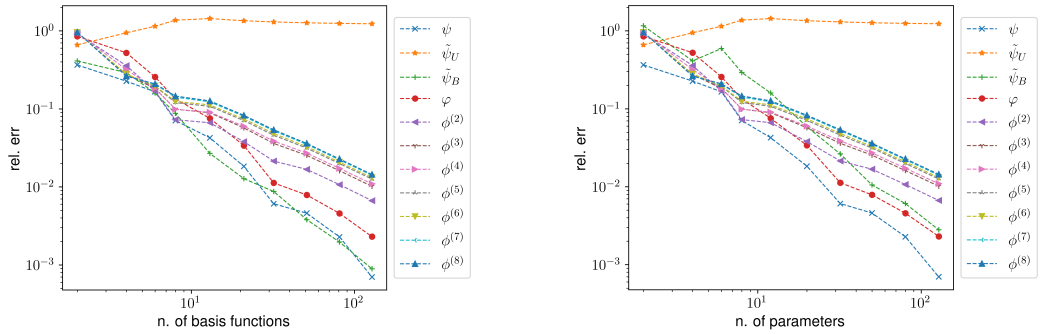
$$(\tilde{s}_1, \tilde{s}_0) = \bigcup_{i\in I} \text{supp}(\varphi_i),$$

where the support has to be intended in s. By solving the linear problem, the weights ω_i, $i \in I$ are updated.
Figure 2.8 gives an intuition on how to identify the set $\{\varphi_i\}_{i\in I}$ of basis functions satisfying (2.26) and the interval $(\tilde{s}_1, \tilde{s}_0)$.

Remark 2.2. In the case of truncated Gaussian basis functions, the set of weights to update will depend not only on the length of the interval but also on its 'position'. For instance, if the last part of the trajectory has to be updated, i.e. $[t_0, t_1] = [t_0, T]$, all the weights must be updated since $T \in \text{supp}(\tilde{\psi}_i)$ for any $i = 0, 1, \ldots, N$.
Thus, in general, when a portion of the trajectory has to be modified, the number of weights to re-compute will be less when using compactly supported basis functions than when using truncated Gaussian basis functions.

2.5.4. Results

In this Section, we present various tests to analyze and compare the numerical accuracy, stability, and time efficiency for the basis functions presented above.
In particular, we first test the numerical accuracy of Gaussian (2.4), truncated Gaussian (2.21), Wendland (2.24), and mollifier-like (2.25) basis functions when approximating a given target function. This test aims at showing that various basis



(a) Error as function of the number of basis functions.

(b) Error as function of the number of parameters.

Figure 2.9: Plot of the L^2 error done by approximating the target function (2.27). The first plot shows the error w.r.t. the number of basis functions, while the second plot shows the error w.r.t. the number of parameters that have to be learned. Truncated Gaussian basis functions are tested both using the (classical) unbiased formulation (2.3) and the biased one (2.22). These two different approaches are denoted respectively by $\tilde{\psi}_U$ and $\tilde{\psi}_B$ in the legend.

functions result in different approximation errors. However, the rates of convergence (as a function of the number of basis functions N) are comparable between basis functions.

As second test, we analyze the condition number (for further details, see Appendix B) of matrix $\mathbf{A} = [a_{hk}]$, with components defined in (2.15a) from the learning process presented in Section 2.3. This test will show how compactly supported basis functions result in a 'numerically more stable' minimization problem.

Thirdly, we will show that using a set of compactly supported basis functions results in a minimization problem that is numerically faster to solve.

Finally, we will provide a synthetic test showing the trajectory update property mentioned in Section 2.5.3.

Numerical Accuracy

In this Section, we test the goodness of the classes of basis functions presented above by computing the error done when approximating various 'target' functions. We perform tests both on a pre-defined target function and on forcing terms extracted from real robotic executions.

At first, we define the target function

$$\eta(t) = t^2 \cos(\pi t), \quad t \in [0, 1]. \tag{2.27}$$

Figure 2.9 shows the approximation error, measured w.r.t. the L^2-norm,

$$\texttt{err} = \left\|\mathbf{x}(t) - \mathbf{x}_{\text{target}}(t)\right\|_2 = \sqrt{\int_0^T (\mathbf{x}(t) - \mathbf{x}_{\text{target}}(t))\, dt},$$

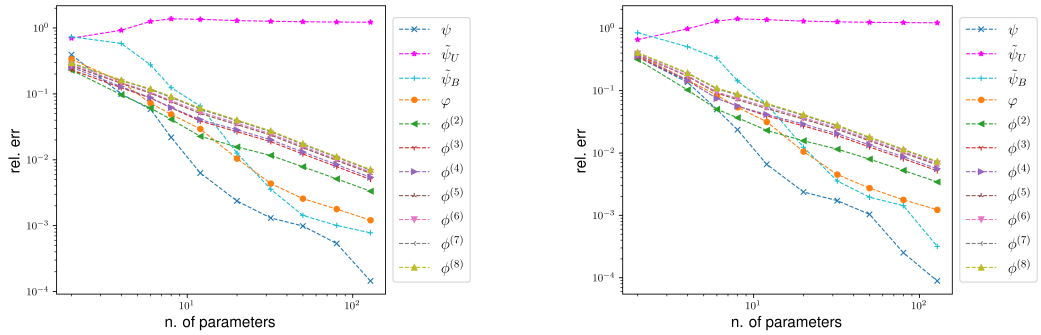
both w.r.t. the number of basis functions (Figure 2.9a), and w.r.t. the number of parameters that need to be learned (Figure 2.9b). We recall that for truncated Gaussian basis functions the number of parameters is double the number of basis functions since every basis function requires one weight and one bias term. On the other hand, for all other classes of basis functions, the number of parameters coincides with the number of basis functions, since for each basis function only one parameter (the weight) has to be computed.

The plot shows that truncated Gaussian functions (2.21) work properly only using the biased formulation (denoted by $\tilde{\psi}_B$ in the legend) for the forcing term (2.22), which means that given N basis functions we are solving a $2N$-long linear system when computing the weights and the biases. On the other hand, when using the unbiased formulation (2.3) (denoted by $\tilde{\psi}_U$ in the legend) the approximant does not converge to the desired forcing term.

We observe that when comparing the error w.r.t. the number of basis functions, the error obtained by using truncated Gaussian basis functions is comparable to the error done using classical Gaussian basis functions. On the other hand, when comparing the error w.r.t. the number of parameters that have to be computed, truncated Gaussian basis functions approximate worse than mollifier basis functions when the number of parameters is below sixty. Usually, in the applications, no more than fifty basis functions are used. For this particular value and target function, mollifier-like basis functions and truncated Gaussian have almost identical approximation errors. However, we remark that these results depend on the particular choice of the target function η in (2.27), and that different target functions may give different results. Tests performed with different target functions show similar results: classical Gaussian functions remain the best approximator, while truncated Gaussian and mollifier-like give similar approximation accuracy and usually work better than Wendland functions.

We tested classical Gaussians, Wendland, and mollifier-like basis functions also using the biased formulation (2.22), without noticing any difference in the goodness of the approximation.

We performed this accuracy test also on trajectories obtained from real task executions. In particular, we relied on the JIGSAW dataset [39]. This dataset contains data on three elementary surgical tasks (suturing, knot-tying, and needle-passing). The demonstrations were collected from eight different subjects of different expertise in robotic surgery. In Figure 2.10 we show the convergence error for two



(a) `pushing_needle`. (b) `positioning_needle`.

Figure 2.10: Plot of the L^2 error done approximating two real gestures. The error is given as a function of the number of parameters that need to be learned. Truncated Gaussian basis functions are tested both using the (classical) unbiased formulation (2.3) and the biased one (2.22). These two different approaches are denoted respectively by $\tilde{\psi}_U$ and $\tilde{\psi}_B$ in the legend.

gestures extracted from the dataset. In particular, Figure 2.10a shows the results for the `pushing_needle` gesture, while Figure 2.10b shows the results for the `positioning_needle` gesture. Both gestures were extracted from the same user ('D') performing the same task (suturing). As it can be seen, the results are similar to those obtained by approximating function η in (2.27): Gaussian basis functions (2.4) are the best approximators, truncated Gaussian basis functions (2.21) and mollifier-like basis functions (2.25) have similar convergence result, and Wendland basis functions (2.24) are the less accurate basis functions.
Even for these tests on real forcing terms, we tested classical Gaussians, Wendland, and mollifier-like basis functions also using the biased formulation (2.22), without noticing any difference in the goodness of the approximation.

Condition Number

The numerical accuracy tested above depends on the particular choice of the target function. Here, we discuss the numerical advantages of the various classes of basis functions with a test that does not depend on the choice of the target function. In particular, we discuss the relative numerical accuracy of the minimization problem used to compute the weights ω_i in (2.14) where $\mathbf{A} = [a_{hk}]$ is defined in (2.15a) and $\mathbf{b} = [b_h]$ is defined in (2.15b). To do so, we investigate the condition number of matrix $\mathbf{A}$, $\text{cond}(\mathbf{A}) \stackrel{\text{def}}{=} \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$. The importance of this test lies in the fact that when one solves numerically a linear system, the approximation error is directly proportional to the condition number (for further details, see Appendix B).
In this test, we do not consider truncated Gaussian basis functions since, in this

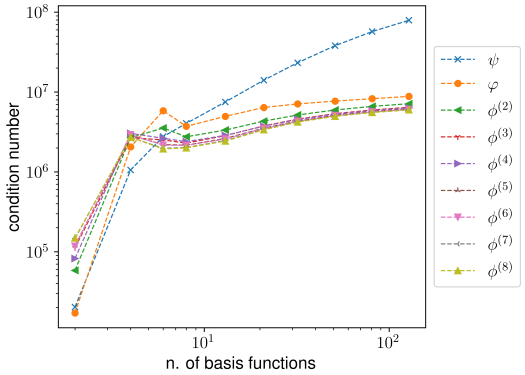


Figure 2.11: Condition number of the matrix $\mathbf{A}$ with elements a_{hk} defined in (2.15a) w.r.t. the number of basis functions.

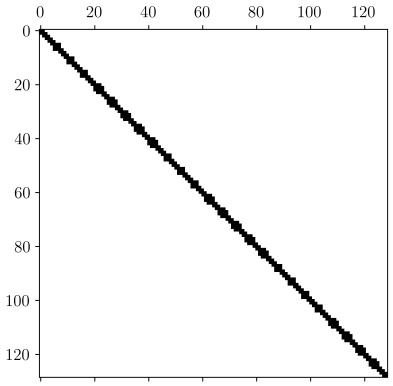


Figure 2.12: Sparsity pattern for matrix $\mathbf{A}$ with elements a_{hk} defined in (2.15a) in the case of $N = 2^7 = 128$ using mollifier-like basis functions.

case, the components $\mathbf{A}$ and $\mathbf{b}$ of the linear problem have different formulations since both the weights and the biases have to be learned.

Figure 2.11 shows that the condition number $\text{cond}(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$ of matrix $\mathbf{A}$ defined as in (2.15a) is bigger for Gaussian basis functions. This means that solving the minimization problem using Gaussian basis functions results in a more severe numerical cancellation error than mollifier-like or Wendland's functions, due to the fact that, when numerically solving a linear system, the relative error is directly proportional to the condition number.

The lower condition number is due to the fact that, since mollifier-like and Wendland's basis functions are compactly supported, the resulting matrix $\mathbf{A}$ will have many off-diagonals components equal to zero, since $\int_{\mathbb{R}} \varphi_i(x)\varphi_j(x)\,dx = 0$ for multiple couples $(i, j) \in \{0, 1, \ldots, N\}^2$. This can be seen in the sparsity pattern shown in Figure 2.12. A "more-sparse" matrix results in an "easier to solve", from a computational point of view, linear problem, which translates in a lower condition number. On the other hand, when using Gaussian basis functions, all the components of $\mathbf{A}$ are non-zero, since $\psi_i(s) > 0, \forall s \in \mathbb{R}$ implies $\int_{\mathbb{R}} \psi_i(x)\psi_j(x)\,dx \neq 0$ for any couple $(i, j) \in \{0, 1, \ldots, N\}^2$. This results in a full matrix $\mathbf{A}$, which gives a bigger condition number.

We recall that convergences shown in Figure 2.9 and 2.10 are done on particular choices of the target function. The convergence error may differ depending on the forcing term that needs to be approximated, and various basis functions should be tested to choose the best one for the particular case.

On the other hand, the condition number shown in Figure 2.11 does not depend on

the target function, but only on the choice of basis functions.

In summary, we showed that Gaussian basis functions (2.4) usually result in the lowest approximation error (even tho, we recall, this depends on the target function). On the other hand, being compactly supported, Wendland and mollifier-like basis functions result in a more numerically stable linear problem in the learning phase.

When it is feasible, various basis functions should be tested to choose the best one for the particular case. However, this is rarely possible in practice. Thus, we encourage the use of mollifier-like basis functions since they are the most accurate compactly supported basis functions. Indeed, this choice allows having a fast to solve and numerically stable minimization problem and null forcing term after a finite amount of time; while having a small approximation error.

Computational Time

An additional advantage in using compactly supported basis functions lies in the improvements in computational time when solving the linear problem (2.14). Indeed, when the number of basis functions N increases, the entries in matrix $\mathbf{A}$ increases quadratically: $O(N^2)$. When Gaussian basis functions are used, all entries in $\mathbf{A}$ are non-zero. On the other hand, when compactly supported basis functions are adopted, the number of non-zero entries in $\mathbf{A}$ increases linearly, $O(N\ell)$, where ℓ is the number of non-zero diagonals in $\mathbf{A}$, and is determined by the overlapping between basis functions, which depends on parameter $\tilde{h}$ in (2.23).
Consequently, as the number of basis functions N increases, the number of operations that are needed to solve the linear problem (2.14) increases cubically, $O(N^3)$, when the matrix is full; and only quadratically, $O(N^2\ell)$, when the matrix is sparse. Thus, one should expect the minimization problem (2.14) to be faster to solve when compactly supported basis functions are used.

To test the improvement in computational time, we perform the following test. For different values of the number of basis functions N, we compute the matrix $\mathbf{A}$ in (2.14). Then, for each value of N, we consider thirty different values for vector $\mathbf{b}$ and solve the linear problem (2.14) saving the time needed to solve it. We perform this test for the Gaussian basis functions (2.4), mollifier-like basis functions (2.25), and Wendland basis functions (2.24).
Figure 2.13 shows the result of this test. In particular, the log-scale plot shows the average computational time for each value of N. As it can be seen, as the number of basis functions increases, the computational time needed to solve the linear problem (2.14) has a greater growth when Gaussian basis functions are used. On the other hand, when compactly supported basis functions are adopted, the increment in computational time is reduced.

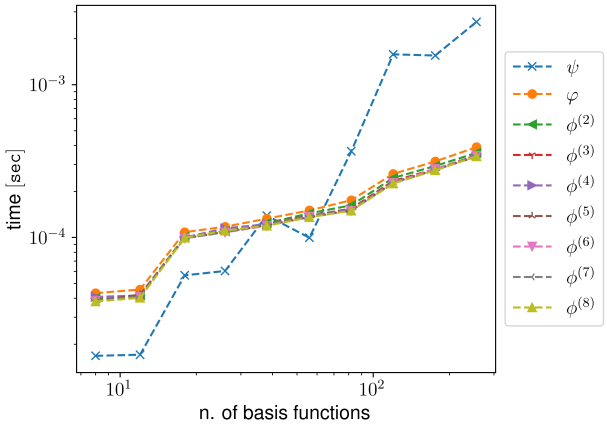


Figure 2.13: Average computational time when solving the minimization problem (2.14) as function of the number of basis functions N.

Tests were performed on a notebook with a quad-core Intel Core i7-7000 CPU with 16 GB of RAM and were implemented using Python 3.7.

Trajectory Update

We now present a synthetic test to show the "trajectory update" property presented before. To do so, we generate two trajectories, which can be seen on the left plot in Figure 2.14, using clamped splines (to have null initial and final velocities). The trajectories are identical on the 'tails' but they differ in the central portion (one is larger than the other). To show the update properties of compactly supported basis functions, we start by learning a DMP from the 'largest' trajectory (shown in red in Figure 2.14). DMPs parameters are $\mathbf{K} = K\,\mathbf{I}\mathrm{d}_2$ and $\mathbf{D} = D\,\mathbf{I}\mathrm{d}_2$ with $K = 150$ and $D = 2\sqrt{K} \approx 24.49$, $\alpha = 4$, and $N = 100$.
Next, we identificate, using (2.26), the set of basis functions that have to be updated when the middle portion of the trajectory has to be updated. This result in the set of indexes $I = \{24, 25, \dots, 63\} \subset \{0, 1, \dots, N\} = \{0, 1, \dots, 100\}$. Then, we re-compute the set of weights $\{\omega_i\}_{i \in I}$.
The plot on the right in Figure 2.14 shows the execution of the resulting, updated DMP. As one can observe, by updating only a subset of the weights the resulting trajectory mimics the new desired behavior.

2.6. Affine-Invariant DMPs

In this Section, we present a modification to DMPs formulation (2.10), based on Theorem 2.1, that improves its adaptability property to arbitrary changes in goal

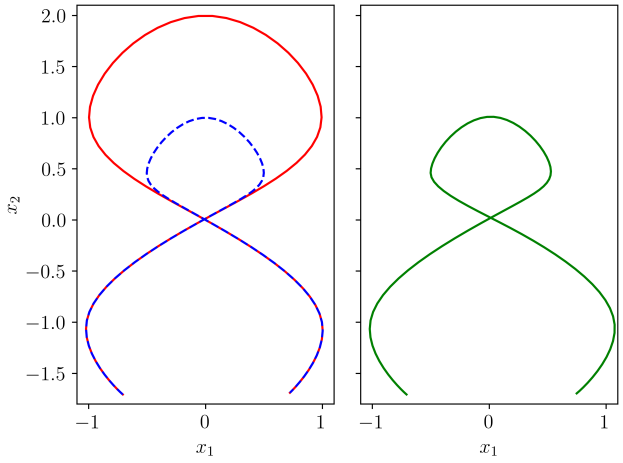


Figure 2.14: Result for the 'trajectory update' property of compactly supported basis functions. On the left, the two trajectories are shown, showing that they differ only in the central portion. On the right, the execution of the obtained DMP is shown. The DMP is initially learned from the 'big' trajectory and then updated using the middle portion of the small one.

position.

Indeed, as we explained in Section 2.2, formulation (2.10) solves some drawbacks of formulation (2.1). However, it also introduces a new drawback: since the forcing term $\mathbf{f}(s)$ no longer scale with quantity $\mathbf{g} - \mathbf{x}_0$, the generalization property works properly only locally, that is only if the new, desired, relative position between goal and starting positions, $\mathbf{g} - \mathbf{x}_0$, is "similar" to the learned one $\mathbf{g}^\star - \mathbf{x}_0^\star$.

Example 2.3. In Figure 2.15 we show how DMP formulation (2.10) generalizes when modifying the goal position. The desired trajectory is

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t \\ \sin^2(t) \end{bmatrix}, \qquad t \in [0, \pi] .$$

The original goal position is $\mathbf{g}^\star = \mathbf{x}(\pi) = [\pi, 0]^\intercal$, while the new goal poistions are $\mathbf{g}' = (\pi, 1/2)$ and $\mathbf{g}'' = (-1, 1)$. As it can be seen, the DMP generalizes well for $\mathbf{g}'$, i.e when the learned and modified goal positions are similar. On the other hand, when the original $\mathbf{g}^\star$ and new goal positions $\mathbf{g}''$ are sensibly different, the resulting DMP appears distorted.

As mentioned above, thanks to Theorem 2.1, we can solve this unique drawback of formulation (2.10) [47]. Indeed, consider a trajectory which is learned together with the relative position between the goal and the starting point, i.e. the vector $\mathbf{g}^\star - \mathbf{x}_0^\star$; and a new trajectory, with starting and ending points $\mathbf{x}_0$ and $\mathbf{g}$ respectively has to be executed. If we were able to compute an invertible transformation matrix

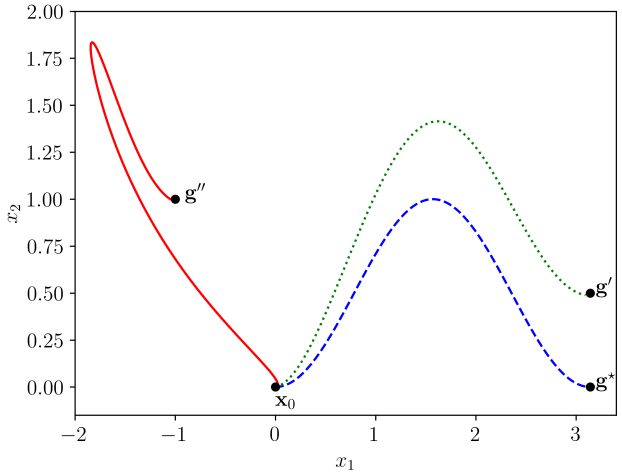


Figure 2.15: Comparison of the same DMP with different goal positions. The desired (and learned) trajectory is showed in dashed blue. Trajectories in dotted green and solid red show the resulting DMP for different goal positions, $\mathbf{g}'$ and $\mathbf{g}''$ respectively.

$\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ such that $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}(\mathbf{g}^\star-\mathbf{x}_0^\star)=\mathbf{g}-\mathbf{x}_0$, then we could apply transformations (2.11) and (2.12) to make the new trajectory a rescaled version of the learned one. Following this idea, we present the *extended DMPs* as

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}(\mathbf{g}-\mathbf{x}) - \mathbf{D}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\mathbf{v} - \mathbf{K}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}(\mathbf{g}-\mathbf{x}_0)s + \mathbf{K}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\mathbf{f}(s) & \text{(2.28a)} \\ \tau\dot{\mathbf{x}} = \mathbf{v} & \text{(2.28b)} \end{cases}$$

where, from (2.11) and (2.12) we define

$$\mathbf{K}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\,\mathbf{K}\left(\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\right)^{-1},$$
$$\mathbf{D}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\,\mathbf{D}\left(\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\right)^{-1}.$$

During the learning fase, $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ is fixed to be the identity matrix.
We remark that extended DMPs can be used even if the goal position changes with time during the execution of the trajectory simply by updating the matrix $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$.
We also remark that, when matrices $\mathbf{K}$ and $\mathbf{D}$ are multiple of the identity matrix, $\mathbf{K}=K\,\mathbf{Id}$ and $\mathbf{D}=D\,\mathbf{Id}$, we have

$$\mathbf{K}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \mathbf{K},$$
$$\mathbf{D}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \mathbf{D}.$$

In this case, extended DMPs (**??**) reads

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g}-\mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g}-\mathbf{x}_0)s + \mathbf{K}\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}\mathbf{f}(s) \\ \tau\dot{\mathbf{x}} = \mathbf{v} \end{cases}$$

A case of particular interest is when $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ describes a *roto-dilatation*. We first compute the rotation matrix $\mathbf{R}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$, which maps the unit vector $\widehat{\mathbf{g}^\star-\mathbf{x}_0^\star}$ to $\widehat{\mathbf{g}-\mathbf{x}_0}$, where $\hat{\mathbf{v}}$ denotes the unit vector with direction given from $\mathbf{v}$

$$\hat{\mathbf{v}} \stackrel{\text{def}}{=} \frac{\mathbf{v}}{\|\mathbf{v}\|}.$$

Rotation matrix $\mathbf{R}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ can be computed using the algorithm presented in [141], and recalled in Appendix C. Then, after performing the rotation, we perform a dilatation of $\|\mathbf{g}-\mathbf{x}_0\|/\|\mathbf{g}^\star-\mathbf{x}_0^\star\|$ obtaining the transformation matrix

$$\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \frac{\|\mathbf{g}-\mathbf{x}_0\|}{\left\|\mathbf{g}^\star-\mathbf{x}_0^\star\right\|}\mathbf{R}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}. \tag{2.29}$$

We remark that this method cannot be performed when starting and ending points coincide, neither in the learned nor in the new desired behavior. Indeed, if the learned relative position $\mathbf{g}^\star-\mathbf{x}_0^\star$ is zero, we would get a division by zero in (2.29). On the other hand, if the desired relative position $\mathbf{g}-\mathbf{x}_0$ is zero, $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ would become the zero matrix, resulting in a system without evolution.

We remark that both the "old" DMP formulation (2.9) and the "new" one (2.10) can be written as particular cases of extended DMPs (2.28). The latter is trivially obtained by setting $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ to be the identity matrix $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \mathbf{I}\mathrm{d}$. The former is obtained by defining $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ as the diagonal matrix

$$\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star} = \begin{bmatrix} \frac{1}{K_1}\frac{g^{(1)}-x_0^{(1)}}{g^{\star(1)}-x_0^{\star(1)}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{K_2}\frac{g^{(2)}-x_0^{(2)}}{g^{\star(2)}-x_0^{\star(2)}} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \frac{1}{K_{d-1}}\frac{g^{(d-1)}-x_0^{(d-1)}}{g^{\star(d-1)}-x_0^{\star(d-1)}} & 0 \\ 0 & \cdots & 0 & 0 & \frac{1}{K_d}\frac{g^{(d)}-x_0^{(d)}}{g^{\star(d)}-x_0^{\star(d)}} \end{bmatrix},$$

where the superscript $\cdot^{(p)}$ denotes the $p-$th component of the vector. Indeed, the component-by-component product $\odot$ in (2.9) between two vectors $\mathbf{u}$ and $\mathbf{v}$ can be written as

$$\begin{aligned} \mathbf{u}\odot\mathbf{v} &= \mathrm{diag}(\mathbf{u})\,\mathbf{v} \\ &= \begin{bmatrix} u_1 & 0 & 0 & \cdots & 0 \\ 0 & u_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{N-1} & 0 \\ 0 & \cdots & 0 & 0 & u_N \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ v_N \end{bmatrix}. \end{aligned}$$

2.6.1. Results

In this Section, we test the robustness gained by Theorem 2.1 by performing experiments on both synthetic and real scenarios.

At first, we investigate the robustness gain ensured by the invariance property under rotation and dilatation of the reference system by considering two examples in which a trajectory is learned and then executed changing the relative position between goal and starting point $\mathbf{g}-\mathbf{x}_0$. In these tests, we do not change $\mathbf{x}_0$ since DMPs are natively translational invariant. We compare the different behaviors obtained with and without the scaling term $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ defined in (2.29).[1]

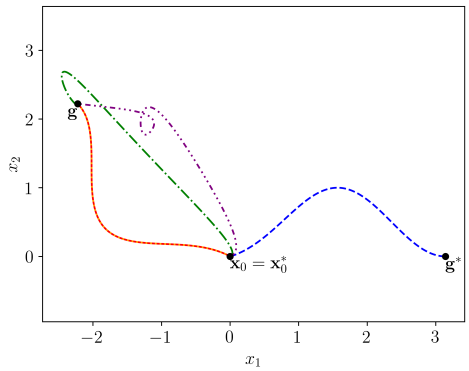
In the following, we refer to *classical* DMPs when talking about the DMPs implementation without exploiting the invariance property (2.10). Similarly, we refer to *extended* DMPs when talking about the DMP formulation (2.28) we introduced in this Section, in which the invariance property is exploited by using, as linear invertible transformation, the roto-dilatation defined in (2.29).
The following tests will show how extended DMPs result in trajectories that are able to properly adapt to arbitrary changes in $\mathbf{g}-\mathbf{x}_0$ from learned and desired behavior. Moreover, these tests will show how extended DMPs result in almost identical behaviors even with a different choice of hyperparameters. In particular, in Example 2.4 we will test the robustness against changes of α in the Canonical System (2.2), while Example 2.5 will show the robustness when changing elastic and damping parameters $\mathbf{K}$ and $\mathbf{D}$.

Example 2.4 (Changing α). In the first example, we generate the desired curve in the plane:

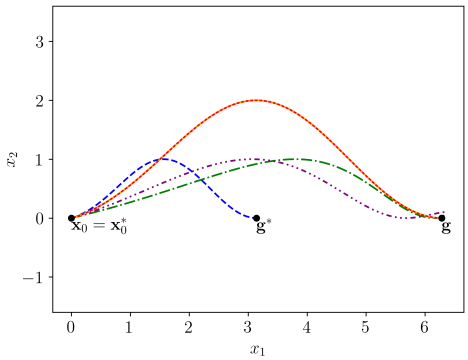
$$\mathbf{x}(t)=\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix}=\begin{bmatrix}t\\\sin^2(t)\end{bmatrix},\quad t\in[0,\pi]. \tag{2.30}$$

Then, we perform the learning step to compute the weights ω_i, and we test the generalization properties of DMPs by changing in different manners the goal position (both in a static and a dynamic way). All tests are executed using both classical (2.10) and extended (2.28) DMPs.
We performed these tests with elastic term $K = 150$, and damping term $D = 2\sqrt{150} \approx 24.49$ for both directions x_1 and x_2, that is $\mathbf{K} = K\mathbf{I}\mathrm{d}_2$ and $\mathbf{D} = D\mathbf{I}\mathrm{d}_2$. We test the generalization by using two different values of α in the canonical system (2.2), $\alpha = 4$ and $\alpha = 2$. Both these values for α result in a canonical system that vanishes at the final time within a tolerance smaller than 1%. Indeed, since the final time is $t_1 = \pi$ in (2.30), for $\alpha = 2$ we have $\exp(-\alpha T) = \exp(-2\pi) \approx 1.9\mathtt{e}-03$; while

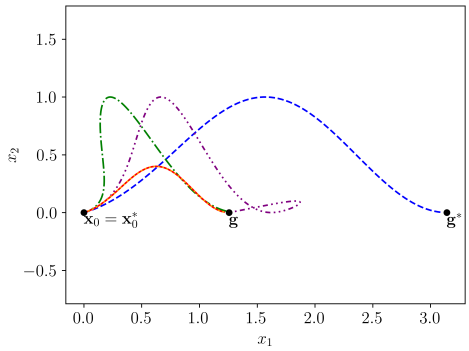
[1] We will not compare the results with the original DMPs (2.9) since the drawbacks of such formulation have already been discussed in the literature [103, 54, 104] and in Section 2.2.



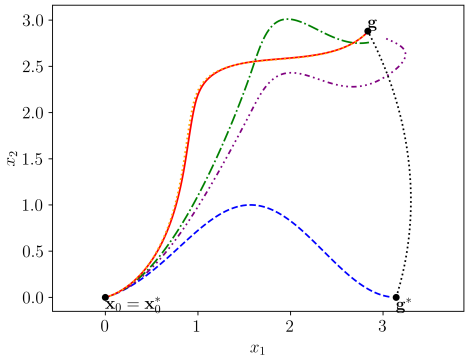
(a) Rotation



(b) Dilatation



(c) Shrinking



(d) Moving goal

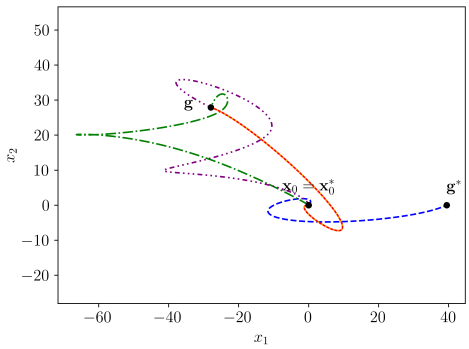
Figure 2.16: Results for Example 2.4 The desired curve is given by (2.30). In all four plots, the desired curve is plotted using the dashed blue line. The execution of extended DMP is shown by the solid red line for $\alpha = 4$, and by the dotted orange line for $\alpha = 2$ (in all these experiments, they overlap). The execution of classical DMPs is marked with the dash-dotted green line when $\alpha = 4$, and with the one-dash-three-dots purple line when α is set to 2. The black dots trial in Figure 2.16d shows the movement of the goal.

for $\alpha = 4$ we have $\exp(-\alpha T) = \exp(-4\pi) \approx 3.5\mathsf{e}-06$. Figure 2.16 shows the results of our tests.

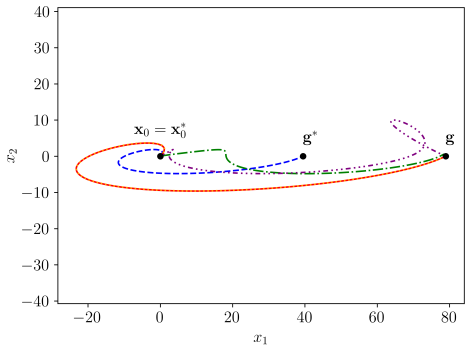
Example 2.5 (Changing $\mathbf{K}$). In the second simulation, the desired curve is:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t^2 \\ \sin(t) \end{bmatrix}, \quad t \in [0, 2\pi]. \tag{2.31}$$

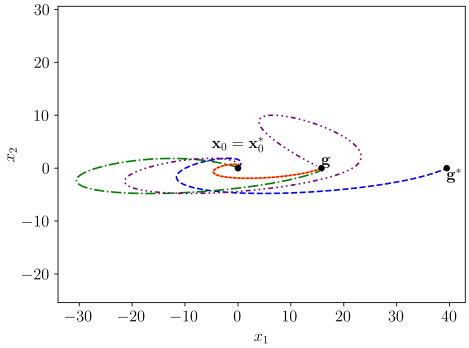
Also in this case we test the generalization properties of both classical DMPs and extended DMPs when the relative position $\mathbf{g} - \mathbf{x}_0$ is changed. The main different is that in this second test we keep the canonical system's decrease rate fixed to $\alpha = 4$ in (2.2), and we test two different values for $\mathbf{K}$ (and, thus, for $\mathbf{D}$). Indeed, we set



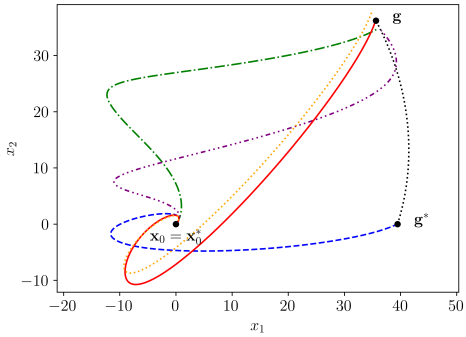
(a) Rotation



(b) Dilatation



(c) Shrinking

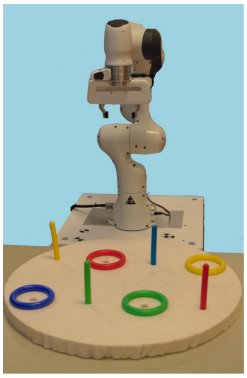


(d) Moving goal

Figure 2.17: Results for Example 2.5 The desired curve is given by (2.31). In all four plots, the desired curve is plotted using the dashed blue line. The execution of extended DMP is shown by the solid red line for $K = 150$, and by the dotted orange line for $K = 15$ (in Figures 2.17a–2.17c, they overlap). The execution of classical DMPs is marked with the dash-dotted green line when $K = 150$, and with the one-dash-three-dots purple line when K is set to 15. The black dots trial in Figure 2.17d shows the movement of the goal.

$\mathbf{K} = K\,\mathbf{I}\mathrm{d}$ and set K to assume value 150 and 15. In both cases, the damping term is set to $\mathbf{D} = \sqrt{K}\,\mathbf{I}\mathrm{d}_2$. Figure 2.17 shows the results of our tests.

These tests show how extended DMPs are more robust than the classical DMP formulation since the trajectories generated by taking advantage of the invariance property have a shape that closely resembles the learned trajectory, while classical DMPs may generate trajectories that do not resemble the learned behavior. Moreover, we notice that the goodness of the generalization of classical DMPs heavily depends on the choice of the parameters. For instance, in the cases of dilatation and moving goal in Example 2.4, they generalize well when $\alpha = 4$, but fails when $\alpha = 2$ (see Figure 2.16b and 2.16d). Similarly, we observe that the generalization of classical DMPs heavily depends also on the choice of parameter $\mathbf{K}$ (and, consequently,



(a) Panda setup.

(b) daVinci setup.

Figure 2.18: Setups of the Peg & Ring task.

D). For instance, it is possible to observe that the trajectories are different, even if the change in goal position is the same, see, for instance, Figure 2.17d. On the other hand, we see that the generalization of the trajectory is robust against the particular choice of hyperparameters when using extended DMPs. Indeed, in almost all our tests, the generalizations obtained by extended DMPs completely overlap when changing goal positions. The only case in which there is an actual difference, even if hardly noticeable, is for the test in Figure 2.17d. This is due to the fact that the scaling matrix $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ depends explicitly on time (since the goal is moving), and different values for $\mathbf{K}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ influence the un-perturbed evolution of dynamical system (2.10). On the other hand, we observe that, in the case presented in 2.16d, there is no difference between the two trajectories, even if $\mathbf{S}^{\mathbf{g}-\mathbf{x}_0}_{\mathbf{g}^\star-\mathbf{x}_0^\star}$ depends on time, because the un-perturbed evolution of the system is not influenced by changes in α.

On a real setup, we test our new formulation by performing a task on a 7 Degrees-of-Freedom (DoF) industrial manipulator, a Panda robot from Franka Emika shown in Figure 2.18a. We will perform two tests. In the first test, the learned behavior will be used to execute the task on the same robot, and we will show how extended DMPs (2.28) result in a better adaptation than classical DMPs (2.10). The second test, instead, will show how extended DMPs (2.28) can be used to *transfer* the learned behavior to a different setup, in our case, the daVinci surgical robot from Intuitive.

The *Peg & Ring* task consists of grabbing, one at a time, four colored rings and moving them to the same-colored peg. The task consists of two gestures, namely `move`, in which the end-effector has to reach the ring, and `carry`, in which the ring is moved toward the peg. Automation of this task on surgical setups is a crucial aspect in Autonomous Robotic Surgery since it presents several challenges of real surgery (e.g. grasping) [44, 45]. During the learning phase, we learn a 3−dimensional DMP

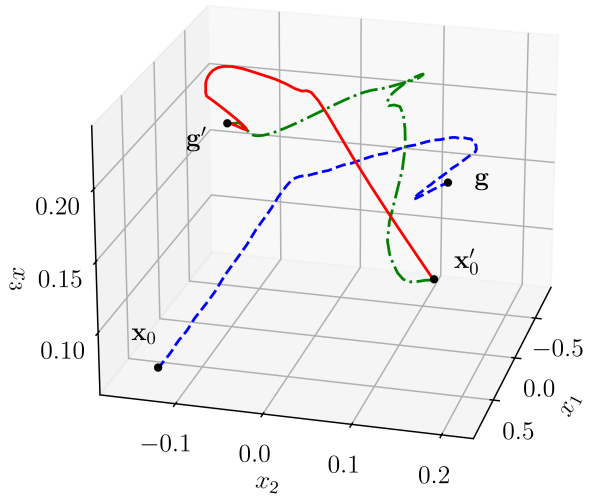


Figure 2.19: Generalization of extended DMPs (2.28). The blue dashed line shows the desired behavior, learned from the `carry` movement for the red ring. The solid red line shows the executed extended DMP (2.28) for the starting and ending points for the `carry` movement for the blue ring. The green dash-dotted line shows the behavior obtained using classical DMPs (2.10) adapted to the critical points of the blue ring.

for both gestures, on the Panda, via kinesthetic teaching. For both tests, DMPs parameters are $\mathbf{K} = K\,\mathbf{I}\mathrm{d}_3$ and $\mathbf{D} = D\,\mathbf{I}\mathrm{d}_3$, with $K = 150$ and $D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. As it can be seen from Figure 2.18a, the four `carry` movements are qualitatively different for each color: the red and blue rings must be moved 'in diagonal' along the base to be put in the corresponding pegs, while the green and yellow rings must be moved 'horizontally'.

To prove the adaptability of extended DMPs, we learn the desired behavior from the execution of the `carry` gesture for the red ring and use it to execute the gesture for all the rings in the scene. Figure 2.19 shows the result of this test. In particular, one can observe that the shape of the executed trajectory (for the blue ring) maintains a trajectory of similar shape to the learned behavior (from the execution for the red ring). For completeness, we plot the behavior obtained with classical DMPs (2.10). From this, it is clear that the behavior of extended DMPs success in maintaining the same shape as the learned behavior, while classical DMPs fail in doing so.

As the second test, we show how extended DMPs allow to 'transfer' the desired behavior between different robotic setups. To do so, we use the DMPs obtained from the execution performed on the Panda industrial manipulator to execute the

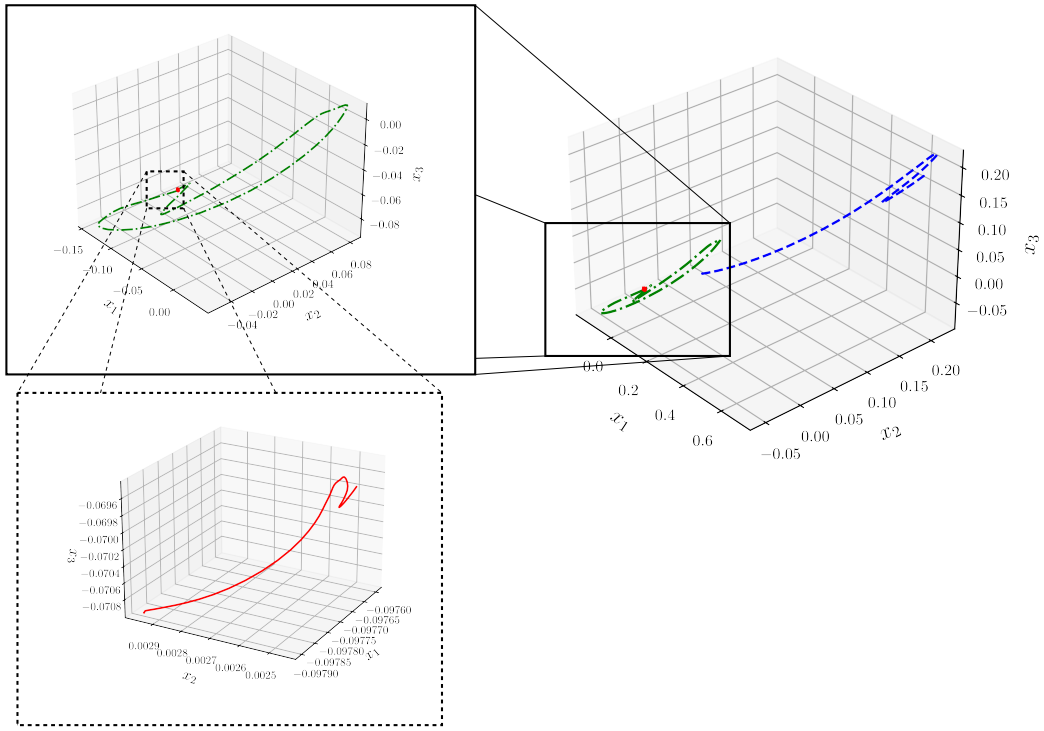


Figure 2.20: Generalization of DMPs between the learned trajectory on the Panda and the executed trajectory on the daVinci surgical robot. The blue dashed line shows the desired (and learned) trajectory, obtained on the Panda industrial manipulator via kinesthetic teaching. The red solid line shows the executed extended DMPs on the daVinci surgical robot. The green dash-dotted line shows the adaptation of classical DMPs to the same starting and goal positions as the red line.

Peg & Ring task on the daVinci surgical robot (which setup can be seen in Figure 2.18b). Figure 2.20 shows the resulting trajectories for this test for an instance of the `move` gesture. Additionally, we plot the trajectory obtained by adapting classical DMPs (2.10) to the new starting and goal positions. This experiment shows that extended DMPs (2.28) are able to generalize to a very different length-scale, while maintaining a shape similar to the learned behavior, effectively permitting to transfer movement from the industrial manipulator to the surgical robot. Moreover, it is possible to observe that the trajectory obtained using classical DMPs fails in adapting to the new length-scale, generating a behavior that cannot be executed with the surgical robot.

2.7. Regression Over Multiple Demonstrations

Traditionally in the literature, DMPs are used as a framework for "one-shot learning", i.e. a Learning from Demonstration approach that requires only one demonstration to learn a behavior.

So-called *Stylistic DMPs* [90, 91] were developed to learn from multiple demonstrations by introducing an additional variable, called *style parameter* ζ, and by making the weights probabilistically dependent on this new variable: $p(\boldsymbol{\omega}|\zeta)$. This approach is useful when the "style" is needed to describe a trajectory, for example, if it is necessary to learn the dependency of the trajectory from the height of an obstacle. However, when the style is not an issue, this approach introduces additional and undesired variables that increase the complexity of the problem.
Other probabilistic approaches for trajectory learning can deal with learning from multiple demonstrations. These approaches include, for instance, *Probabilistic Movement Primitives* [102], *Kernelized Movement Primitives* [57], and *Gaussian Mixture Models* [16]. However, these methods necessarily require multiple trajectories to extract a common behavior, and, differently from DMPs, cannot be used as one-shot learning frameworks. Moreover, the goodness of the obtained behavior is heavily limited by the quality of the dataset. Finally, these methods have not only a probabilistic learning phase but also a stochastic execution. This aspect makes them not completely suitable in some critical scenarios, such as Robotic Minimally Invasive Surgery.

In this Section, we show how to extract a unique set of weights $\boldsymbol{\omega}_i \in \mathbb{R}^d$, $i = 0, 1, \ldots, N$, from a set of multiple observations as a single linear regression problem in the context of DMPs.
Let us consider a set of *M demonstrated trajectories*

$$\left\{\mathbf{x}^{(j)}(t), t \in \left[t_0^{(j)}, t_f^{(j)}\right]\right\}_{j=1,2,\ldots,M}.$$

Starting from the technique introduced in Section 2.6 we can transform each trajectory in such a way that all the starting and ending points are the same (we choose $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{g} = \mathbf{1}$). To do so, we compute the roto-dilatation matrices, for $j = 1, 2, \ldots, M$,

$$\mathbf{S}^{(j)} = \frac{\|\mathbf{1} - \mathbf{0}\|}{\left\|\mathbf{x}^{(j)}(t_f) - \mathbf{x}^{(j)}(t_0)\right\|} \mathbf{R}^{\widehat{\mathbf{1}-\mathbf{0}}}_{\widehat{\mathbf{x}^{(j)}(t_f)-\mathbf{x}^{(j)}(t_0)}}. \tag{2.32}$$

We remark that any linear invertible transformation mapping the starting point $\mathbf{x}_0$ to the origin $\mathbf{0}$ and the goal $\mathbf{g}$ to vector $\mathbf{1}$ can be used. We then use the matrices $\mathbf{S}^{(j)}$ to create the new set of transformed trajectories

$$\left\{\check{\mathbf{x}}^{(j)}(t) = \mathbf{S}^{(j)}\left(\mathbf{x}^{(j)}(t) - \mathbf{x}^{(j)}(t_0)\right)\right\}_{j=1,2,\ldots,M}.$$

In this way, all trajectories $\breve{\mathbf{x}}^{(j)}$ start in $\mathbf{0}$ and ends in $\mathbf{1}$:

$$\breve{\mathbf{x}}^{(j)}\left(t_0^{(j)}\right) = \mathbf{0}, \qquad \breve{\mathbf{x}}^{(j)}\left(t_f^{(j)}\right) = \mathbf{1}.$$

Next, we perform a *time scaling* step, so that the time domain of each trajectory is $[0, T]$ for a fixed $T > 0$. To do so, let us define the curves $\tilde{\mathbf{x}}^{(j)}(t)$ for $j = 1, 2, \ldots, M$ as

$$\tilde{\mathbf{x}}^{(j)}(t) = \breve{\mathbf{x}}^{(j)}\left(\frac{t_f^{(j)} - t_0^{(j)}}{T}\, t + t_0^{(j)}\right). \tag{2.33}$$

It's easy to verify that

$$\tilde{\mathbf{x}}^{(j)}(0) = \breve{\mathbf{x}}^{(j)}\left(t_0^{(j)}\right), \qquad \tilde{\mathbf{x}}^{(j)}(T) = \breve{\mathbf{x}}^{(j)}\left(t_f^{(j)}\right).$$

From these two steps, we obtain a new set of 'transformed' curves $\{\tilde{\mathbf{x}}^{(j)}(t)\}_{j=1}^{M}$, each with time domain $[0, T]$, and $\mathbf{0}$ and $\mathbf{1}$ as starting and ending points respectively:

$$\tilde{\mathbf{x}}^{(j)}(0) = \mathbf{0}, \qquad \tilde{\mathbf{x}}^{(j)}(T) = \mathbf{1}.$$

From (2.13) we compute the set of forcing terms $\{\mathbf{f}^{(j)}\}_{j=1,2,\ldots,M}$, and we are now able to compute the set of weights $\{\boldsymbol{\omega}_i\}_{i=0,1,\ldots,N}$ that minimizes the sum of the squared errors (w.r.t. the L_2-norm) between the function generated using (2.3) and the forcing terms $\mathbf{f}^{(j)}$, $j = 1, 2, \ldots, M$.

For this purpose, we decompose, as for the one-shot learning phase presented in Section 2.3, the problem in each independent component. For each component $p = 1, 2, \ldots, d$, we look for the *weight vector* $\boldsymbol{\omega}^\star = [\omega_0^\star, \omega_1^\star, \ldots, \omega_N^\star]^\intercal \in \mathbb{R}^{N+1}$ satisfying

$$\boldsymbol{\omega}^\star = \arg\min_{\boldsymbol{\omega} \in \mathbb{R}^{N+1}} \underbrace{\sum_{j=1}^{M} \left\| \frac{\sum_{i=0}^{N} \omega_i \varphi_i(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, s - f^{(j)}(s) \right\|_2^2}_{F(\boldsymbol{\omega})}.$$

The existence and uniqueness of a minimum for F is guaranteed by its continuity and strict convexity. Moreover, since F is smooth, $\boldsymbol{\omega}^\star$ is the vector that nullify its gradient: $\nabla_{\boldsymbol{\omega}} F(\cdot)|_{\boldsymbol{\omega}^\star} = \mathbf{0}$. Let us denote with G the gradient of F: $G(\cdot) \doteq \nabla_{\boldsymbol{\omega}} F(\cdot) : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$. Its h-th component is (recalling that $\|\zeta(s)\|_2^2 = \int_{s_0}^{s_1} (\zeta(s))^2\, ds$)

$$\begin{aligned} G_h(\boldsymbol{\omega}) &= \frac{\partial F}{\partial \omega_h}(\boldsymbol{\omega}) \\ &= \sum_{j=1}^{M} \int_{s_0}^{s_1} 2\left(\frac{\sum_{i=0}^{N} \omega_i \varphi_i(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, s - f^{(j)}(s) \right)\left(\frac{\varphi_h(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, s \right) ds \end{aligned}$$

Function G is linear in $\boldsymbol{\omega}$, thus the minimization problem can be written as a linear system: $\mathbf{A}\,\boldsymbol{\omega} = \mathbf{b}$. The component in row h and column k of $\mathbf{A}$, a_{hk}, is, for $h,k \in \{0,1,\dots,N\}$,

$$\begin{aligned} a_{hk} &= \frac{\partial G_h}{\partial \omega_k}(\omega) \\ &= \sum_{j=1}^{M} \int_{s_0}^{s_1} 2\frac{\varphi_h(s)\varphi_k(s)}{\left(\sum_{i=0}^{N}\varphi_i(s)\right)^2}\, s^2\, ds \\ &= 2M \int_{s_0}^{s_1} \frac{\varphi_h(s)\varphi_k(s)}{\left(\sum_{i=0}^{N}\varphi_i(s)\right)^2}\, s^2\, ds, \end{aligned} \tag{2.34}$$

while the h-th component of the vector $\mathbf{b}$, b_h, is, for $h = 0,1,\dots,N$,

$$b_h = \sum_{j=1}^{M} \int_{s_0}^{s_1} 2\left(\frac{\varphi_h(s)}{\sum_{i=0}^{N}\varphi_i(s)}\, f^{(j)}(s)\, s\, ds\right). \tag{2.35}$$

Using any quadrature formula (e.g. Simpson's rule), the integrals in equations (2.34) and (2.35) can be computed. Thus, we can solve the linear problem and obtain $\boldsymbol{\omega}^\star$. We remark that this procedure has to be repeated for each direction $p = 1,2,\dots,d$.

The algorithm to perform regression is summarized in Algorithm 2.2. We remark that, since matrix $\mathbf{A}$ does not depend on the forcing term $f(s)$ but only on the choice of basis functions $\{\varphi_i(s)\}_{i=0,1,\dots,N}$, we can compute it once for all before looping along the directions $p = 1,2,\dots,d$.

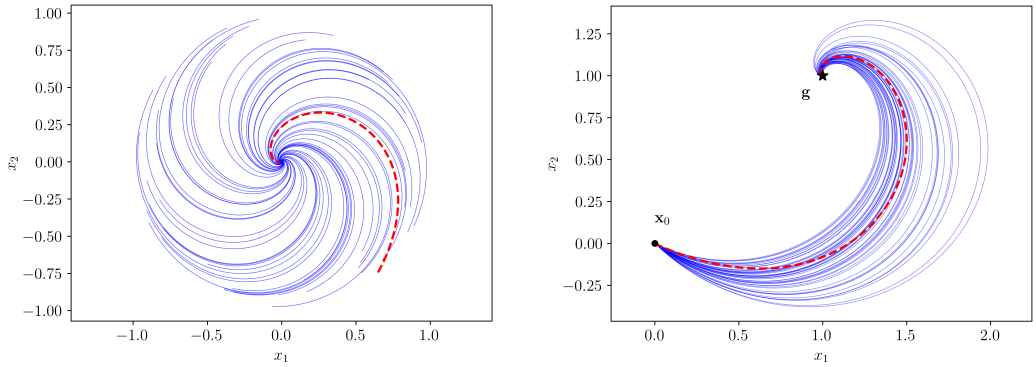
2.7.1. Results

To test the regression method for DMPs, we present both a synthetic example and a test on a real robot.

Example 2.6. In this example, we generate the trajectories computing the solution of the dynamical system

$$\begin{cases} \dot{x}_1 = x_1^3 + x_2^2\, x_1 - x_1 - x_2 \\ \dot{x}_2 = x_2^3 + x_1^2\, x_2 + x_1 - x_2 \end{cases}, \tag{2.36}$$

which is known to have a *alpha-limit* on the circumference of radius 1 and an attractive equilibrium at the origin. The set of (fifty) trajectories is generated by choosing a random angle $\theta_0 \in [0, 2\pi)$ and a random radius $\rho_0 \in (0.8, 1)$, and then setting as initial position $\mathbf{x}(0) = [x_1(0), x_2(0)]^\intercal = [\rho_0\cos(\theta_0), \rho_0\sin(\theta_0)]^\intercal$. Then, the dynamical system is integrated using the *classic fourth-order Runge-Kutta method* [73, 14] up to a random final time $T \in (5, 10)$. Choosing a final time greater than 5 allows



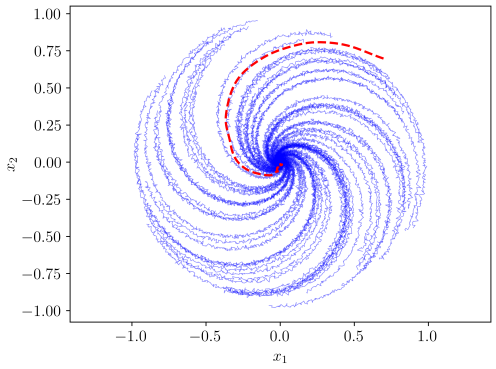
(a) Synthetic test before rescaling.

(b) Synthetic test after rescaling.

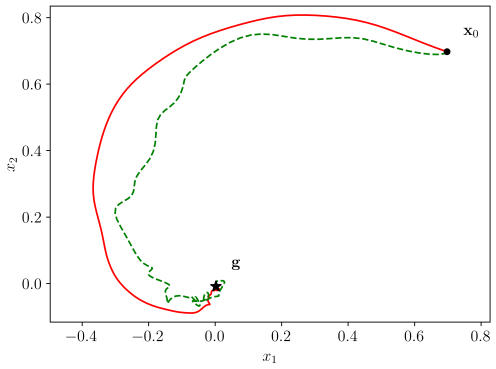
Figure 2.21: Tests for our proposed approach to perform DMPs learning from multiple trajectories. The trajectories are obtained by integrating (2.36) with different initial conditions. In both plots, the solid blue lines represent the demonstrated trajectories, while the dashed red one is an execution of the learned DMP. The tests are plotted both before (Figure 2.21a) and after (Figure 2.21b) the spatial scaling step.

the system to reach the origin with an error smaller than 2%: $\|\mathbf{x} - \mathbf{0}\| < 0.02$. The maximum possible value for T, $T = 10$ has been chosen so that the demonstrated trajectories have different time domains, to show the capability of the regression algorithm to extract a common behavior even when the demonstrated trajectories have different time scales. After generating the set of observations, we use Algorithm 2.2 to learn a DMP. The DMP's hyper-parameters are $\mathbf{K} = K\,\mathbf{I}\mathrm{d}_2$, $\mathbf{D} = D\,\mathbf{I}\mathrm{d}_2$, with $K = 150$ and $D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. Finally, we randomly select an initial point $\mathbf{x}_0 : \|\mathbf{x}_0\| \in (0.8, 1)$ and execute the obtained DMP by setting as goal the attractive equilibrium of the system: $\mathbf{g} = (0, 0)$.
Figure 2.21 shows the results of these tests. In particular Figure 2.21a shows the set of trajectories obtained by integrating the ODE (2.36), together with an execution of the obtained DMP. Figure 2.21b shows the same trajectories when spatially rescaled to have $\mathbf{0}$ and $\mathbf{1}$ as starting and ending points respectively.
To show the importance of performing regression, we add Gaussian noise to the set of observations. The obtained, noisy, data-set is shown in Figure 2.22a

To emphasize the fact that performing regression reduces undesired oscillations, we perform a similar comparison on the JIGSAW [39] dataset. In particular, we extract all the occurrences of the gesture 'G1' (*reaching needle with the right hand*) and use them to extract a unique DMP via Algorithm 2.2. DMPs' parameters are $K = 150, D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. Since this gesture involves only the right arm, we extract only the Cartesian components for it and we ignore the left end-



(a) Noisy data-set.



(b) Comparison of two DMPs obtained by the noisy data-set.

Figure 2.22: Tests performed added Gaussian noise to the dataset. On the left, the blue solid lines show the trajectories of the data-set, while the red dashed line shows the execution of the obtained DMP. On the right, the solid red line shows the trajectory obtained using the proposed method for regression over multiple observations. The dashed green line shows the execution of a DMP learned from one sample of the dataset.

effector. As it can be seen in Figure 2.23, in particular from the second component x_2, oscillations are reduced when a unique behavior is extracted from multiple demonstrations.

For the experiments on a real setup, we executed the Peg & Ring task on a Panda industrial manipulator. The main phases of the task are presented in Figure 2.24: the robot must reach for the green ring, grab it, carry it towards the green peg, and release it. The task can be decomposed into two gestures, namely `move` and `carry`. The first gesture is executed to reach the grasping point of the ring with the robot end-effector. The second gesture, instead, is used to carry the ring toward the same colored peg. The task uses four colored rings (and, thus, four same-colored pegs). We execute the task a total of fifty times on the Panda robot, changing the grasping and releasing positions (i.e. the starting and final positions of each gesture), thus obtaining a total of 200 samples for both the `move` and `carry` gestures.

Figure 2.25a and 2.25b show the trajectories of, respectively, `move` and `carry` obtained with the industrial manipulator together with an execution of the obtained DMP with parameters $K = 150, D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. These are plotted after the rescaling so that $\mathbf{x}_0 = \mathbf{0}$, and $\mathbf{g} = \mathbf{1}$ for both gestures.

We then used these DMPs to automate the Peg & Ring task. We implemented a Finite State Machine which takes as input the order of colors and then automatically executes the task by alternating the `move` and `carry` gestures. The learned DMPs

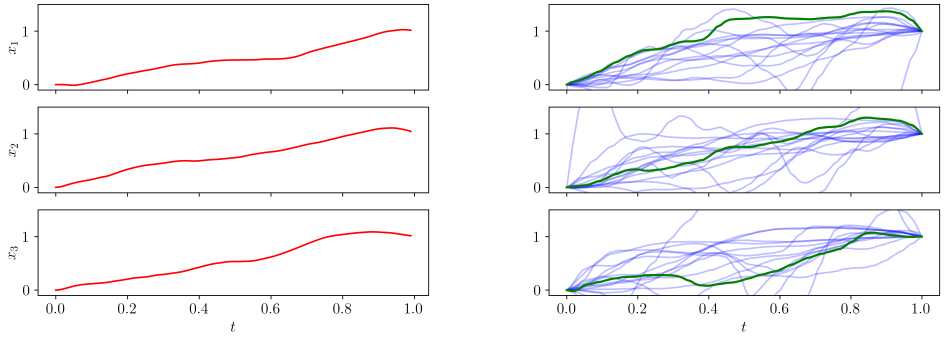
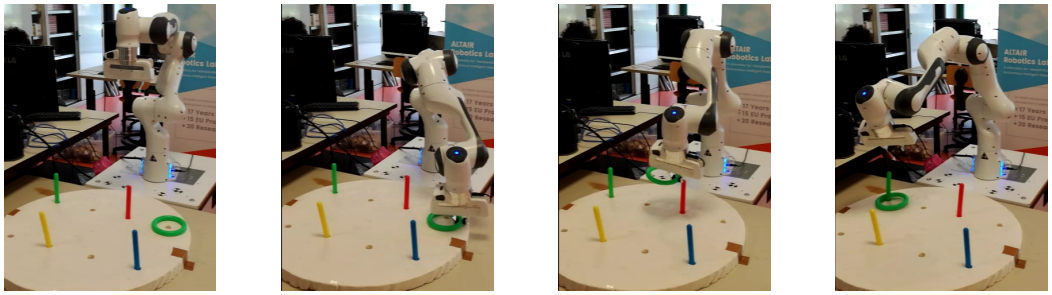


Figure 2.23: Comparison between two DMPs obtained by a real gesture. The solid red line (on the left) shows the execution of the DMP obtained via regression. The dataset is shown on the right in blue. The solid green line (on the right) shows the DMP extracted from a single demonstration.



(a) Initial setup. (b) Grasped ring. (c) Carrying the ring. (d) Released ring.

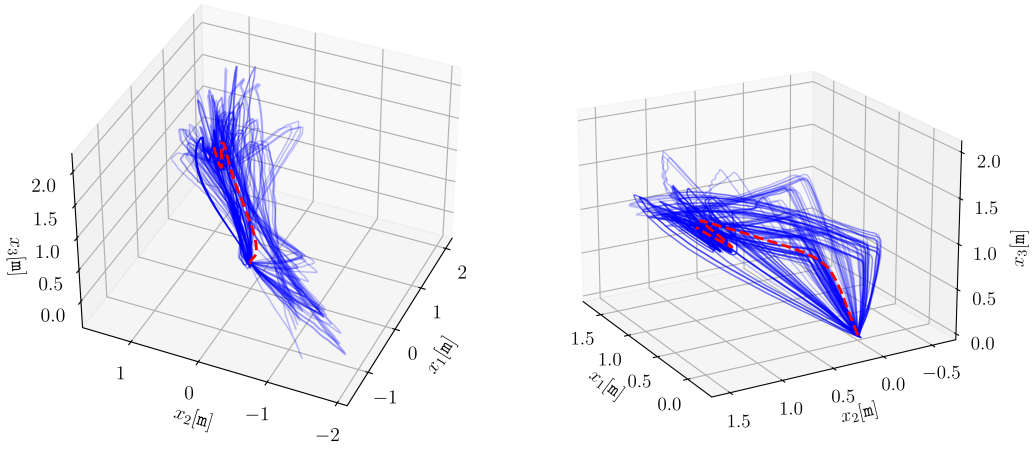
Figure 2.24: The Peg&Ring task with the Panda robot.

are used to model the Cartesian evolution of the robot's end-effector of the robot. In Figure 2.26 the execution of both gestures for the green ring is shown.
These tests show the capability of Algorithm 2.2 to extract a common behavior from multiple demonstrations.

2.8. Conclusions

In this Chapter, we presented the Dynamic Movement Primitives framework, highlighting the properties that make them useful in robotic trajectory imitation. In particular, we showed how DMPs are able to learn a behavior and to replicate it generalizing both spatially, by adapting to new starting and ending positions, and temporally, by allowing to slow down or speed up the execution by changing a single parameter.

We presented several improvements to the state-of-the-art. In particular, we



(a) `move` (b) `carry`

Figure 2.25: Experiments on the real robot. For both gestures (`move` and `carry`), the trajectories of the dataset are plotted using solid lines, while the execution of the obtained DMP is plotted using a dashed line. All the trajectories are plotted after the rescaling, so that $\mathbf{x}_0 = \mathbf{0}$, and $\mathbf{g} = \mathbf{1}$ for both gestures.

presented and compared multiple classes of basis functions to encode the desired behavior, showing the strengths and weaknesses of each of them. Moreover, we showed how to improve the spatial scalability of the original framework, by making DMPs invariant under affine transformations of the reference frame, focusing on roto-dilatations. Finally, we presented an algorithm to learn a unique behavior from a set of multiple demonstrations.

In this Chapter, we tested the DMP framework on an industrial manipulator, and not on a surgical setup. Later, in Chapter 3 we will implement obstacle avoidance within the DMP framework, and test the DMPs with the daVinci surgical robot. Then, in Chapter 6, we will use DMPs to automate the Peg & Ring task on the daVinci surgical robot by creating a framework for automating surgical-related tasks.

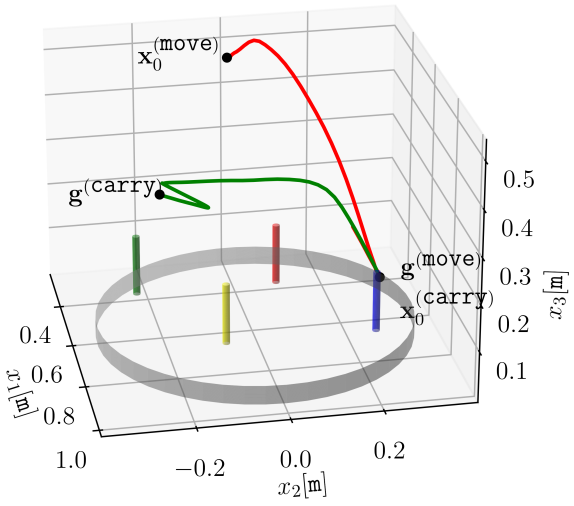


Figure 2.26: Plot of the `move` (in red) and `carry` (in green) gesture in an automatic execution of the Peg & Ring task on the Panda robot.

Algorithm 2.2 Regression Over Multiple Demonstrations

Input: Set of desired trajectories $\{\mathbf{x}^{(j)}(t)\}_{j=1,2,\dots,M}$, set of basis functions $\{\varphi_i(s)\}_{i=0,1,\dots,N}$, DMPs hyperparameter $\mathbf{K}, \mathbf{D}$ and α, final time $T > 0$;

Output: Set of weights $\{\boldsymbol{\omega}^{(p)}\}_p$ for each direction $p = 1, 2, \dots, d$.

▷ Align the trajectories (both temporally and spatially)

for $j = 1, 2 \dots, M$ **do**
  Compute $\mathbf{S}^{(j)}$ using (2.32)
  Compute $\check{\mathbf{x}}^{(j)}(t) = \mathbf{S}^{(j)}\mathbf{x}(t)$
  Compute $\tilde{\mathbf{x}}^{(j)}(t)$ using (2.33)
end for
Evaluate the canonical system extrema $s_0 = 1, s_1 = \mathrm{e}^{-\alpha T}$
▷ Compute the matrix $\mathbf{A} = [a_{hk}]_{h,k=0,1,\dots,N}$
for $h = 0, 1, \dots, N$ **do**
  $a_{hh} = 2M \int_{s_0}^{s_1} \frac{\varphi_h(s)^2}{\left(\sum_{i=0}^{N} \varphi_i(s)\right)^2} s^2 \, ds$
  for $k = h + 1, h + 2, \dots, N$ **do**
    $a_{hk} = 2M \int_{s_0}^{s_1} \frac{\varphi_h(s)\varphi_k(s)}{\left(\sum_{i=0}^{N} \varphi_i(s)\right)^2} s^2 \, ds$
    $a_{kh} = a_{hk}$
  end for
end for
▷ For loop along the directions
for $p = 1, 2, \dots, d$ **do**
  For each trajectory $\tilde{\mathbf{x}}^{(j)}(t)$ compute the forcing term $f^{(j)}(s)$ for direction p using (2.13).
  ▷ Compute the vector $\mathbf{b} = [b_h]_{h=0,1,\dots,N}$
  for $h = 0, 1, \dots, N$ **do**
    $b_h = \sum_{j=1}^{M} \int_{s_0}^{s_1} 2 \frac{\varphi_h(s)}{\sum_{i=0}^{N} \varphi_i(s)} f^{(j)}(s) \, s \, ds$
  end for
  Solve the minimization problem: $\mathbf{A}\boldsymbol{\omega}^{(p)} = \mathbf{b}$
end for

CHAPTER 3

Obstacle Avoidance Methods for DMPs

In robotics, both mobile and manipulative, obstacle avoidance is a crucial topic: autonomous robots should be able to quickly and safely adapt to the presence of an obstacle, so to avoid collisions that would damage the robot itself or people around it. In surgical applications, this is a particularly critical aspect since a collision may result in organ damage.

Obstacle avoidance for DMPs can be accounted for in many ways. *Stylistic DMPs* [90, 91] offer a way to learn a probabilistic distribution of the weights as function of a 'style' parameter ζ. Such a parameter can be, for instance, the height of an obstacle. In this way, the DMP can adapt to the presence of obstacles as long as the scenarios are similar between the learning and the execution phase. Even other methods different from Stylistic DMPs, such as learning methods [110, 111, 125], require multiple demonstrations with different types and sizes of the obstacles.

These "learning-based" methods are not suitable for scenarios in which is not possible to obtain a lot of examples. Since in Minimally Invasive Surgery (MIS) it is usually hard to obtain data, hand-crafted obstacle avoidance methods are preferable since they do not require any additional learning phase and are readily available to be utilized in real scenarios.

Moreover, the obstacle-avoidance behavior obtained from the learning methods heavily depends on the quality of the data used to learn the method. On the other hand, the behavior of the 'hand-crafted' obstacle-avoidance methods presented in this Chapter depends only on the choice of a few hyper-parameters, that can be easily tuned.

Due to the second-order ODE formulation of DMPs, the most natural way to implement obstacle avoidance is to add a repulsive term $\mathbf{p} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$, $(\mathbf{x}, \mathbf{v}) \mapsto$

$\mathbf{p}(\mathbf{x}, \mathbf{v})$ to the differential equation (2.10a), that now reads [103]

$$\tau \dot{\mathbf{v}} = \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g} - \mathbf{x}_0) + \mathbf{K}\mathbf{f}(s) + \mathbf{p}(\mathbf{x}, \mathbf{v}). \tag{3.1}$$

The term $\mathbf{p}(\mathbf{x}, \mathbf{v})$ is a perturbation term that is used to "push" the trajectory away from the obstacle and is often written as the negative gradient of a *potential field*. The usage of potential fields in path planning and obstacle avoidance is not new in robotics [59, 109, 135, 13] and it has been proven effective also in the DMP framework.

In this Chapter, we present different strategies to define the repulsive term $\mathbf{p}(\mathbf{x}, \mathbf{v})$. In Section 3.1 we present three methods to model point obstacles. Since they are limited to treat point-mass like obstacles, these methods may result in odd obstacle avoidance behaviors when a real obstacle, with a shape and a volume, has to be avoided. In Section 3.2 we propose two methods to treat volumetric obstacle avoidance. These new methods allow treating a solid obstacle as a whole, instead of as a mesh of point-wise obstacles. In Section 3.3 all these methods (both for point and solid obstacles) are tested and compared.

3.1. Point Obstacles

In this Section, we introduce three methods for obstacle avoidance for point obstacles. In particular, in Section 3.1.1 we present a static potential method, in Section 3.1.2 a dynamic potential method, and, in Section 3.1.3, a steering angle strategy.

3.1.1. Static-Potential Point Obstacle

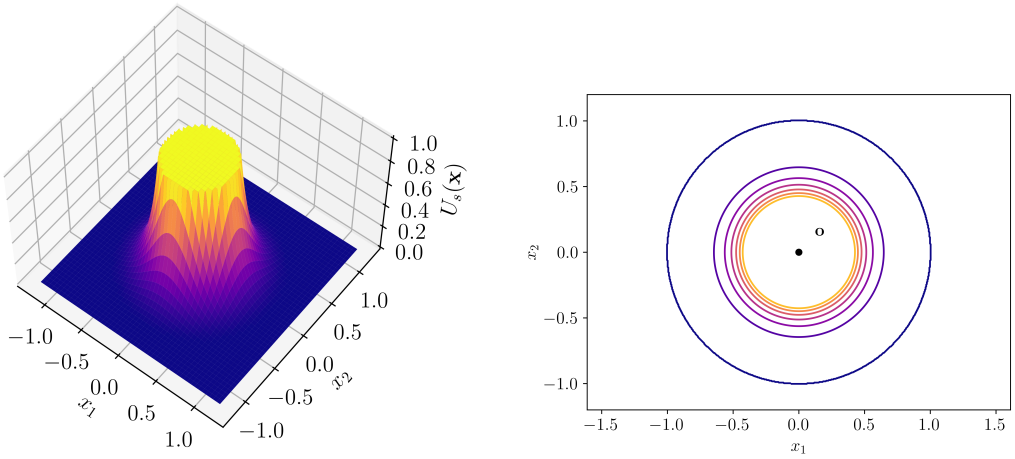
The first, simplest, method to implement obstacle-avoidance in the context of DMPs that we present is a *static potential method*. With this method, each point obstacle generates a potential field, whose negative gradient is the perturbation term $\mathbf{p}$. The term 'static' refers to the fact that the potential (and, thus, the perturbation term) depends only on the position of the system, and not on its velocity $\mathbf{p}(\mathbf{x}, \mathbf{v}) \equiv \mathbf{p}(\mathbf{x})$.

In [68] the following potential is defined:

$$U_s(\mathbf{x}) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{if } \rho(\mathbf{x}) > \rho_0 \end{cases}, \tag{3.2}$$

where $\eta \in \mathbb{R}_+$ is a constant gain, $\rho_0 \in \mathbb{R}_+$ is the influence radius of the obstacle, and $\rho(\mathbf{x}) \in \mathbb{R}_+$ is the distance between the obstacle and the system's position,

$$\rho(\mathbf{x}) = \|\mathbf{x} - \mathbf{o}\|,$$



(a) Mesh plot.

(b) Isocontour plot.

Figure 3.1: Example of the static potential $U_s(\mathbf{x}, \mathbf{v})$ given in (3.2) for a point obstacle $\mathbf{o} = [0, 0]^\intercal \in \mathbb{R}^2$. The gain is set to $\eta = 1$. In both figures, the potential has been cropped at the value of 1 for display purposes: it goes to infinity at the obstacle position.

where $\mathbf{o} \in \mathbb{R}^d$ denotes the position of the obstacle. Figure 3.1 shows an example of potential $U_s(\mathbf{x})$. The perturbation term $\mathbf{p}(\mathbf{x}, \mathbf{v})$ in (3.1) is defined, for this method, as the negative gradient of the potential itself,

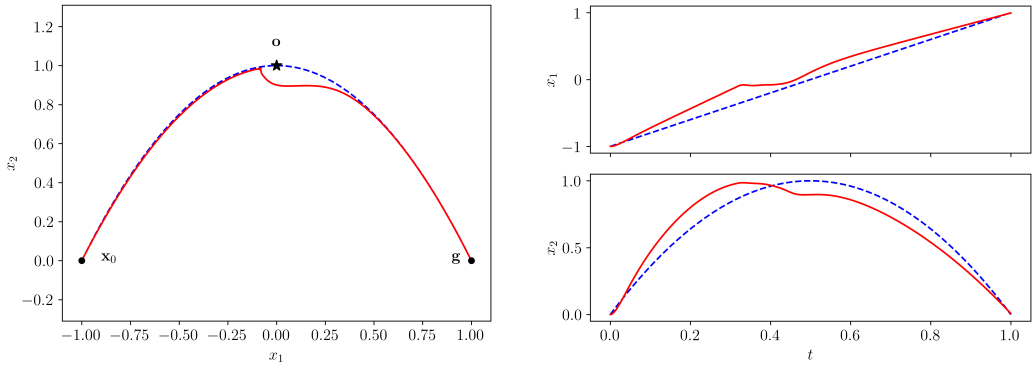
$$\mathbf{p}(\mathbf{x}, \mathbf{v}) = \mathbf{p}_s(\mathbf{x}) = -\nabla_{\mathbf{x}} U_s(\mathbf{x}). \tag{3.3}$$

We now compute the gradient in (3.3). Clearly, for $\rho(\mathbf{x}) > \rho_0$ we have $\nabla_{\mathbf{x}} U_s(\mathbf{x}) = 0$. For $\rho(\mathbf{x}) \leq \rho_0$ we compute

$$\begin{aligned}
\mathbf{p}_s(\mathbf{x}) &= -\nabla_{\mathbf{x}} \left(\frac{\eta}{2} \left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0} \right)^2 \right) \\
&= -\eta \left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0} \right) \nabla_{\mathbf{x}} \left(\frac{1}{\rho(\mathbf{x})} \right) \\
&= -\eta \left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0} \right) \left(\frac{-1}{\rho^2(\mathbf{x})} \right) \nabla_{\mathbf{x}} (\rho(\mathbf{x})).
\end{aligned} \tag{3.4}$$

The gradient $\nabla_{\mathbf{x}} \rho(\mathbf{x})$ of the distance from the obstacle $\rho(\mathbf{x})$ in (3.4) is computed as follows:

$$\begin{aligned}
\nabla_{\mathbf{x}} (\rho(\mathbf{x})) &= \nabla_{\mathbf{x}} \|\mathbf{x} - \mathbf{o}\| \\
&= \nabla_{\mathbf{x}} \left(\sqrt{\sum_j (x_j - o_j)^2} \right) \\
&= \frac{1}{2\sqrt{\sum_j (x_j - o_j)^2}} 2 (\mathbf{x} - \mathbf{o})
\end{aligned}$$



(a) Trajectory. (b) Solution.

Figure 3.2: Obstacle avoidance behavior using the static potential for point obstacles (3.3). In both plots, the blue dashed line shows the desired (learned) curve, while the solid red line shows the adaptation of the DMP to the presence of the obstacle (marked with a black star).

$$= \frac{1}{2\rho(\mathbf{x})} 2(\mathbf{x} - \mathbf{o}). \tag{3.5}$$

Thus, perturbation term for potential (3.2) reads

$$\mathbf{p}_s(\mathbf{x}) = \begin{cases} \eta\left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0}\right)\frac{1}{\rho^3(\mathbf{x})}(\mathbf{x} - \mathbf{o}) & \text{if } \rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{if } \rho(\mathbf{x}) > \rho_0 \end{cases}. \tag{3.6}$$

Example 3.1. To show the behavior of this method for obstacle avoidance, we learn a DMP on the desired trajectory

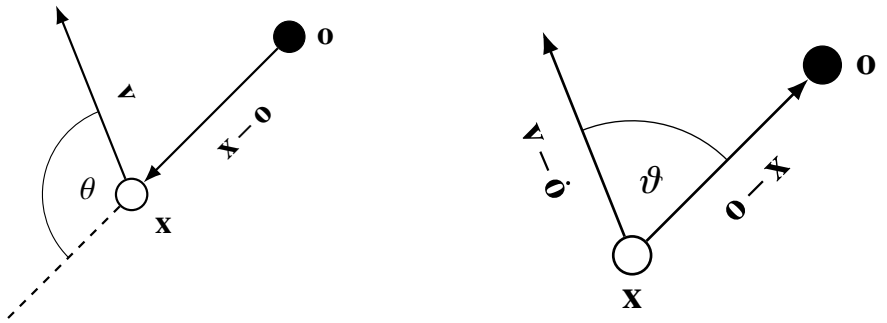
$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t \\ 1 - t^2 \end{bmatrix}, \qquad t \in [-1, 1].$$

Then, we place a point obstacle in $\mathbf{o} = [0, 1]^\intercal$. The parameters of the potential function $U_s(\mathbf{x})$ in (3.2) are $\rho_0 = 0.1$ and $\eta = 1$. Figure 3.2 shows the adaptation of the DMP to the presence of the obstacle.

3.1.2. Dynamic-Potential Point Obstacle

In [103] was pointed out that the perturbation term (3.6) may result in non-smooth obstacle behaviors. This can be seen, for instance, in Figure 3.2. In particular, the first component of the solution (top plot in Figure 3.2b) shows a sharp change in velocity near time $t = 0.3$.

In general, it is desirable to avoid sudden changes in direction in robotics. Indeed, a smooth trajectory is preferable since it reduces both the robot's energy consumption and the damage done to the actuators [40]. In [103] it was pointed out that a



(a) Angle θ defined in (3.8). (b) Steering angle ϑ defined in (3.11).

Figure 3.3: Depiction of angles θ and ϑ defined in (3.8) and (3.11) respectively. We remark that the two angles are complementary. Thus, the cosines are opposite: $\cos\theta = -\cos\vartheta$.

potential depending also on the velocity $\mathbf{v}$ of the system and not only on its position $\mathbf{x}$ would result in smoother obstacle avoidance behaviors. Thus, the following 'dynamic' (i.e. velocity-dependent) potential is proposed [103]

$$U_d(\mathbf{x},\mathbf{v}) = \begin{cases} \lambda(-\cos\theta)^\beta \dfrac{\|\mathbf{v}\|}{\rho(\mathbf{x})} & \text{if } \theta \in \left(\frac{\pi}{2},\pi\right] \\ 0 & \text{if } \theta \in \left[0,\frac{\pi}{2}\right] \end{cases}, \tag{3.7}$$

where $\lambda,\beta \in \mathbb{R}_+$ are constant gains, and θ, depicted in Figure 3.3a, is the angle taken between the current velocity $\mathbf{v}$ and the system's position $\mathbf{x}$ relative to the position $\mathbf{o}$ of the obstacle,

$$\cos\theta = \frac{\langle \mathbf{v}, \mathbf{x}-\mathbf{o}\rangle}{\|\mathbf{v}\|\rho(\mathbf{x})}, \tag{3.8}$$

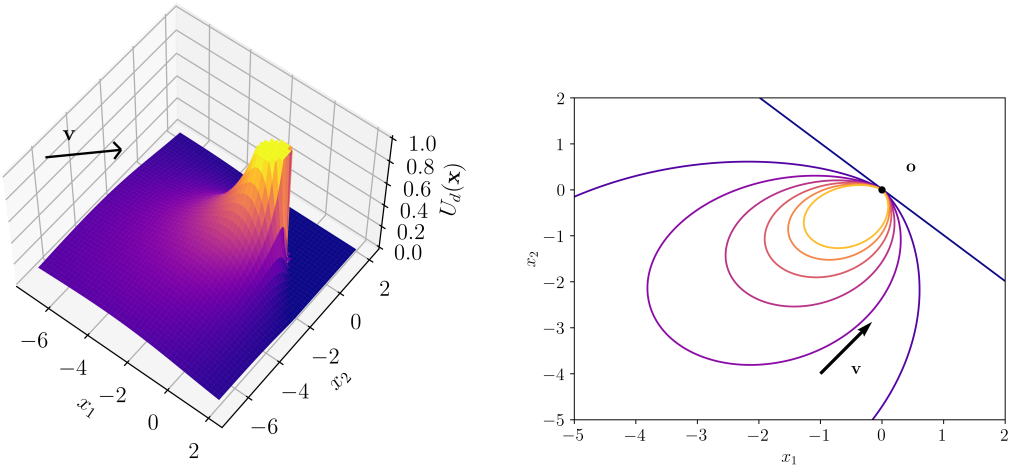
where $\rho(\mathbf{x})$ still denotes the distance between the system's position $\mathbf{x}$ and the obstacle $\mathbf{o}$, $\rho(\mathbf{x}) = \|\mathbf{x}-\mathbf{o}\|$. The potential U_d in (3.7) is set to zero when $\theta \in [0,\pi/2]$ so that the DMP's evolution is not influenced by the presence of the obstacle if the system is already 'going away' from the obstacle. Figure 3.4 shows an example of potential $U_d(\mathbf{x},\mathbf{v})$.

For potentials depending on both position $\mathbf{x}$ and velocity $\mathbf{v}$ of the system, the perturbation term is defined as the negative gradient w.r.t. the position:

$$\mathbf{p}(\mathbf{x},\mathbf{v}) = \mathbf{p}_d(\mathbf{x},\mathbf{v}) = -\nabla_{\mathbf{x}} U_d(\mathbf{x},\mathbf{v}).$$

Clearly, for $\theta \in [0,\pi/2]$, the perturbation term is null, $\mathbf{p}_d(\mathbf{x},\mathbf{v}) = 0$. Let us now compute it for $\theta \in (\pi/2,\pi]$:

$$\mathbf{p}_d(\mathbf{x},\mathbf{v}) = -\nabla_{\mathbf{x}}\left(\lambda(-\cos\theta)^\beta \frac{\|\mathbf{v}\|}{\rho(\mathbf{x})}\right).$$



(a) Mesh plot.

(b) Isocontour plot.

Figure 3.4: Example of the dynamic potential $U_s(\mathbf{x}, \mathbf{v})$ given in (3.7) for a point obstacle $\mathbf{o} = [0,0]^\intercal \in \mathbb{R}^2$. The velocity is set to $\mathbf{v} = [1,1]^\intercal \in \mathbb{R}^2$. The gains are set to $\lambda = 1, \beta = 2$. In both figures, the potential has been cropped at the value of 1 for display purposes: it goes to infinity at the obstacle position.

By the product rule of derivation,

$$\mathbf{p}_d(\mathbf{x}, \mathbf{v}) = -\lambda \|\mathbf{v}\| \left(\frac{1}{\rho(\mathbf{x})} \nabla_\mathbf{x} \left((-\cos\theta)^\beta \right) + (-\cos\theta)^\beta \nabla_\mathbf{x} \left(\frac{1}{\rho(\mathbf{x})} \right) \right).$$

Next, we compute the gradients in the brackets, obtaining

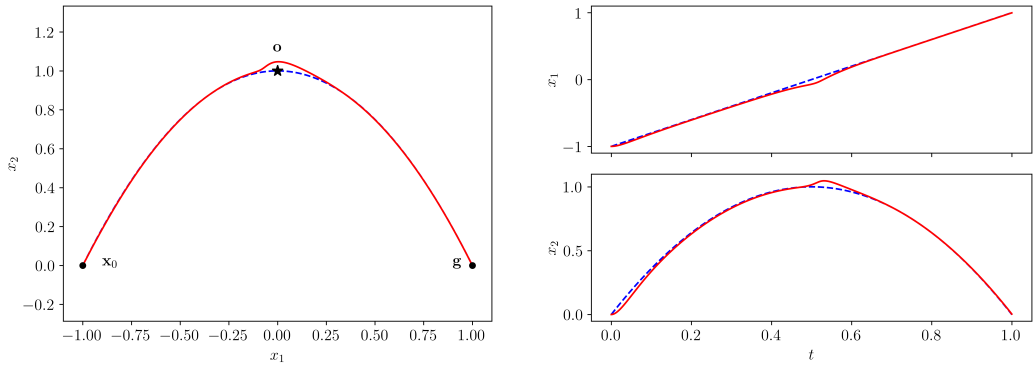
$$\begin{aligned}
\mathbf{p}_d(\mathbf{x}, \mathbf{v}) &= -\lambda \|\mathbf{v}\| \left(\frac{\beta(-\cos\theta)^{\beta-1}}{\rho(\mathbf{x})} \nabla_\mathbf{x}(-\cos\theta) - \frac{(-\cos\theta)^\beta}{\rho^2(\mathbf{x})} \nabla_\mathbf{x}(\rho(\mathbf{x})) \right) \\
&= -\frac{\lambda \|\mathbf{v}\| (-\cos\theta)^{\beta-1}}{\rho(\mathbf{x})} \left(\beta \nabla_\mathbf{x}(-\cos\theta) - \frac{-\cos\theta}{\rho(\mathbf{x})} \nabla_\mathbf{x}(\rho(\mathbf{x})) \right) \\
&= -\frac{\lambda \|\mathbf{v}\| (-\cos\theta)^{\beta-1}}{\rho(\mathbf{x})} \left(-\beta \nabla_\mathbf{x}(\cos\theta) + \frac{\cos\theta}{\rho(\mathbf{x})} \nabla_\mathbf{x}(\rho(\mathbf{x})) \right) \\
&= \frac{\lambda \|\mathbf{v}\| (-\cos\theta)^{\beta-1}}{\rho(\mathbf{x})} \left(\beta \nabla_\mathbf{x}(\cos\theta) - \frac{\cos\theta}{\rho(\mathbf{x})} \nabla_\mathbf{x}(\rho(\mathbf{x})) \right).
\end{aligned}$$

Thus, perturbation term is given by

$$\mathbf{p}_d(\mathbf{x}, \mathbf{v}) = \begin{cases} \lambda(-\cos\theta)^{\beta-1} \dfrac{\mathbf{v}}{\rho(\mathbf{x})} \left(\beta \nabla_\mathbf{x}(\cos\theta) - \dfrac{\cos\theta}{\rho(\mathbf{x})} \nabla_\mathbf{x} \rho(\mathbf{x}) \right) & \text{if } \theta \in \left(\frac{\pi}{2}, \pi \right] \\ 0 & \text{if } \theta \in \left[0, \frac{\pi}{2} \right] \end{cases}. \quad (3.9)$$

Quantity $\nabla_\mathbf{x}(\rho(\mathbf{x}))$ is computed as in (3.5):

$$\nabla_\mathbf{x}(\rho(\mathbf{x})) = \frac{\mathbf{x} - \mathbf{o}}{\rho(\mathbf{x})}.$$



(a) Trajectory. (b) Solution.

Figure 3.5: Obstacle avoidance behavior using the dynamic potential for point obstacles (3.9). In both plots, the blue dashed line shows the desired (learned) curve, while the solid red line shows the adaptation of the DMP to the presence of the obstacle (marked with a black star).

Quantity $\nabla_{\mathbf{x}}(\cos\theta)$ is computed as follows:

$$\begin{aligned}\nabla_{\mathbf{x}}(\cos\theta) &= \nabla_{\mathbf{x}}\left(\frac{\langle \mathbf{v}, \mathbf{x}-\mathbf{o}\rangle}{\|\mathbf{v}\|\rho(\mathbf{x})}\right) \\ &= \frac{1}{\|\mathbf{v}\|\rho(\mathbf{x})^2}(\rho(\mathbf{x})\nabla_{\mathbf{x}}\langle \mathbf{v}, \mathbf{x}-\mathbf{o}\rangle - \langle \mathbf{v}, \mathbf{x}-\mathbf{o}\rangle \nabla_{\mathbf{x}}(\rho(\mathbf{x}))).\end{aligned}$$

Quantity $\nabla_{\mathbf{x}}\langle \mathbf{v}, \mathbf{x}-\mathbf{o}\rangle$ is given by

$$\begin{aligned}\nabla_{\mathbf{x}}\langle \mathbf{v}, \mathbf{x}-\mathbf{o}\rangle &= \left[\frac{\partial}{\partial x_i}\left(\sum_{j=1}^{d} v_j(x_j - o_j)\right)\right]_{i=1,2\ldots,d} \\ &= [v_i]_{i=1,2\ldots,d} \\ &= \mathbf{v}.\end{aligned}$$

Example 3.2. To show the behavior of this method for obstacle avoidance, we learn a DMP on the desired trajectory

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t \\ 1 - t^2 \end{bmatrix}, \qquad t \in [-1, 1].$$

Then, we place a point obstacle in $\mathbf{o} = [0, 1]^\intercal$. The parameters of the potential function $U_d(\mathbf{x}, \mathbf{y})$ in (3.7) are $\lambda = 1$ and $\beta = 2$. Figure 3.5 shows the adaptation of the DMP to the presence of the obstacle.

From this example, we can observe that the dynamic perturbation term (3.9) results in a smoother DMP trajectory. Moreover, the obtained trajectory deviates less from the unperturbed behavior w.r.t. the static perturbation term (3.3).

3.1.3. Steering Angle Method

The next obstacle avoidance method is based on a differential equation that models human obstacle avoidance [34]. This model describes how the angle between the direction of the human and the direction towards the obstacle evolves.

In [54] the following perturbation term was proposed:

$$\mathbf{p}(\mathbf{x},\mathbf{v}) = \mathbf{p}_\vartheta(\mathbf{x},\mathbf{v}) = \gamma\,\mathbf{R}\,\mathbf{v}\,\vartheta\,\exp(-\beta\vartheta), \tag{3.10}$$

where $\gamma,\beta \in \mathbb{R}_+$ are constant gains. The *steering angle* ϑ (depicted in Figure 3.3b) is defined as

$$\vartheta = \arccos\left(\frac{\langle \mathbf{o}-\mathbf{x}, \mathbf{v}\rangle}{\|\mathbf{o}-\mathbf{x}\|\,\|\mathbf{v}\|}\right), \tag{3.11}$$

where $\mathbf{o}$ is the position of the point obstacle.
Matrix $\mathbf{R}$ is defined as the rotation matrix of angle $\pi/2$ w.r.t. the axis generated by $(\mathbf{o}-\mathbf{x})\times\mathbf{v}$, where $\times$ denotes the cross product in $\mathbb{R}^3$. The matrix describing the rotation of angle ϱ around an axis $\mathbf{u} = [u_1, u_2, u_3]$, $\|\mathbf{u}\| = 1$ is given by [126, 22]

$$\mathbf{R} = \begin{bmatrix} \cos\varrho + u_1^2(1-\cos\varrho) & u_1u_2(1-\cos\varrho) - u_3\sin\varrho & u_1u_3(1-\cos\varrho) + u_2\sin\varrho \\ u_2u_1(1-\cos\varrho) + u_3\sin\varrho & \cos\varrho + u_2^2(1-\cos\varrho) & u_2u_3(1-\cos\varrho) - u_1\sin\varrho \\ u_3u_1(1-\cos\varrho) - u_2\sin\varrho & u_3u_2(1-\cos\varrho) + u_1\sin\varrho & \cos\varrho + u_3^2(1-\cos\varrho) \end{bmatrix}.$$

In our case, we set

$$\mathbf{u} = \frac{(\mathbf{o}-\mathbf{x})\times\mathbf{v}}{\|(\mathbf{o}-\mathbf{x})\times\mathbf{v}\|},$$

and $\varrho = \pi/2$, obtaining

$$\mathbf{R} = \begin{bmatrix} u_1^2 & u_1u_2 - u_3 & u_1u_3 + u_2 \\ u_2u_1 + u_3 & u_2^2 & u_2u_3 - u_1 \\ u_3u_1 - u_2 & u_3u_2 + u_1 & u_3^2 \end{bmatrix}.$$

This formulation presents an important advantage and two shortcomings w.r.t. the previous two approaches. The advantage is that this formulation guarantees convergence to the goal position if the obstacles are still.

Theorem 3.1 (Hoffmann et al. [54])**.** *Equation* (3.1) *with* (3.10) *converges to the goal position* $\mathbf{g}$*.*

Proof. Since $s \to 0$ as $t \to \infty$, we need to study the convergence of the reduced equation

$$\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g}-\mathbf{x}) - \mathbf{D}\mathbf{v} + \gamma\mathbf{R}\mathbf{v}\vartheta\exp(-\beta\vartheta),$$

where we set $\tau = 1$ for simplicity.
Let us consider the Lyapunov function candidate $V : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}_+, \quad \mathbf{x}, \mathbf{v} \mapsto V(\mathbf{x}, \mathbf{v})$ defined as the energy of the linear spring system (with unit mass) $\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{D}\mathbf{v}$:

$$V(\mathbf{x}, \mathbf{v}) = \frac{1}{2}(\mathbf{g} - \mathbf{x})^\top \mathbf{K}(\mathbf{g} - \mathbf{x}) + \frac{1}{2}\mathbf{v}^\top \mathbf{v}.$$

To prove convergence we have to prove that $\dot{V} < 0$ for $\mathbf{v} \neq 0$. Let us compute

$$\begin{aligned}
\dot{V}(\mathbf{x}, \mathbf{v}) &= \langle \nabla_{\mathbf{x}} V(\mathbf{x}, \mathbf{v})) \,, \dot{\mathbf{x}} \rangle + \langle \nabla_{\mathbf{v}} V(\mathbf{x}, \mathbf{v})) \,, \dot{\mathbf{v}} \rangle \\
&= -(\mathbf{g} - \mathbf{x})^\top \mathbf{K}\mathbf{v} + \mathbf{v}^\top \dot{\mathbf{v}} \\
&= -\mathbf{v}^\top \mathbf{K}(\mathbf{g} - \mathbf{x}) + \mathbf{v}^\top \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{v}^\top \mathbf{D}\mathbf{v} + \gamma \mathbf{v}^\top \mathbf{R}\mathbf{v}\vartheta \exp(-\beta\vartheta) \\
&= -\mathbf{v}^\top \mathbf{D}\mathbf{v} < 0.
\end{aligned}$$

The damping matrix $\mathbf{D}$ is positive definite by construction. The term $\mathbf{v}^\top \mathbf{R}\mathbf{v}$ is zero since the matrix $\mathbf{R}$ is a rotation by 90 degrees. If $\mathbf{v} = \mathbf{0}$ and $\mathbf{x} \neq \mathbf{g}$, then $\dot{V} = 0$. However, if $\mathbf{x} \neq \mathbf{g}$ then $\mathbf{v} \neq \mathbf{0}$ and V changes. Thus, according to LaSalle's theorem, $\mathbf{x}$ converges to $\mathbf{g}$. □

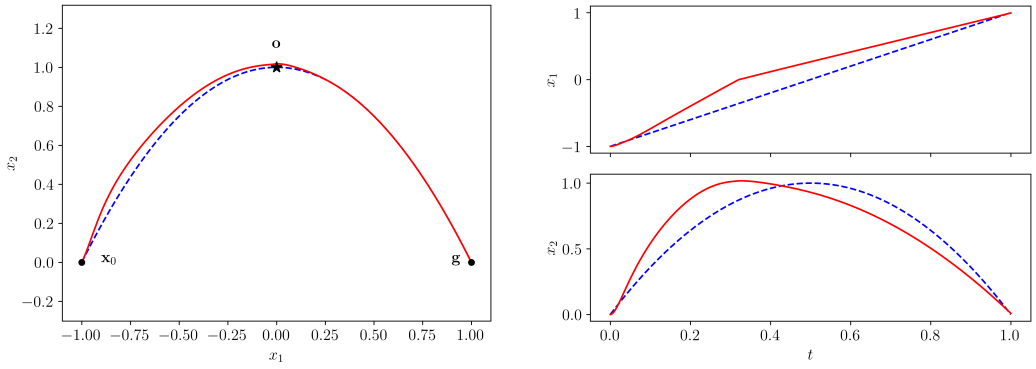
On the other hand, using potential functions (3.2) and (3.7), there may be cases in which the system remains 'trapped' in a local minimum. However, as a shortcoming, the definition of the matrix $\mathbf{R}$ makes sense only in $\mathbb{R}^3$ (and $\mathbb{R}^2$ by ignoring one of the three components). Thus, this approach is applicable only when DMPs are used in ambient space, and not joint space. Moreover, formulation (3.10) does not depend on the distance from the obstacle, and the same 'importance' is given to close and far obstacles. This may result in oscillatory behaviors, as pointed out in [43], and as we will show later in Section 3.3 with some examples.

Example 3.3. To show the behavior of this method for obstacle avoidance, we learn a DMP on the desired trajectory

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t \\ 1 - t^2 \end{bmatrix}, \qquad t \in [-1, 1].$$

Then, we place a point obstacle in $\mathbf{o} = [0, 1]^\top$. The parameters of the perturbation term (3.10) are $\gamma = 50$ and $\beta = 1$. Figure 3.6 shows the adaptation of the DMP to the presence of the obstacle.

From this example, it is possible to observe that, being a 'dynamic' (i.e. velocity-dependent) perturbation term, the obstacle-avoidance behavior is smooth. However, since (3.10) does not depend on the distance from the obstacle, the system deviates from the learned behavior from the very beginning. We will discuss this aspect in more detail in Section 3.3.



(a) Trajectory. (b) Solution.

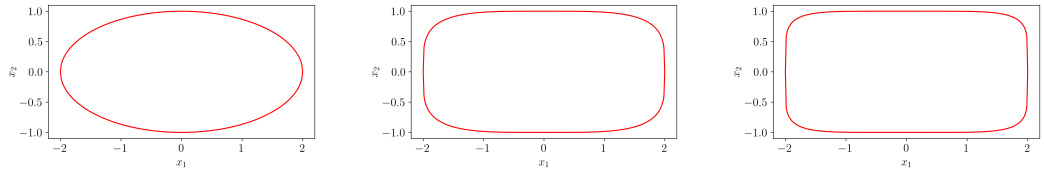
Figure 3.6: Obstacle avoidance behavior using the steering angle method for point obstacles (3.10). In both plots, the blue dashed line shows the desired (learned) curve, while the solid red line shows the adaptation of the DMP to the presence of the obstacle (marked with a black star).

3.2. Volumetric Obstacles

The methods presented in Section 3.1 work only for point obstacles. Volumetric obstacles can be modeled using point clouds or by choosing a 'critical point' (e.g. the closer one) on the surface of the obstacle itself. However, both these strategies may generate odd and undesired behaviors: using a point cloud may result in high computational time, and it is in general hard to decide a priori how dense the point cloud should be, and using a critical point can result in non-smooth behaviors since this point is constantly changing. In [110] was proposed a method that uses both the obstacle center and the point on the obstacle surface that is closer to the system position $\mathbf{x}$. However, due to the high number of hyper-parameters, it was not possible to tune by hand the hyper-parameters' value, and multiple demonstrations with different types and sizes of the obstacles were required to obtain a functioning formulation.

In this Section, we present two methods (a 'static' and a 'dynamic' potential) to deal with volumetric obstacles without the need to limit ourselves to critical points, but by instead considering the whole obstacle surface. To do so, we start by defining the concept of *isopotential* for an obstacle.

Definition 3.1 (Isopotential)**.** Let $\mathcal{A} \subset \mathbb{R}^d$ be a non-empty closed, simply connected subset of $\mathbb{R}^n$. An *isopotential* is a function $C : \mathbb{R}^d \to \mathbb{R}$ that satisfies the following properties:



(a) $n_1 = n_2 = 1$. (b) $n_1 = n_2 = 2$. (c) $n_1 = n_2 = 3$.

Figure 3.7: Example on how increasing the exponent value in the pseudo-ellipsoid definition results in a flattening of the edges.

I1 The boundary of the set $\mathcal{A}$, $\partial\mathcal{A}$, is the zero-level set of the isopotential:

$$\mathbf{x} \in \partial\mathcal{A} \Rightarrow C(\mathbf{x}) = 0;$$

I2 The value of C increases when the distance from the set $\mathcal{A}$ increases:

$$\text{dist}(\mathcal{A}, \mathbf{y}) > \text{dist}(\mathcal{A}, \mathbf{x}) \Rightarrow C(\mathbf{y}) > C(\mathbf{x}), \qquad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d.$$

An example of isopotential in $\mathbb{R}^3$ is the following [133]:

$$C(\mathbf{x}) = \left(\left(\frac{x_1 - \hat{x}_1}{f_1(\mathbf{x})} \right)^{2n} + \left(\frac{x_2 - \hat{x}_2}{f_2(\mathbf{x})} \right)^{2n} \right)^{\frac{2m}{2n}} + \left(\frac{x_3 - \hat{x}_3}{f_3(\mathbf{x})} \right)^{2m} - 1, \tag{3.12}$$

which vanishes on a *generalized ellipsoid* centered in $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \hat{x}_3]^\intercal$. By tuning parameters m, n and functions f_1, f_2, f_3 it is possible to model obstacles of any shape (their boundary will be the zero-level set of (3.12)).

An example of isopotential can be obtained by setting $m = n = 1$, and $f_i = \texttt{const}$, obtaining the isopotential

$$C(\mathbf{x}) = \left(\frac{x_1 - \hat{x}_1}{\ell_1} \right)^2 + \left(\frac{x_2 - \hat{x}_2}{\ell_2} \right)^2 + \left(\frac{x_3 - \hat{x}_3}{\ell_3} \right)^2 - 1,$$

that vanishes on an ellipsoid.
This isopotential can be generalized by using different exponents, obtaining the isopotential

$$C(\mathbf{x}) = \left(\frac{x_1 - \hat{x}_1}{\ell_1} \right)^{2n_1} + \left(\frac{x_2 - \hat{x}_2}{\ell_2} \right)^{2n_2} + \left(\frac{x_3 - \hat{x}_3}{\ell_3} \right)^{2n_3} - 1, \tag{3.13}$$

that vanishes on a *pseudo-ellipsoid*. This formulation can be easily extended to $d-$dimensional space $\mathbb{R}^d$ as

$$C(\mathbf{x}) = \sum_{j=1}^{d} \left(\frac{x_j - \hat{x}_j}{\ell_j} \right)^{2n_j} - 1.$$

The main advantage of using pseudo-ellipsoids (3.13) is that they are both easy and computationally fast to compute. Moreover, by increasing the exponent value n_i it is possible to 'flatten' the edges, making the shape resemble a rectangle (in 2D) or a parallelepiped (in 3D). Figure 3.7 shows an example in 2D.
We now show how to compute the gradient of (3.13). The partial derivative over direction i is

$$\begin{aligned}\frac{\partial}{\partial x_i}C(\mathbf{x}) &= \frac{\partial}{\partial x_i}\left(\frac{x_i-\hat{x}_i}{\ell_i}\right)^{2n_i}\\ &= \frac{1}{\ell_i^{2n_i}}2n_i(x_i-\hat{x}_i)^{2n_i-1}\\ &= 2\frac{n_i}{\ell_i}\left(\frac{x_i-\hat{x}_i}{\ell_i}\right)^{2n_i-1}.\end{aligned}$$

For instance, let us consider the case in which the isopotential $C(\mathbf{x})$ is an ellipsoid in $\mathbb{R}^3$ with center $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \hat{x}_3]^\intercal$, and axes (ℓ_1, ℓ_2, ℓ_3),

$$C(\mathbf{x}) = \left(\frac{x_1-\hat{x}_1}{\ell_1}\right)^2 + \left(\frac{x_2-\hat{x}_2}{\ell_2}\right)^2 + \left(\frac{x_3-\hat{x}_3}{\ell_3}\right)^2.$$

In this case, the gradient is

$$\nabla_{\mathbf{x}}C(\mathbf{x}) = 2\begin{bmatrix}\frac{x_1-\hat{x}_1}{\ell_1^2} & \frac{x_2-\hat{x}_2}{\ell_2^2} & \frac{x_3-\hat{x}_3}{\ell_3^2}\end{bmatrix}^\intercal.$$

3.2.1. Static-Potential Volume Obstacle

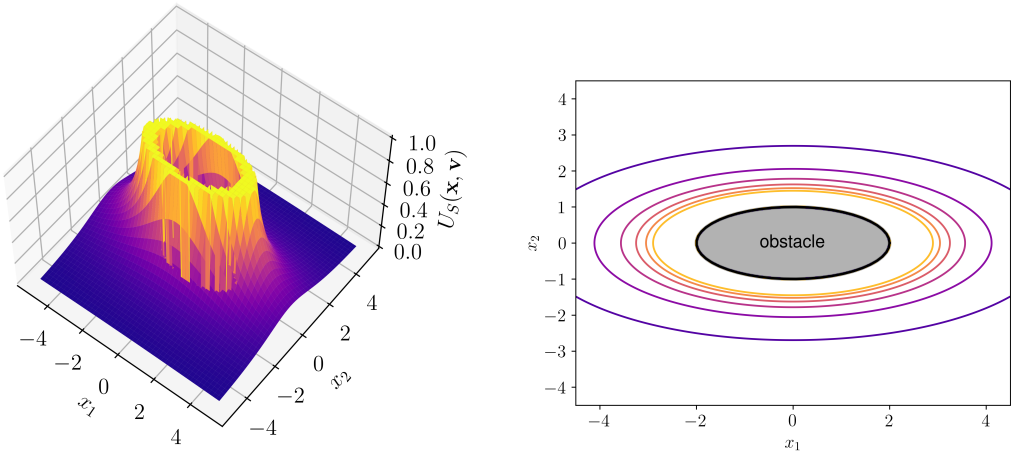
In [43] we proposed a method to implement volumetric obstacle avoidance, based on the theory of *superquadric potential functions* [133]. In this approach, the following static potential function is defined

$$U_S(\mathbf{x}) = \frac{A\exp(-\eta C(\mathbf{x}))}{C(\mathbf{x})}, \tag{3.14}$$

where $A, \eta \in \mathbb{R}_+$ are gain parameters, and $C(\mathbf{x})$ is an isopotential satisfying Properties I1 and I2. Figure 3.8 shows an example of potential $U_S(\mathbf{x})$.

The perturbation term is defined, as in (3.3), as the negative gradient, w.r.t. the system's position $\mathbf{x}$, of the potential and can be computed as follows:

$$\begin{aligned}\mathbf{p}(\mathbf{x},\mathbf{v}) = \mathbf{p}_S(\mathbf{x}) &= -\nabla_{\mathbf{x}}U_S(\mathbf{x})\\ &= -\nabla_{\mathbf{x}}\left(\frac{A\exp(-\eta C(\mathbf{x}))}{C(\mathbf{x})}\right)\\ &= -\frac{A}{C^2(\mathbf{x})}\Big(C(\mathbf{x})\nabla_{\mathbf{x}}\Big(\exp(-\eta C(\mathbf{x}))\Big) - \exp(-\eta C(\mathbf{x}))\nabla_{\mathbf{x}}C(\mathbf{x})\Big)\end{aligned}$$



(a) Mesh plot.

(b) Isocontour plot.

Figure 3.8: Example of the static potential $U_S(\mathbf{x})$ given in (3.14) for an ellipse in $\mathbb{R}^2$. The gains are set to $A = 1$, and $\eta = 0.01$. The ellipse has center in $[0,0]^\intercal$, horizontal semi-axis 2, and vertical semi-axis 1.

$$= -\frac{A}{C^2(\mathbf{x})}\Big(-\eta C(\mathbf{x})\exp(-\eta C(\mathbf{x}))\nabla_{\mathbf{x}}C(\mathbf{x}) - \exp(-\eta C(\mathbf{x}))\nabla_{\mathbf{x}}C(\mathbf{x})\Big)$$
$$= \frac{A}{C^2(\mathbf{x})}\exp(-\eta C(\mathbf{x}))(1+\eta C(\mathbf{x}))\nabla_{\mathbf{x}}C(\mathbf{x}).$$

Example 3.4. To show the behavior of this method for obstacle avoidance, we learn a DMP on the desired trajectory

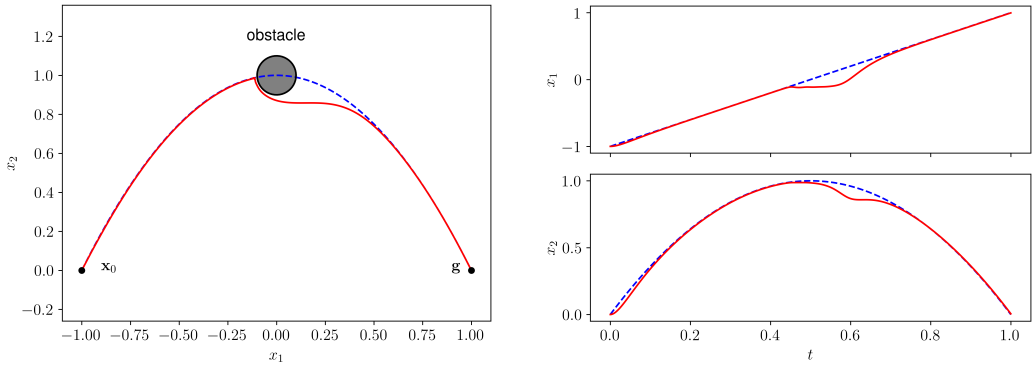
$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t \\ 1 - t^2 \end{bmatrix}, \qquad t \in [-1, 1].$$

Then, we place a circle-shaped obstacle with center in $[0,1]^\intercal$ and radius 0.1. This means that isopotential $C(\mathbf{x})$ reads

$$C(\mathbf{x}) = \left(\frac{x_1}{0.1}\right)^2 + \left(\frac{x_2 - 1}{0.1}\right)^2 - 1.$$

The parameters of the potential function $U_S(\mathbf{x})$ in (3.14) are $A = 1$ and $\eta = 1$. Figure 3.9 shows the adaptation of the DMP to the presence of the obstacle.

As can be seen in Figure 3.9, this method allows to successfully avoid the obstacle. However, similar to the static potential for point obstacles, the resulting trajectory is non-smooth since the perturbation term $\mathbf{p}$ doen not take into account the velocity of the system: $\mathbf{p}(\mathbf{x}, \mathbf{v}) \equiv \mathbf{p}(\mathbf{x})$. In the next Section, we propose a velocity-dependent potential field for volumetric obstacles which allows to smoothly avoid obstacles with volume.



(a) Trajectory. (b) Solution.

Figure 3.9: Obstacle avoidance behavior using the static potential for volume obstacles (3.14). In both plots, the blue dashed line shows the desired (learned) curve, while the solid red line shows the adaptation of the DMP to the presence of the obstacle (drawn in black).

3.2.2. Dynamic-Potential Volume Obstacle

Similar to what we discussed at the beginning of Section 3.1.2 for point obstacles, a static potential may result in non-smooth obstacle avoidance behavior. This is true also in the case of volumes obstacle, as it can be seen in Figure 3.9. The trajectory shows a hard right-turn in the trajectory in Figure 3.9a. Moreover, the plot of the solution in Figure 3.2b (top) shows that the change in velocity is not continuous in the first component near time $t = 0.5$.

Similarly to Section 3.1.2, we aim at designing a potential that satisfies the following three properties [103]:

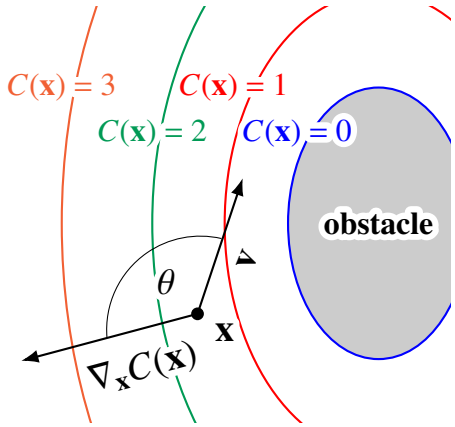
P1 The magnitude of the potential decreases with the distance of the system $\mathbf{x}$ from the obstacle;

P2 The magnitude of the potential increases with the velocity of the system $\|\mathbf{v}\|$ and is zero when the system is not moving;

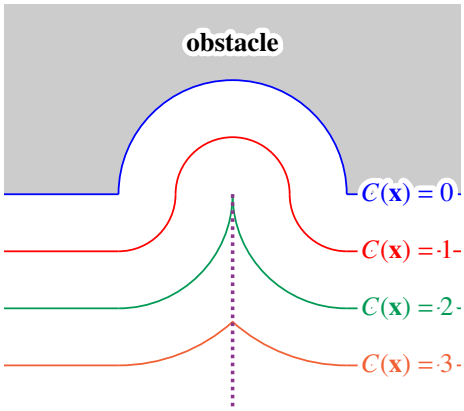
P3 The magnitude of the potential decreases with the angle between current velocity direction $\mathbf{v}/\|\mathbf{v}\|$, and the direction towards the obstacle; and, if the system is moving away from the obstacle, the potential should vanish.

To this end, we define, similarly to (3.7), the potential function [46]

$$U_D(\mathbf{x},\mathbf{v}) = \begin{cases} \lambda(-\cos\theta)^\beta \dfrac{\|\mathbf{v}\|}{C^\eta(\mathbf{x})} & \text{if } \theta \in \left(\frac{\pi}{2},\pi\right] \\ 0 & \text{if } \theta \in \left[0,\frac{\pi}{2}\right] \end{cases}, \tag{3.15}$$



(a) Example of convex obstacle in which θ is well defined.



(b) Example of non-convex obstacle in which θ is not well defined: the purple dotted line shows the points in which $\nabla_{\mathbf{x}} C(\mathbf{x})$ is not well defined.

Figure 3.10: Figure 3.10a shows how the angle θ is defined when the gradient $\nabla_{\mathbf{x}} C(\mathbf{x})$ of the isopotential exists. Figure 3.10b, instead, shows an example on how non-convex obstacles result in non differentiable isopotentials, and thus it is not possible to define the angle θ.

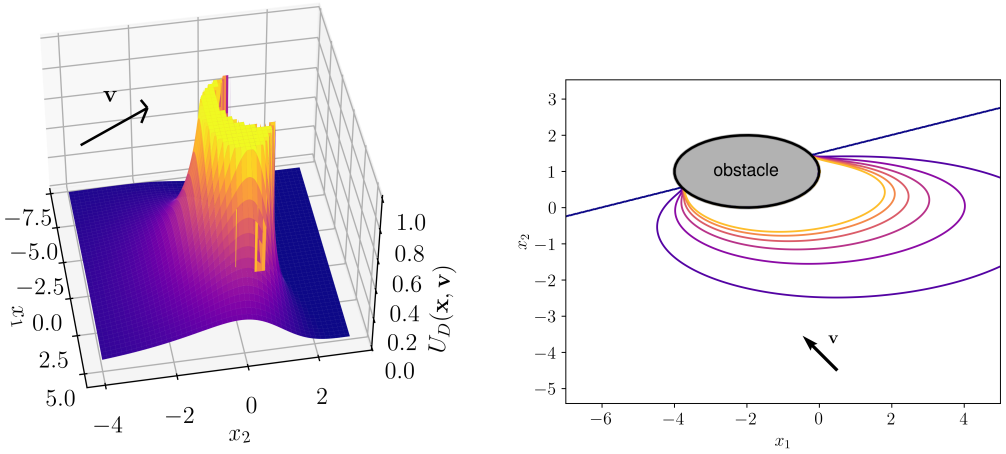
where λ,β, and $\eta \in \mathbb{R}_+$ are constant gains, and function $C(\mathbf{x})$ is any ispotential satisfying Properties I1 and I2 given in Definition 3.1. The angle θ is taken between the system's velocity $\mathbf{v}$ and the direction between the system's position $\mathbf{x}$ and the closest point of the obstacle. Thanks to Property I2, we have that the gradient of the isopotential $C(\mathbf{x})$, $\nabla_{\mathbf{x}} C(\mathbf{x})$, is always perpendicular to the obstacle surface. Thus, the angle θ can be computed, for convex obstacles, using

$$\cos\theta = \frac{\langle \nabla_{\mathbf{x}} C(\mathbf{x}), \mathbf{v} \rangle}{\|\nabla_{\mathbf{x}} C(\mathbf{x})\| \|\mathbf{v}\|},$$

while it is not in general well defined for non-convex obstacles. An intuition for these two observations are given in Figure 3.10.

Remark 3.1. For non-convex obstacles, some workarounds can be employed. First, if neither the starting position nor the goal is in the 'holes' of the obstacle, that is they are not in the convex hull of the obstacle, then the convex hull itself can be used as an obstacle. Second, one can think of relaxing the concept of the gradient to allow sub-differentials. In such a case, the sub-gradient exists but it is not unique. Thirdly, a non-convex obstacle can be split into multiple convex components, and each component would generate its own potential.

The potential defined in (3.15) clearly satisfies Properties P1–P3. Indeed, the potential is a decreasing function of $C(\mathbf{x})$ and an increasing function of θ, thus it



(a) Mesh plot.

(b) Isocontour plot.

Figure 3.11: Example of the dynamic potential $U_D(\mathbf{x}, \mathbf{v})$ given in (3.15) for an ellipse in $\mathbb{R}^2$. The velocity vector $\mathbf{v}$ is set to $\mathbf{v} = [-1, 1]^\intercal$. The gains are set to $\lambda = 2, \beta = 2$, and $\eta = 1$. The ellipse has center in $[-2, 1]^\intercal$, horizontal semi-axis 2, and vertical semi-axis 1. In both figures, the potential has been cropped at the value of 1 for display purposes: it goes to infinity on half of the boundary of the obstacle (on the other half, the system goes away from the obstacle, so the potential is zero).

satisfies Properties P1 and P3. Moreover, it is an increasing function of $\|\mathbf{v}\|$, and thus it satisfies also Property P2.
As an example, we show in Figure 3.11 the potential (3.15) for an elliptic obstacle in $\mathbb{R}^2$, whose isopotential is

$$C(\mathbf{x}) = \left(\frac{x_1 - \hat{x}_1}{\ell_1}\right)^2 + \left(\frac{x_2 - \hat{x}_2}{\ell_2}\right)^2 - 1,$$

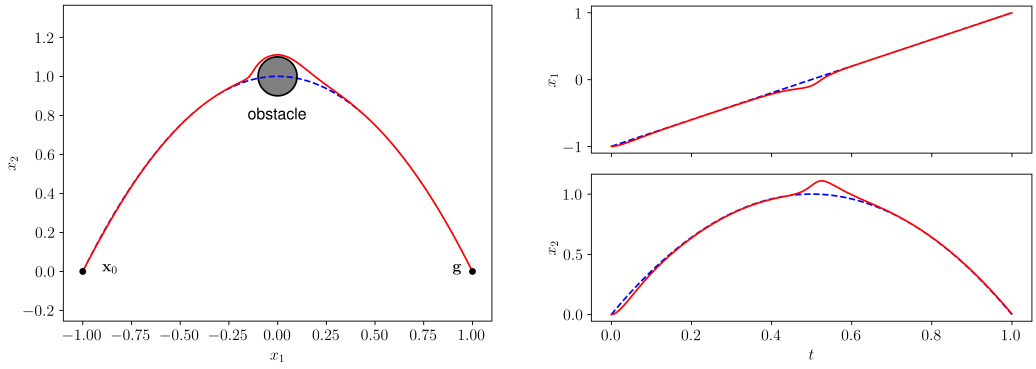
where the center of the ellipse is $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2]^\intercal$ and the horizontal and vertical axes are, respectively, ℓ_1 and ℓ_2.

The perturbation term is defined again as the negative gradient, w.r.t. the system's position $\mathbf{x}$, of the potential, $\mathbf{p}(\mathbf{x}, \mathbf{v}) = \mathbf{p}_D(\mathbf{x}, \mathbf{v}) = -\nabla_{\mathbf{x}} U_D(\mathbf{x}, \mathbf{v})$. We can compute it as follows:

$$\begin{aligned}
\mathbf{p}_D(\mathbf{x}, \mathbf{v}) &= -\nabla_{\mathbf{x}}(U_D(\mathbf{x}, \mathbf{v})) \\
&= -\nabla_{\mathbf{x}}\left(\lambda(-\cos\theta)^\beta \frac{\|\mathbf{v}\|}{C^\eta(\mathbf{x})}\right) \\
&= -\frac{\lambda\|\mathbf{v}\|(-\cos\theta)^{\beta-1}}{C^\eta(\mathbf{x})}\left(-\beta\nabla_{\mathbf{x}}(\cos\theta) + \frac{\eta\cos\theta}{C(\mathbf{x})}\nabla_{\mathbf{x}}(C(\mathbf{x}))\right).
\end{aligned} \tag{3.16}$$

The term $\nabla_{\mathbf{x}}(\cos\theta)$ can be computed as

$$\nabla_{\mathbf{x}}(\cos\theta) = \nabla_{\mathbf{x}}\left(\frac{\langle \nabla_{\mathbf{x}} C(\mathbf{x}), \mathbf{v}\rangle}{\|\nabla_{\mathbf{x}} C(\mathbf{x})\|\|\mathbf{v}\|}\right)$$



(a) Trajectory. (b) Solution.

Figure 3.12: Obstacle avoidance behavior using the dynamic potential for volume obstacles (3.15). In both plots, the blue dashed line shows the desired (learned) curve, while the solid red line shows the adaptation of the DMP to the presence of the obstacle (drawn in black).

$$= \frac{1}{\|\mathbf{v}\| \, \|\nabla_{\mathbf{x}} C(\mathbf{x})\|^2} \Big(\|\nabla_{\mathbf{x}} C(\mathbf{x})\| \, \nabla_{\mathbf{x}} (\langle \nabla_{\mathbf{x}} C(\mathbf{x}), \mathbf{v} \rangle) - \langle \nabla_{\mathbf{x}} C(\mathbf{x}), \mathbf{v} \rangle \, \nabla_{\mathbf{x}} (\|\nabla_{\mathbf{x}} C(\mathbf{x})\|) \Big). \tag{3.17}$$

For an isopotential as (3.13) with $n_i = 2$, quantities $\nabla_{\mathbf{x}}(\langle \nabla_{\mathbf{x}} C(\mathbf{x}), \mathbf{v} \rangle)$ and $\nabla_{\mathbf{x}}(\|\nabla_{\mathbf{x}} C(\mathbf{x})\|)$ in (3.17) read, respectively,

$$\nabla_{\mathbf{x}}(\langle \nabla_{\mathbf{x}} C(\mathbf{x}), \mathbf{v} \rangle) = 2 \begin{bmatrix} \frac{v_1}{\ell_1^2} & \frac{v_2}{\ell_2^2} & \frac{v_3}{\ell_3^2} \end{bmatrix}^\intercal,$$

and

$$\nabla_{\mathbf{x}}(\|\nabla_{\mathbf{x}} C(\mathbf{x})\|) = \frac{4}{\|\nabla_{\mathbf{x}} C(\mathbf{x})\|} \begin{bmatrix} \frac{x_1 - \hat{x}_1}{\ell_1^4} & \frac{x_2 - \hat{x}_2}{\ell_2^4} & \frac{x_3 - \hat{x}_3}{\ell_3^4} \end{bmatrix}^\intercal.$$

Example 3.5. To show the behavior of this method for obstacle avoidance, we learn a DMP on the desired trajectory

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t \\ 1 - t^2 \end{bmatrix}, \qquad t \in [-1, 1].$$

Then, we place a circle-shaped obstacle with center in $[0, 1]^\intercal$ and radius 0.1. The isopotential $C(\mathbf{x})$ is the same as for Example 3.4 The parameters of the potential function $U_D(\mathbf{x}, \mathbf{v})$ in (3.15) are $\lambda = 10$, $\beta = 2$, and $\eta = 1$. Figure 3.12 shows the adaptation of the DMP to the presence of the obstacle.

As it can be seen, the obstacle-avoidance behavior obtained with the dynamic potential is smoother than that obtained with a static potential. In particular, with the dynamic potential, the obstacle is avoided from the top (see Figure 3.12a). This

Method	Type of obstacle	Space of definition	Type of potential	Distance dependent	Guaranteed convergence
Static potential (3.2)	Point	$\mathbb{R}^d, d \in \mathbb{N}$	Static	Yes	No
Dynamic potential (3.7)	Point	$\mathbb{R}^d, d \in \mathbb{N}$	Dynamic	Yes	No
Steering angle (3.10)	Point	$\mathbb{R}^2, \mathbb{R}^3$	Dynamic	No	Yes*
Static potential (3.14)	Volume	$\mathbb{R}^d, d \in \mathbb{N}$	Static	Yes	No
Dynamic potential (3.15)	Volume	$\mathbb{R}^d, d \in \mathbb{N}$	Dynamic	Yes	No

Table 3.1: Summary of the properties of various methods for obstacle avoidance. The desired properties are underlined. The asterisk on the guaranteed convergence for the steering angle method remarks the fact that this property is true only if the obstacle is not moving $\dot{\mathbf{o}} = \mathbf{0}$.

results in a small modification of the direction of the system's velocity $\mathbf{v}$ (the DMPs system was already moving upwards and to the right). On the other hand, with the static potential, the system avoids the obstacle from the bottom (see Figure 3.9a), and this results in a more pronounced change in the direction of the system's velocity.

3.3. General Considerations and Synthetic Tests

In this Section, we present a comparison between the obstacle avoidance methods we discussed in this Chapter.

At first, let us discuss the properties (summarized in Table 3.1) of each method. We can consider five major properties to classify each method: the *type of obstacle* (point or volume), the *space of definition*, the *type of potential* (static or dynamic), if the method is *distance dependent*, and if the obstacle avoidance strategy ensure *convergence to the goal*.

Clearly, a method able to model volumetric obstacles is preferable. Indeed, to model obstacles in real scenarios (that, obviously, are not point) using point obstacles methods one should generate a point cloud of the obstacle surface and use each of the points as a point-obstacle. Otherwise, one should choose a set of critical points on the obstacle. However, the first method would result in a huge computational cost. The second method, instead, may result in oscillatory behaviors since the 'critical point' keep changing position. Later in this Section, we will present a comparison between using a volume and a mesh of points.

Moreover, since one may prefer working in joint-space instead of ambient-space, methods able to model obstacles in general d - dimensional spaces $\mathbb{R}^d$ are preferable.

Except for the steering angle method (3.10) (which works only in $\mathbb{R}^2$ and $\mathbb{R}^3$), all other methods we presented (both for point and volume obstacles) can be used in general $\mathbb{R}^d$.
An obstacle avoidance method should depend on both its position and velocity (relative to the obstacle). Indeed, being dependent on the distance from the obstacle allows one to avoid early, or completely unnecessary, perturbations of the trajectory if the obstacle is far from the system's position. Similarly, being dependent on the velocity results usually in smoother behaviors as we already saw in previous examples and as we will analyze further in this Section. Moreover, a dynamic potential can avoid perturbations of the trajectory if the system is going away from the obstacle. Dynamic potentials (3.7), (3.15) depend on both position and velocity of the system. Static potentials (3.2) and (3.14) depends only on the position of the system. The steering angle method (3.10) depends on both the position and the velocity of the system. However, the dependence on the position does not take into account the distance from the obstacle, but only the direction between the obstacle and the system's position.
Finally, it would be preferable for an obstacle avoidance method to guarantee convergence of the DMP to the goal position. This property is guaranteed, by Theorem 3.1, only by the steering angle method (3.10).

Remark 3.2. Dynamic potentials (3.7), (3.15) and steering angle method (3.10) do not take into account the velocity of the obstacle. However, it is straightforward to extend the definition to this case by simply substituting $\mathbf{v}$ with $\mathbf{v} - \dot{\mathbf{o}}$, where $\dot{\mathbf{o}}$ denotes the velocity of the obstacle.

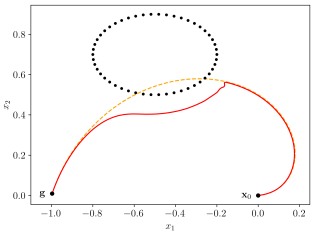
We remark that proof of Theorem 3.1 holds true only if $\dot{\mathbf{o}} = \mathbf{0}$ [54]. Thus, if the obstacle is moving, none of the obstacle avoidance methods for DMPs can guarantee convergence of the system to the goal.

We now present a comparison between the methods presented in this Chapter. In particular, for all the proposed methods, we present a 2 - dimensional test both with one and two obstacles. For point obstacle methods, the outline of the obstacle will be covered by a uniform mesh of point obstacles.

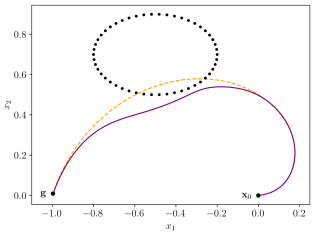
Example 3.6. In the first example, we learn the desired behavior

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} t\cos(\pi t) \\ t\sin(\pi t) \end{bmatrix}, \qquad t \in [0,1].$$

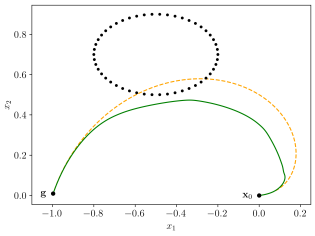
the obstacle is an ellipse centered in $[-0.5, 0.7]^\intercal$ with horizontal and vertical semi-axis 0.3 and 0.2 respectively. For the tests done with point obstacle methods, the outline of the ellipse is covered with fifty uniformly distributed points. The parameter of the obstacles are:



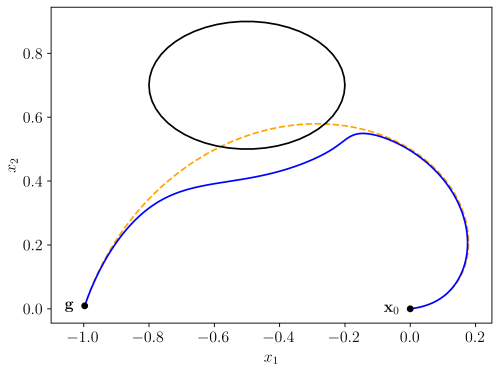
(a) Static potential (3.2).



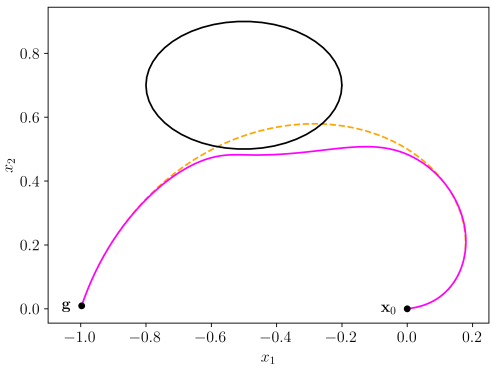
(b) Dynamic potential (3.7).



(c) Steering angle (3.10).



(d) Static potential (3.14).

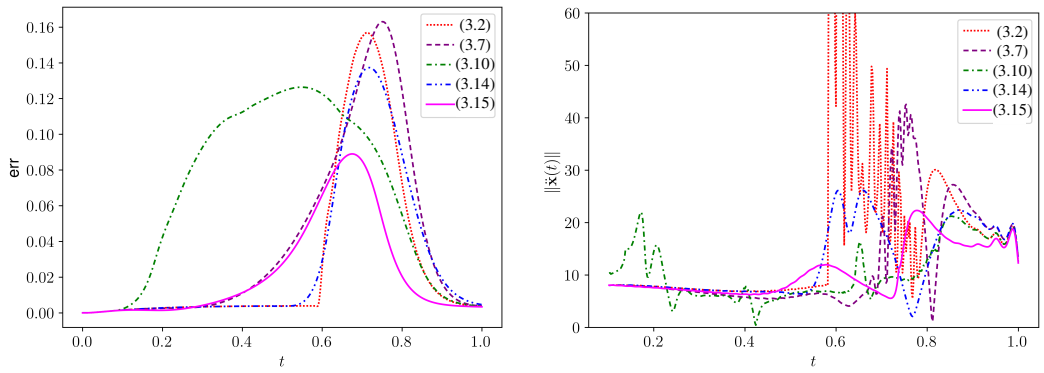


(e) Dynamic potential (3.15).

Figure 3.13: Obstacle avoidance behaviors from Example 3.6. In all plots, the dashed orange line shows the desired trajectory, while the solid colored line shows the adaptation of the DMP to the presence of the obstacle. In the three top figures, the black dots mark the point obstacles used as mesh. In the two bottom figures, the boundary of the obstacle is plotted using the solid black line.

- for the static point obstacle method (3.2), $\rho_0 = 0.1$ and $\eta = 1$;
- for the dynamic point obstacle method (3.7), $\lambda = 0.2$ and $\beta = 2$;
- for the steering angle method (3.10), $\gamma = 20$ and $\beta = 3.0$;
- for the static volume obstacle method (3.14), $A = 10$ and $\eta = 1$;
- for the dynamic volume obstacle method (3.15), $\lambda = 10$, $\beta = 2$, and $\eta = 0.5$.

In Figure 3.13 the behaviors of all the obstacle avoidance methods are presented. Moreover, in Figure 3.14, we plot the distance between learned and adapted behavior (on the left), and the norm of the acceleration of the adapted behavior (on the right) as a function of time. As it can be seen, the steering angle method results in an earlier deviation from the learned behavior, since the perturbation term (3.10) does not depend on the distance between the system position and the obstacle. Moreover, we remark that static methods result in more oscillatory behaviors, while dynamic methods give a smoother acceleration profile.



(a) Plot of the distance (in 2-norm) between the desired trajectory and the executed one.

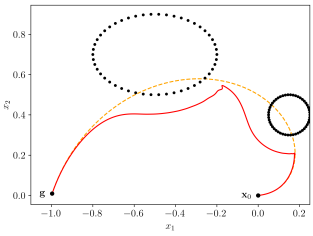
(b) Plot of the norm of the acceleration of the executed DMP.

Figure 3.14: For tests described in Example 3.6 (and depicted in Figure 3.13), plot of the distance between desired and executed trajectory (left), and of the norm of the acceleration (right) as functions of time.

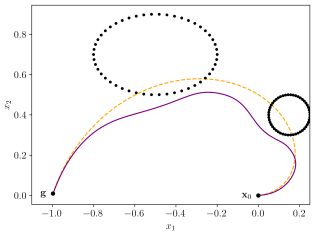
Table 3.2 shows the maximum and average values of both the error and the acceleration norms. From it, we see that the steering angle method (3.10) result in smaller accelerations. However, the acceleration profile still results in more oscillatory behavior than the volumetric dynamic potential (3.15), as it can be seen from Figure 3.14b.

In summary, dynamic potential (3.15) gives both the less oscillatory behavior and the trajectory that remains closer to the learned one between all the methods we presented in this Chapter.

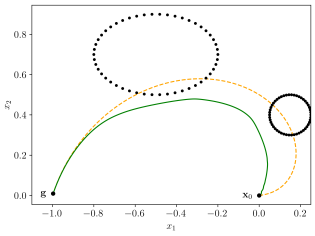
Example 3.7. In the second example, we maintain the same desired trajectory and obstacle as in Example 3.6; and we add a second circular obstacle. This second obstacle has center in $[0.15, 0.4]^\intercal$ and radius 0.1. For the tests done with point obstacle methods, the outline of the circle is covered with fifty uniformly distributed points. The parameter of the obstacles are the same as in Example 3.6. In Figure 3.15 the behaviors of all the obstacle avoidance methods are presented.
Moreover, in Figure 3.16, we plot the distance between learned and adapted behavior (on the left), and the norm of the acceleration of the adapted behavior (on the right) as a function of time. Again, the steering angle method results in an earlier deviation from the learned behavior, since the perturbation term (3.10) does not depend on the distance between the system position and the obstacles. In this case, the effect is more pronounced since the contributions of both obstacles have the same 'importance' even tho one is closer than the other. Again, static methods result in more oscillatory behaviors, while dynamic methods give a smoother acceleration profile.



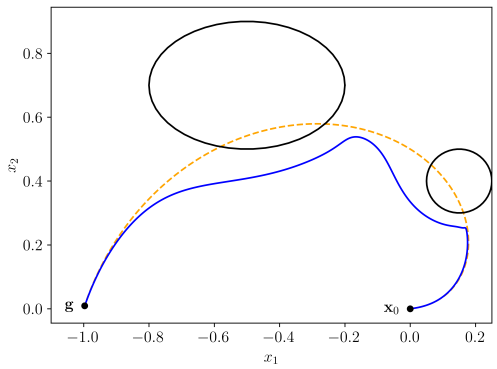
(a) Static potential (3.2).



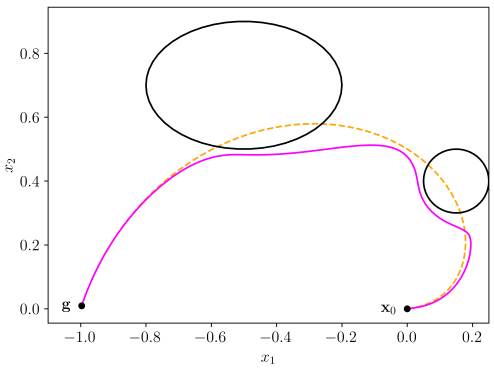
(b) Dynamic potential (3.7).



(c) Steering angle (3.10).



(d) Static potential (3.14).



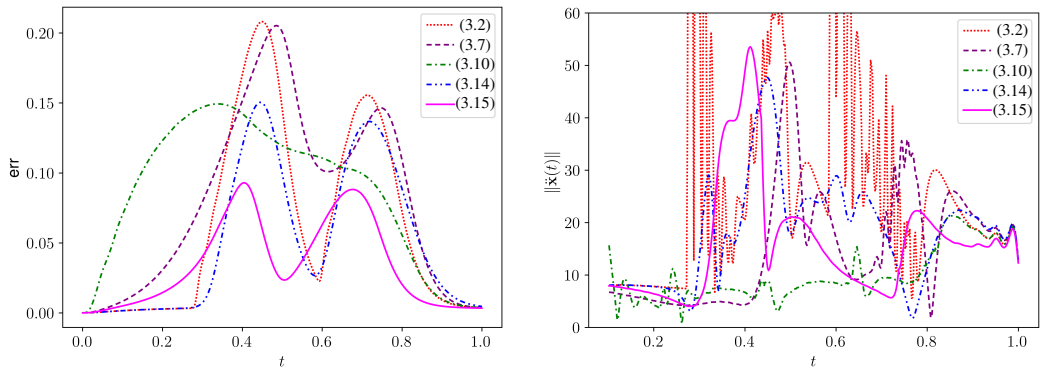
(e) Dynamic potential (3.15).

Figure 3.15: Obstacle avoidance behaviors from Example 3.7. In all plots, the dashed orange line shows the desired trajectory, while the solid colored line shows the adaptation of the DMP to the presence of the obstacle. In the three top figures, the black dots mark the point obstacles used as mesh. In the two bottom figures, the boundary of the obstacle is plotted using the solid black line.

Table 3.2 shows the maximum and average values of both the error and the acceleration norms. As for the one obstacle test, the steering angle method (3.10) result in smaller accelerations, even tho the acceleration profile results more oscillatory than the volumetric dynamic potential (3.15).
Also in this test, it is possible to observe that the volumetric dynamic method (3.15) still gives the trajectory that less deviates from the learned one while maintaining the less oscillatory behavior at the acceleration level.

Finally, we present a synthetic test with a non-convex obstacle, testing two workarounds given in Remark 3.1.

Example 3.8. We define a 'U'-shaped non-convex obstacle and present two scenarios: in the first, the goal is inside the convex hull of the 'U' and the obstacle is modeled by dividing it into three convex components; in the second, the goal is outside the 'hole', and the obstacle is modeled as the convex hull of the 'U'.
In the first scenario (Figure 3.17a), the goal is inside the 'hole' of the 'U'. Thus, we subdivide the obstacle into three components (two vertical and one horizontal);



(a) Plot of the distance (in 2-norm) between the desired trajectory and the executed one.

(b) Plot of the norm of the acceleration of the executed DMP.

Figure 3.16: For tests described in Example 3.7 (and depicted in Figure 3.15), plot of the distance between desired and executed trajectory (left), and of the norm of the acceleration (right) as functions of time.

then, we cover each component with a 2-dimensional pseudo-ellipsoid (i.e. we use formulation (3.13) with $n_1 = n_2 = 2$ and no third component). We will use the dynamic potential (3.15) with parameters $\lambda = 10, \eta = 2$, and $\beta = 2$; while the DMP parameters are $\mathbf{K} = K\mathbf{I}\mathrm{d}_2, \mathbf{D} = D\mathbf{I}\mathrm{d}_2$ with $K = 150$ and $D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$.
In the second scenario (Figure 3.17b) neither the goal nor the initial position is in the 'hole' of the obstacle. Thus, we consider as the obstacle the convex hull of the 'U', and we cover it with the 2-dimensional pseudo-ellipsoid.
It is possible to see that both approaches result in proper obstacle avoidance behavior.

We now discuss the computational time. Our DMPs' implementation and obstacle avoidance methods are implemented in Python3.6 and can be found at `https://github.com/mginesi/dmp_vol_obst`.
To test the execution time we generate one obstacle for each method for different dimensionalities d of the state space. Then, for each value of d, we generate 100 random positions and velocities to compute the perturbation terms $\mathbf{p}(\mathbf{x}, \mathbf{v})$ and use these values to compute the average computational time and its standard deviation. Table 3.3 show the results of this test. We remark that the steering angle method makes sense only in $\mathbb{R}^2$ and $\mathbb{R}^3$ due to the cross product in the definition of the matrix $\mathbf{R}$ in (3.10).
The dynamic potential (3.15) results to be the slowest. However, it is computed in about a tenth of a millisecond which still makes it able to be computed fast enough to not influence the on-line control of most robots.

		(3.2)	(3.7)	(3.10)	(3.14)	(3.15)
1 obstacle	maximum error	0.157	0.163	0.126	0.137	0.089
	average error	0.029	0.040	0.066	0.030	0.022
	maximum acceleration norm	277.99	42.55	22.02	26.15	22.32
	average acceleration norm	19.11	12.40	11.08	12.77	11.20
2 obstacles	maximum error	0.210	0.205	0.149	0.150	0.092
	average error	0.064	0.082	0.088	0.052	0.035
	maximum acceleration norm	311.32	50.60	21.29	47.67	53.53
	average acceleration norm	30.00	15.42	11.65	18.11	16.13

Table 3.2: Statistics of synthetic tests shows in Figures 3.13 and 3.15. Minimum values for each statistic are underlined.

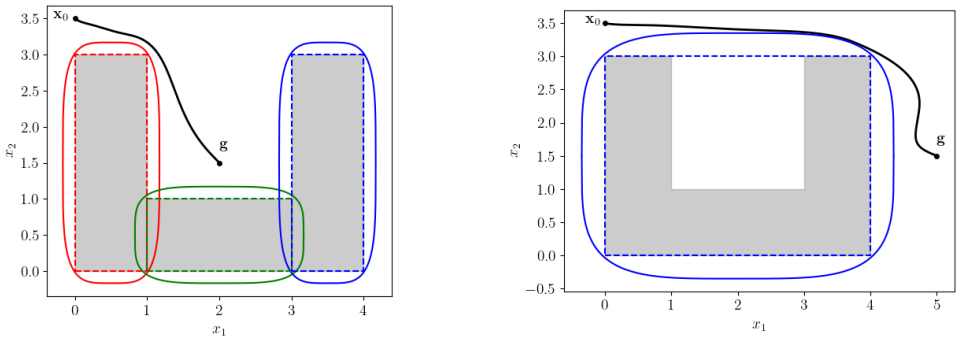
Tests were performed on a machine with a quad-core Intel Core i7-7000 CPU with 16 GB of RAM.

3.4. Experiments on Simulated and Real Setups

In this Section, we present different experiments on simulated and real robots done to test the performances of the volumetric approaches described in Section 3.2.

3.4.1. Benchmark with the Panda Robot

In order to test the static volumetric obstacle avoidance method from Section 3.2 on a real robot, the setup shown in Figure 3.18 is used. It consists of a round base (approximately of radius of 25 cm) with two pegs (approximately of radius of 1 cm and height of 12 cm) and a 7-DOF Panda industrial manipulator by Franka Emika. The robot, controlled using a ROS architecture and MoveIT interface, must move above the whole base at a fixed height while avoiding the pegs on its way. Since we are mainly interested in the obstacle avoidance problem, no manual trajectory is learned; the forcing term for DMPs in (2.10a) is set to zero, $\mathbf{f}(s) \equiv \mathbf{0}$. In this way, the dynamical system converges to the goal position with a linear trajectory. This choice does not limit the generality of the proposed approach, since the perturbation term guarantees the obstacle avoidance independently of the weights of the forcing term. The parameters of DMP system (2.10) are $\mathbf{K} = K\mathbf{I}\mathrm{d}_3$ and $\mathbf{D} = D\mathbf{I}\mathrm{d}_3$ with $K = 3050, D = 2\sqrt{K} \approx 55.23$; and $\alpha = 3$. The isopotential's (3.13) parameters are $n_1 = n_2 = 1$, and $n_3 = 2$. Finally, potential's (3.14) parameters are $A = 50$ and $\eta = 1$.



(a) Convex components.

(b) Convex hull.

Figure 3.17: Different methods to deal with non-convex obstacles. The obstacle is depicted with a gray shade. Dashed lines show the convex components in the plot on the left and the convex hull in the plot on the right. Both plots show the enlarged potential (to be written as a generalized ellipsoid) in the same color as the obstacle. The black solid line shows the executed DMP.

Isopotential semi-axes ℓ_1, ℓ_2 and ℓ_3 change between tests. Indeed, due to the big size of the hand of the manipulator (approx. 18 cm long and 5 cm wide) compared to the pegs, avoiding obstacles while passing very close to them is a challenge which can be solved in two ways, that we will present.

At first, for simplicity, the orientation is kept constant along the whole trajectory, neglecting the problem of adjusting the end effector's orientation to avoid collisions. The cylindric shape of the peg is modeled using the superquadric formulation in (3.13), but it must be modified in such a way that the base is as large as the hand of the robot to avoid collisions because the DMPs model the moving part as a point mass. To do so, we cover the obstacle with an ellipsoid superquadric which has as radii the semi-axes of the end-effector. This results in semi-axis in (3.13) to take values $\ell_1 = 2\text{cm}, \ell_2 = 9\text{cm}$, and $\ell_3 = 7\text{cm}$. As can be seen in Figure 3.20a, this solution ensures obstacle avoidance but it strongly reduces the available workspace for the robot, which is undesired when many objects are in the scene.

In the second test, we introduce the adaptation of the end-effector's orientation in order to avoid the obstacles. This allows to reduce the size of the superquadric potentials, thus increasing the available workspace, since the obstacle has to be enlarged only by the smaller semi-axis of the end-effector (for safety reasons) and the orientation will then adapt to avoid the collision. Orientation adaptation is managed via Trac-IK [7], a modern efficient inverse kinematics plugin for generic industrial and humanoid manipulators. Given the target position computed by the DMP planner, multiple corresponding joint configurations are possible, due to the redundancy

		(3.2)	(3.7)	(3.10)	(3.14)	(3.15)
$d = 2$	mean	1.31e-05	3.91e-05	6.27e-05	2.01e-05	1.14e-04
	st. dev.	6.86e-06	1.61e-05	2.57e-05	7.20e-06	3.83e-05
$d = 3$	mean	1.17e-05	2.63e-05	8.00e-05	2.11e-05	1.08e-04
	st. dev.	3.57e-06	1.03e-05	1.62e-05	5.75e-06	2.52e-05
$d = 4$	mean	1.14e-05	3.14e-05	—	2.29e-05	1.04e-04
	st. dev.	3.23e-06	1.06e-05	—	6.44e-06	1.67e-05
$d = 5$	mean	1.12e-05	2.63e-05	—	2.52e-05	1.06e-04
	st. dev.	2.69e-05	9.96e-06	—	6.44e-06	1.67e-05
$d = 6$	mean	1.20e-05	3.38e-05	—	2.73e-05	1.16e-04
	st. dev.	4.66e-06	1.20e-05	—	6.40e-06	3.31e-05
$d = 7$	mean	1.31e-05	2.43e-05	—	3.14e-05	1.20e-04
	st. dev.	4.62e-06	1.07e-05	—	8.52e-06	2.32e-05

Table 3.3: Computational time (in seconds) of the perturbation term for various methods of obstacle avoidance.

of the robot. Given the models of the obstacle and the robot as sets of voxels, Track-IK first performs a random search to only consider joint values that avoid the intersection between them. The random increment of the joint values is limited by a step parameter which depends on the size of the obstacles. Here, it is chosen as half of the peg's radius, since the peg is the smallest object in the scene. Finally, Trac-IK selects the optimal configuration which maximizes manipulability, measured as $\sqrt{\det(\mathbf{J}\mathbf{J}^\intercal)}$, being $\mathbf{J}$ the Jacobian matrix. This is an arbitrary optimization objective, others are possible (e.g., minimal joint displacement). In order to limit the search space of the algorithm, a bound of 45° on each joint is imposed, except for the hand joint, which is crucial for obstacle avoidance (90° bound).

Figure 3.20 shows the results of these experiments. Figure 3.20a-3.20b show the experiment maintaining a fixed orientation of the end-effector. In particular, Figure 3.20a shows the top view highlighting how the original trajectory would make the end-effector collide with the obstacles, while with the perturbed one the obstacles are avoided. Figure 3.20c shows the experiment with the orientation adaptation. It can be seen that the DMP path passes closer to the obstacles, but the end-effector is still able to avoid them successfully.

3.4.2. Pick & Place task with the Panda robot

In this experiment, we test the volumetric obstacle avoidance methods from Section 3.2 on the Panda industrial manipulator.

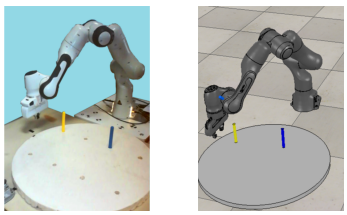
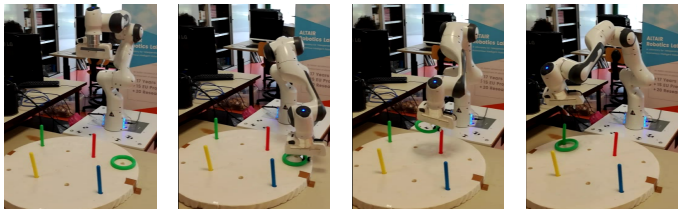


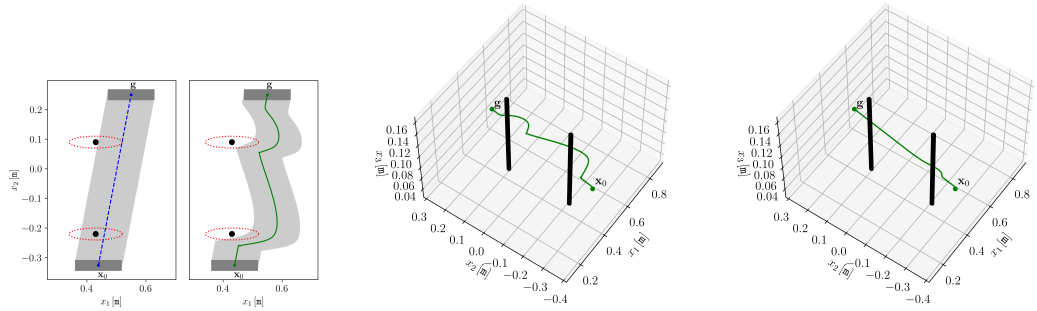
Figure 3.18: Experimental setup with the Panda robot and the peg base (left), and its V-REP simulation (right).



(a) Initial setup. (b) Grasped ring. (c) Avoided obstacle. (d) Released ring.

Figure 3.19: The pick-and-place task with the Panda robot.

The setup for the Panda robot is shown in Figure 3.19a. The robot must pick the green ring and place it on the green peg. On the way to the peg, the robot has to avoid the red peg, i.e. neither the end effector nor the grasped ring has to hit the peg. The task can be described by a simple state machine with four actions/modes: `move_to_ring`, `grasp`, `move_to_peg` and `release_gripper`. The moving actions are kinematically described with two DMPs in Cartesian space with null weights, i.e. straight-line trajectories, with $K = 1050, D = 2\sqrt{K} \approx 64.81$, and $\alpha = 4$. The trajectories describe the motion of the center of the gripper of the robot. Notice from Figure 3.19a that the encumbrance of the end effector is significant, and simply controlling the center of the gripper does not guarantee safe collision avoidance. As explained in Section 3.4.1, there are two solutions to this issue. One is to enlarge the radial dimension of the pegs according to the size of the end effector. The second solution is to exploit the kinematic redundancy of the 7-DOF manipulator and compute an obstacle-free joint configuration for each point in the DMP. We have chosen the latter approach to limit the size of the obstacles and, hence, maximize the reachable workspace for the robot. We control the robot through its standard MoveIt/ROS interface, setting TRAC-IK [7] as the inverse kinematics solver. We set the solver to compute an inverse kinematics solution which maximizes the manipulability of the robot, defined as $\sqrt{\det(\mathbf{J}\mathbf{J}^\intercal)}$ [43]. Though we do not control the orientation of the end effector with our DMP formulation, we constrain the orientation to be within 5° (along each axis) from the initial orientation for each Cartesian waypoint. Then, we gradually relax this tolerance if no inverse kinematics solution is found. We also constrain two consecutive joint configurations to differ no more than 45° on each joint, so that we are able to avoid abrupt movements during the execution. The scene (location of the peg base, the pegs, and the ring) is assumed to be known in advance. Hence, obstacles (the base and the pegs) are represented



(a) No orientation adaptation: top view.

(b) No orientation adaptation.

(c) Orientation adaptation.

Figure 3.20: Experimental results on a real robot. In all three pictures, the pegs are colored in black and the green solid line shows the adaptation of the trajectory to the presence of the obstacles. In the first plot, we show also the enlarged potential in dashed red and the free (no obstacles) trajectory in dashed blue, together with the gray shade showing the area occupied by the end-effector during the movement.

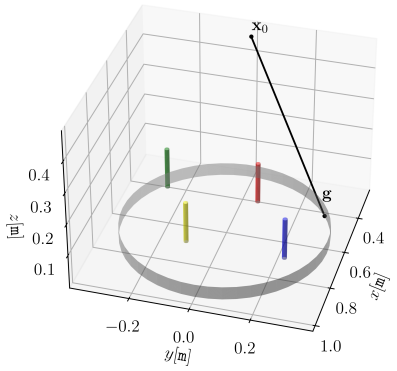
as superquadric potential function shaped as cylinders (assuming the $z-$axis as the normal to the base, exponents in (3.13) are set as $n_1 = n_2 = 1, n_3 = 2$). Figure 3.19 shows the main steps in the task execution.

After the ring is grasped (Figure 3.19b), we have that both the end-effector and the ring should avoid the pegs. Thus, when the ring is held by the robot, we 'enlarge' each obstacle by the radius of the ring. Indeed, during the `move_to_ring` mode, we have that the robot is not holding the ring, and the pegs are modeled with their actual radius (as it can be seen in Figures 3.21a and 3.21b). On the other hand, during the `move_to_peg` mode, the ring is held by the robot, and the obstacles' radius is enlarged by the radius of the ring (as it can be seen in Figure 3.21c and 3.21d) so to guarantee that neither the end-effector nor the grasped ring collides with the peg.

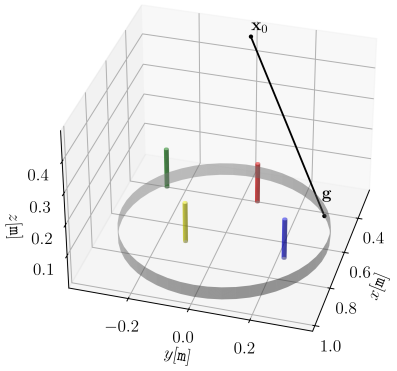
Obstacles are modelled with the static potential formulation in (3.14), setting $A = 10, \eta = 1$; and with the dynamic potential formulation (3.15) by setting $\lambda = 10, \eta = 1, \beta = 1$.

Figure 3.21 shows the result of these experiments. Figures 3.21a and 3.21b shows that, both for the static and dynamic volume potentials (3.14) and (3.15), the trajectories for the `move_to_ring` gesture do not result perturbed from the presence of the obstacles, since there is no risk of collision.

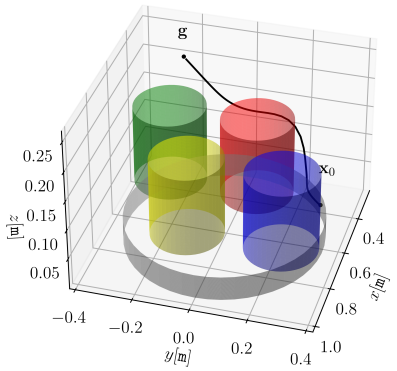
On the other hand, we see that for the `move_to_peg` gesture, the trajectories deviate from the straight-line behavior. Both the static (3.14) and dynamic (3.15) potentials result in a proper obstacle avoidance behavior.



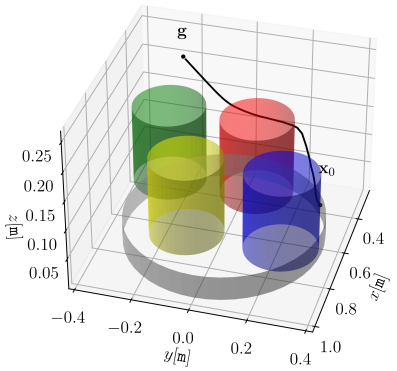
(a) `move_to_ring`, static potential.



(b) `move_to_ring`, dynamic potential.



(c) `move_to_peg`, static potential.



(d) `move_to_peg`, dynamic potential.

Figure 3.21: Moving trajectories for the pick-and-place task with the Panda robot. Axes coordinates are referred to the frame of the base link of the robot. In the bottom plots, the obstacles are larger because both the robot's end-effector and the grasped ring have to avoid colliding with the pegs.

3.4.3. Experiments with simulated Youbots

In this Section, we describe experiments performed with Kuka YouBot models in a simulated environment. These experiments are useful to validate the usage of volumetric obstacle avoidance potentials (3.14) and (3.15) for multi-agent systems applications. The simulation scene is shown in Figure 3.22. It includes three YouBots which can move in a rectangular region defined by four walls (treated as obstacles), with fixed cubes as obstacles on the way. Each robot must reach a specific target position, defined by a platform with the same color as the robot. We assume that the geometry and the positions of the obstacles in the scene are known in advance. The scene is built in the popular CoppeliaSim simulation environment from Coppelia Robotics [117], which allows to simulate the dynamics of the robots and to control

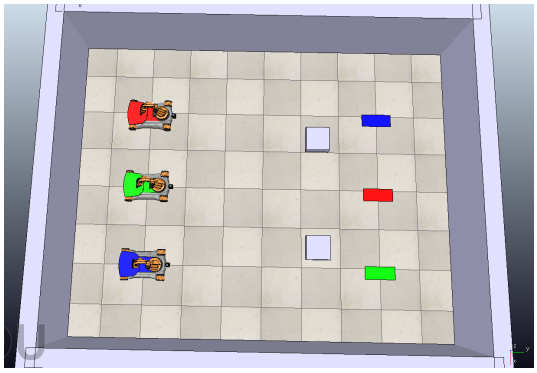
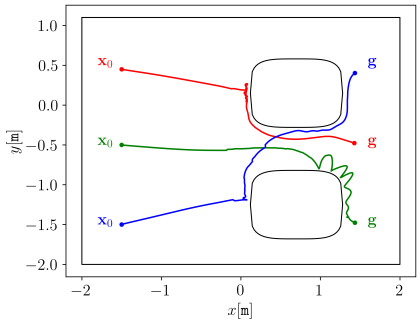


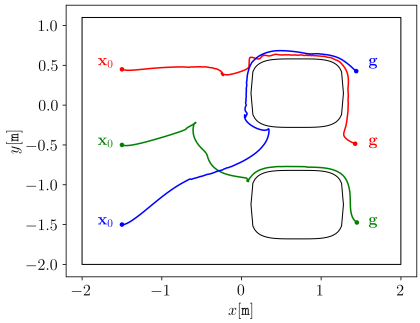
Figure 3.22: The simulation scene in CoppeliaSim for the three YouBots.

them through ROS topics as in real applications.

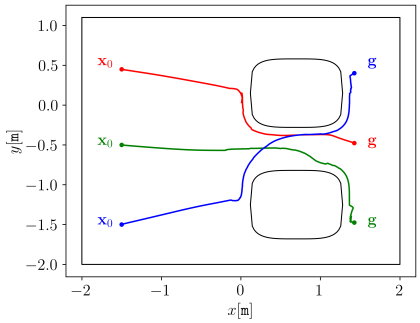
Each Youbot is controlled in position by a DMP with $\mathbf{x}, \mathbf{v} \in \mathbb{R}^2$. We do not control the orientation of the robots along their normal axis, since we are interested in the obstacle avoidance problem for Cartesian DMPs. In order to guarantee the synchronization between the robots, we construct a 6-dimensional DMP, concatenating the components $\mathbf{x}, \mathbf{v}, \dot{\mathbf{v}} \in \mathbb{R}^2$ of position, velocity, and acceleration of each YouBot in a single vector. In this way, the robots share the same canonical system. The obstacle-free trajectory of each robot towards its target is a straight line. In this way, it is clear from the scene that the robots would collide during their motion. Since the objects in the scene are known a priori, one could argue that the collision between the robots could be avoided by computing the trajectories in advance and coordinating the motion of the robots (e.g. tuning the speed of each of them appropriately). In our experiments, we have decided to simulate a more realistic multi-robot task, in which the robots do not know the trajectory of each other in advance. Hence, we model each YouBot as a potential using both the volumetric static (3.14) and dynamic (3.15) potentials. In this way, each YouBot influences the behavior of the other robots. In this way, we show how our framework for obstacle avoidance is suitable for reactive motion planning. At each time step, we build an ellipsoid around each YouBot, setting $n_1 = n_2 = 1$ in (3.13). We control the center point of the YouBots, therefore the semi-axes of the ellipsoid are set as the full dimension of the robot (width $\times$ length) to avoid collisions. When computing the forcing term for each robot, we compute the velocity term in (3.16) as the relative velocity between the robots. We test two different straight-line trajectories, one with a null forcing term and the other with constant speed, to verify the independence of our framework with respect to the specific behavior to be executed. The constant speed trajectory is first generated synthetically; then, the weights are learned as explained in Section 2.3. The DMP parameters are set as as $\mathbf{K} = K\,\mathbf{I}\mathrm{d}_3, \mathbf{D} = D\,\mathbf{I}\mathrm{d}_3$ with $K = 3050, D = 2\sqrt{K} \approx 110.45$, and $\alpha = 4$ for both sets of weights. The trajectories



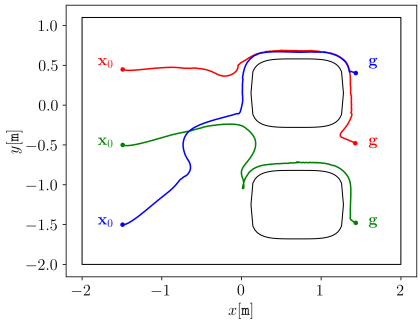
(a) Null weights, static potential.



(b) Constant speed, static potential.



(c) Null weights, dynamic potential.



(d) Constant speed, dynamic potential.

Figure 3.23: DMPs with constant speed and with null weights of the three YouBots in simulation. Obstacles are represented in the scene with the superquadric isopotential approximation, enlarged of the dimensions of the YouBots. The walls are represented as a rectangle containing the robots and the other obstacles for simplicity. Trajectories are referred to as the center points of the robots.

are computed at 1ms step of integration. We model the walls and the fixed obstacles as generalized ellipsoids (enlarged of the dimension of the YouBots), setting $n_1 = n_2 = 2$ in (3.13) to better approximate the sharp edges. We compare the performance of both volumetric static (3.14) and dynamic (3.15) obstacle formulations, modeling the fixed obstacles with both methods. The results are shown in Figure 3.23.

Figures 3.23a-3.23b are obtained using the volumetric static potential method setting $A = 60$ and $\eta = 2$ in (3.14). Figures 3.23c-3.23d are obtained using the volumetric dynamic potential method setting $\lambda = 60$,$\beta = 2$, and $\eta = 2$ in (3.15). We notice that the dynamic potential formulation results in smoother trajectories as the robots move close to the cubes in the scene. Indeed, in Figure 3.23a it is possible to observe that both the red and green YouBots have oscillatory behaviors near the obstacles, as for the red and blue Youbots in Figure 3.23b. On the other hand, oscillations are greatly reduced when the dynamic potential is used, as can be

Figure 3.24: The YouBot with the Realsense D435 camera on its front.

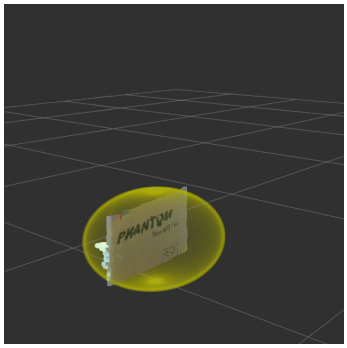


Figure 3.25: Point cloud filtered with the ellipsoid created around the object.

seen in Figures 3.23c and 3.23d.

The main drawbacks of the static formulation can be seen in Figure 3.23a. In particular, the red trajectory shows that the system remains "trapped" for some time near the left side of the obstacle. This follows from the fact that the isopotential has almost flat edges, and thus the repulsive term points towards the left with a small vertical component, while the system's evolution without the obstacle points to the right. Thus, the system keeps oscillating on the edge until it is able to escape. On the other hand, the last portion of the green trajectory shows that while the DMP is trying to go to its nominal behavior (i.e. without obstacle) going down, the perturbation term (that is perpendicular to the obstacle) is pushing the trajectory upwards. This results in oscillations near the obstacle surface.

On the other hand, when using the dynamic potential, the potential (and, consequently, the perturbation term) depends on the velocity of the system and allows smoother behaviors.

In Figures 3.23b and 3.23d some trajectories appear 'pointed', even far from the obstacle (for insance, the green trajectories). This is caused by the fact that each Youbot acts as a moving obstacle for the other robots, making them deviate from their nominal behavior. For instance, in Figure 3.23b, the green Youbot moves upward and slows down while the blue robot is traveling upwards. Once the blue Youbot has bassed by, the green one tries to come back to its nominal behavior, causing the sharp turn.

Similarly, in Figure 3.23d, the blue Youbot slows down while the green one is passing by. At the same time, the green robot accelerates since is 'pushed' by the perturbation term of the blue Youbot. Once there is no longer risk of collision, the green Youbot comes back to its nominal behavior and tries to avoid the lower obstacle, causing the sharp turn.

(a) Start with no obstacle. (b) Avoiding obstacle. (c) Goal reached.

Figure 3.26: Main steps of the YouBot task with the obstacle added to the scene during the execution.

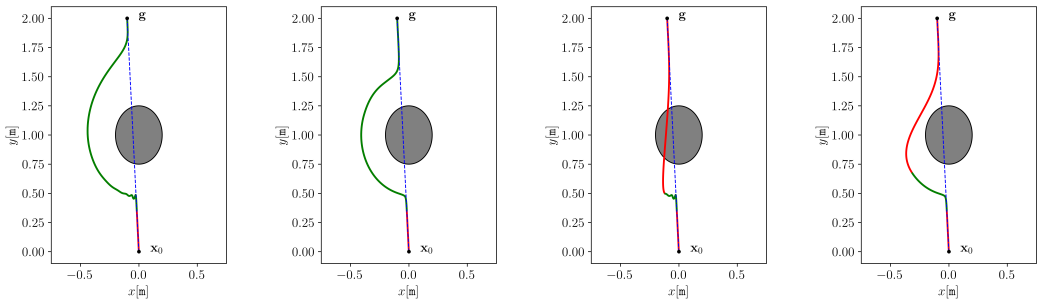
(a) Start with no obstacle. (b) Avoiding obstacle. (c) Goal reached.

Figure 3.27: Main steps of the YouBot task with the obstacle added to and removed from the scene during the execution.

3.4.4. Experiments with One Real YouBot

We test our obstacle avoidance framework with one real YouBot. The robot must move forward in a corridor for 2 meters to a pre-defined target, with an obstacle on its way. We assume that the walls are known in advance and modeled as superquadric potentials. On the contrary, the obstacle on the path of the robot is unknown, and it can be added and removed from the scene during the execution. To this purpose, the YouBot is equipped with a Realsense D435 RGB-depth camera from Intel as shown in Figure 3.24, in order to record the point cloud of the scene in real-time. At each time-step, the point cloud is filtered along the world z-axis to remove the floor and on its own depth to remove points beyond the target. Then it is clustered into separate point clouds for each object in the scene and is registered with the previous point cloud in a common reference frame to update the scene, see Figure 3.25. Finally, an ellipsoid as in (3.13) is fitted with $n_1 = n_2 = 1$, enlarging the axes of the dimensions of the robot (since the motion of the robot is 2-dimensional, we consider only the planar coordinates of the ellipsoid). Fitting a pure ellipsoid rather than a pseudo-ellipsoid ($n_1 = n_2 = 2$) guarantees a smoother perturbation to the trajectory of the robot. The camera and the YouBot controller communicate through a ROS network.

We control the robot with a 2-dimensional DMP with three different behaviors: null forcing term $\mathbf{f} \equiv \mathbf{0}$, constant velocity, and a half-circle trajectory. For each of the three DMPs' behaviors, we test two different scenarios. At first (Figure 3.26) we add a box as an obstacle on the way to the goal right after the YouBot has started moving, and we keep the obstacle in position. In the second scenario, we first add the box in the scene, and then we remove it after some time. The main steps in this scenario are shown in Figure 3.27. In all tests, the obstacle is added after 20`sec`. In the second scenario, the obstacle is removed after 50`sec` from the beginning of the



(a) Static potential, persistent obstacle.

(b) Dynamic potential, persistent obstacle.

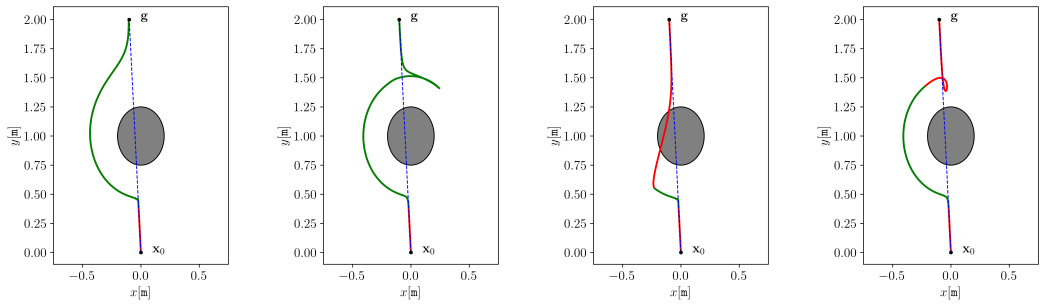
(c) Static potential, temporal obstacle.

(d) Dynamic potential, temporal obstacle.

Figure 3.28: Results for the experiment with the YouBot with null forcing term. In all plots, the blue dashed line shows the desired (obstacle-free) behavior, the gray shade shows the obstacle, the red solid line shows the adapted trajectory when the obstacle is not in the scene, and the green solid line shows the adapted trajectory when the obstacle is in the scene.

test. Thus, the obstacle remain in place for a total of 30`sec`. We test both the static (3.14) and the dynamic (3.15) volumetric potential formulations.
The DMP's parameters are $K = 500$, $D = 2\sqrt{K} \approx 44.72$, and $\alpha = 4$. The obstacle's parameters are $A = 1$, $\eta = 1$ for the static potential (3.14), and $\lambda = 1$, $\beta = 1$, and $\eta = 1$ for the dynamic potential (3.15).

Figures 3.28–3.30 shows the result for these tests. In particular, Figure 3.28 shows the results for the null forcing term DMP; Figure 3.29 shows the result for the constant velocity DMP; while Figure 3.30 shows the results for the half-circle DMP.
From these tests, we see that both static and dynamic methods result in the obstacle being successfully avoided; even if some differences emerge.
The experiments with null forcing term (Figure 3.28) show that, when the obstacle remains in the scene, the dynamic potential (Figure 3.28b) results in a trajectory that deviates less from the desired, un-perturbed, behavior than the static potential (Figure 3.28a). Moreover, when the DMP starts deviating to avoid the obstacle, the static potential results in an oscillatory behavior, while the trajectory obtained with the dynamic potential is smoother. This can be seen also in Figures 3.28c and 3.28d, in which the obstacle is placed and, later, removed. Moreover, these last two tests show that the DMP with the static potential remains 'trapped' for a long time behind the obstacle (all the thirty seconds in which the obstacle is in place), while it is oscillating. On the other hand, the behavior obtained with the dynamic potential immediately deviates from its un-perturbed behavior.
The experiments with the half-circle desired trajectory (Figure 3.30) show simi-



(a) Static potential, persistent obstacle.

(b) Dynamic potential, persistent obstacle.

(c) Static potential, temporal obstacle.

(d) Dynamic potential, temporal obstacle.

Figure 3.29: Results for the experiment with the YouBot with constant velocity DMP. In all plots, the blue dashed line shows the desired (obstacle-free) behavior, the gray shade shows the obstacle, the red solid line shows the adapted trajectory when the obstacle is not in the scene, and the green solid line shows the adapted trajectory when the obstacle is in the scene.

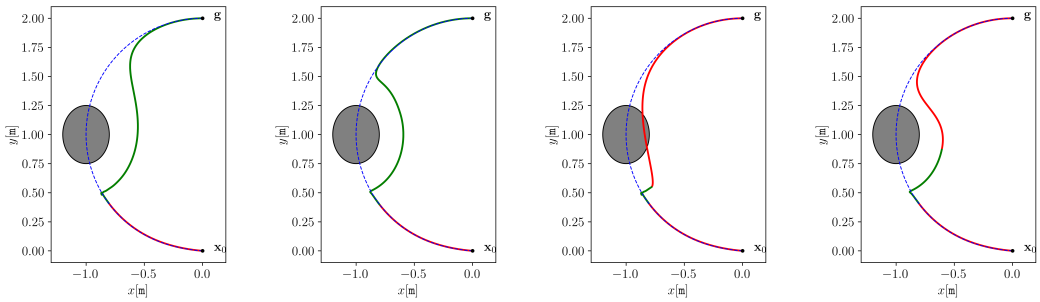
lar results. When the obstacle is kept in place, the dynamic potential results in a trajectory that deviate less from the un-perturbed one (Figures 3.30a and 3.30b). Moreover, as in the null forcing term case, the static potential results in a behavior that remains 'trapped' behind the obstacle for a long time, while the dynamic potential gives a trajectory that manages to deviate immediately from the un-perturbed behavior.

The experiments with the constant-velocity desired behavior (Figure 3.29) show an 'overshoot' of the desired trajectory w.r.t. the un-perturbed one when using the dynamic potential This is due to the fact that, once the obstacle is surpassed, the DMP executed using the static potential is still pushed to the left, while the DMP executed with the dynamic potential has no longer any perturbation due to the obstacle, and it is instead pushed to the right by its own dynamic. This reasoning is more clear to understand from Figure 3.31.

3.5. Conclusions

In the context of Minimally Invasive Surgery, obstacle avoidance is a crucial topic that must be addressed to reach its automation. Indeed, any collision with the patient's tissues may result in organ damage, mining the patient's safety.

In this Chapter, we presented and compared various approaches to implement obstacle avoidance within the DMP framework. In particular, we revised the state of the art of point obstacle avoidance for DMPs and proposed two novel methods (a 'static' and a 'dynamic' one) to treat volumetric obstacles. Both the volumetric



(a) Static potential, persistent obstacle.

(b) Dynamic potential, persistent obstacle.

(c) Static potential, temporal obstacle.

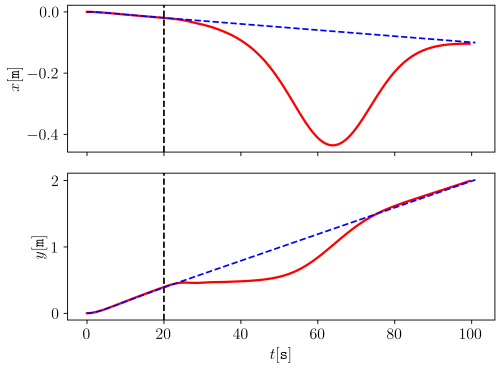
(d) Dynamic potential, temporal obstacle.

Figure 3.30: Results for the experiment with the YouBot for the half-circle DMP. In all plots, the blue dashed line shows the desired (obstacle-free) behavior, the gray shade shows the obstacle, the red solid line shows the adapted trajectory when the obstacle is not in the scene, and the green solid line shows the adapted trajectory when the obstacle is in the scene.

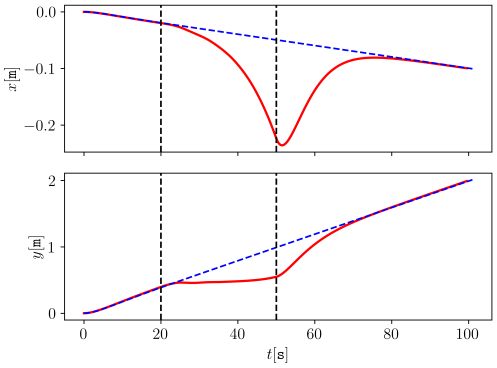
methods require the definition of an *isopotential*, i.e. a function whose zero-level curve coincides with the obstacle surface. Even tho this function is not always easy to compute, we provided a simple formulation (a generalized ellipsoid) that can be used to overcover the obstacle itself.

Tests show that volumetric strategies result in trajectories with fewer oscillations and that remain closer to the desired behavior. Moreover, the dynamic volumetric approach gives more stable trajectories than the static one, reducing oscillations and giving a trajectory that deviates less from the obstacle-free behavior. Finally, while being computationally slower to compute, the volumetric dynamic potential still can be computed fast enough to allow it to be used for on-line robot control.

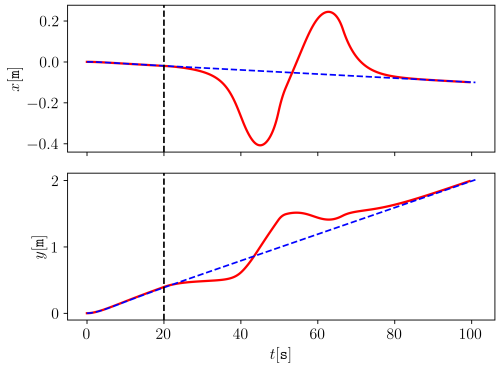
The results presented in this Chapter confirm that DMPs can adapt to obstacles' presence even when obstacles appear suddenly in the scene. This is a crucial aspect in Robotic MIS where the system (or the surgeon) must quickly adapt to unexpected events and scenarios. This confirms that DMPs can be a valid tool in the automation of surgical gestures in the context of autonomous interventions.



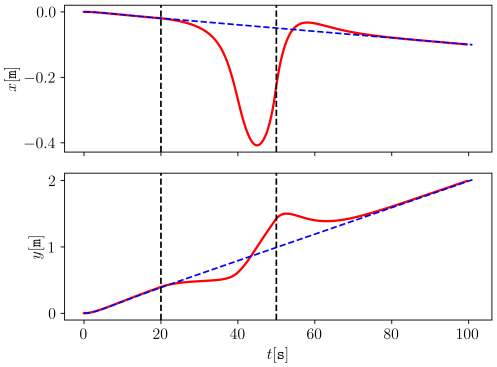
(a) Persistent obstacle, static potential.



(b) Temporal obstacle, static potential.



(c) Persistent obstacle, dynamic potential.



(d) Temporal obstacle, dynamic potential.

Figure 3.31: Solutions for the trajectories obtained with constant velocity DMP on the Youbot. In all plots, the blue dashed line shows the desired solution, and the red solid line shows the obstacle-avoidance behavior. In Figures 3.31a and 3.31c the black dashed line shows the time at which the obstacle is inserted in the scene. In Figures 3.31b and 3.31d the obstacle is present in the time interval marked by the two black dashed vertical lines.

CHAPTER 4

DYNAMIC MOVEMENT PRIMITIVES IN UNIT QUATERNION SPACE

A robot can be controlled both in joint-space or in state-space. In the first case, a Cartesian model is sufficient to describe the whole robot-configuration, since each joint configuration takes value in $\mathbb{R}$. In the second case, when controlling a robot in state space, two aspects must be taken into account: the *position* and the *orientation*. The end-effector's position can be described in Cartesian space. On the other hand, orientations are usually described in non-Cartesian spaces. Two non-Cartesian spaces are usually used to model rigid body's orientations in $\mathbb{R}^3$: the *orthogonal group* SO(3) of all the 3×3 orthogonal matrices, and the unit quaternion space $\mathbb{S}^3$. For a recall, see Appendix D.

In Minimally Invasive Surgery, a huge amount of gestures require precise wrist movements. For instance, tying a knot to close a suture has almost no evolution in the position of the end-effectors, but requires complex changes in the orientation. For this reason, in order to use DMPs to mimic surgical gestures, there is the need for a model able to handle orientations.

In the literature, two DMPs formulations have been proposed to model orientations: one in the orthogonal space SO(3) and one in the unit-quaternion space $\mathbb{S}^3$. The most preferable way to model orientations is the unit quaternion space $\mathbb{S}^3$. Indeed, while both quaternions and rotation matrices provide a singularity-free non-minimal representation of the orientation, the former requires only four parameters, instead of nine of the latter [127]. For this reason, in this Chapter, we present the unit-quaternion formulation.

4.1. DMPs Formulation in Unit Quaternion Space

In 2014, Ude et al. [127] proposed a novel formulation of DMPs for the orientation of the end effector of the robot. In their work, they proposed two formulations: one using rotation matrices and one using quaternions.
Later, in [118], a more general and robust formulation was proposed only for the unit-quaternion space. Here we will present only this last formulation since formulation from [127] can be recovered as a particular case of that proposed in [118]. In general, it is preferable to use quaternions than rotation matrices. Indeed, the latter requires only four parameters to describe the orientation (instead of nine as for rotation matrices), and some maps (like the logarithm) have no discontinuity boundary in this formulation.
For a recall on the theory of quaternions and on how they are used to describe orientations, please refers to Appendix D.

The DMP system in quaternion form reads [118]

$$\begin{cases} \tau\dot{\boldsymbol{\eta}} = K_q\,\mathbf{e}_0(\mathfrak{g},\mathfrak{q}) - K_q\,\mathbf{e}_0(\mathfrak{g},\mathfrak{q}_0)\,s - D_q\,\boldsymbol{\eta} + K_q\,\mathbf{f}_q(s) & \text{(4.1a)} \\ \tau\dot{\mathfrak{q}} = \dfrac{1}{2}\eta * \mathfrak{q} & \text{(4.1b)} \end{cases}$$

Unit quaternion $\mathfrak{q} \in \mathbb{S}^3$ represents the orientation. Unit quaternions $\mathfrak{q}_0$ and $\mathfrak{g} \in \mathbb{S}^3$ are, respectively, the initial and goal quaternion orientation. Vector $\boldsymbol{\eta} = [\eta_1, \eta_2, \eta_3]^\intercal \in \mathbb{R}^3$ is the angular velocity. Vector function $\mathbf{f}_q : \mathbb{R} \to \mathbb{R}^3, s \mapsto \mathbf{f}_q(s)$ is the forcing term, written in terms of basis functions as (2.3). Scalar $s \in (0, 1]$ evolves accordingly the canonical system (2.2). Parameters $K_q, D_q \in \mathbb{R}^+$ are the elastic and damping constants. Scalar $\tau \in \mathbb{R}_+$ is the temporal scaling factor. Function $\mathbf{e}_0 : \mathbb{H} \times \mathbb{H} \to \mathbb{R}^3$ is the *error* between two quaternions. The following two choices are usually used (where the quaternion product is given in Definition D.3, and the conjugate of a quaternion is given in Definiton D.4):

- $\mathbf{e}_0(\mathfrak{q}, \mathfrak{p}) = \mathrm{vec}(\mathfrak{q} * \overline{\mathfrak{p}})$, where function $\mathrm{vec} : \mathbb{H} \to \mathbb{R}^3$ is defined as $\mathrm{vec}(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}) = [b, c, d]^\intercal$;
- $\mathbf{e}_0(\mathfrak{q}, \mathfrak{p}) = 2\log(\mathfrak{q} * \overline{\mathfrak{p}})$, where the logarithm of a quaternion is defined as in (D.10).

Quaternion $\eta \in \mathbb{H}$ in (4.1b) should be intended as a quaternion with null real part, i.e. if $\eta = [\eta_1, \eta_2, \eta_3]^\intercal$, we have $\eta = \eta_1\,\mathbf{i} + \eta_2\,\mathbf{j} + \eta_3\,\mathbf{k}$.
We remark that Equation 4.1a lives in $\mathbb{R}^3$, while Equation 4.1b lives in $\mathbb{H}$.

In the computation of the forcing term, parameters $\boldsymbol{\omega}_i$ are obtained by solving

$$\arg\min_{\boldsymbol{\omega}\in\mathbb{R}^{N+1}} \left\| \frac{\sum_{i=0}^N \boldsymbol{\omega}_i \psi_i(s)}{\sum_{i=0}^N \psi_i(s)} s - \left(\tau\frac{\dot{\boldsymbol{\eta}}}{K_q} - \mathbf{e}_0(\mathfrak{g},\mathfrak{q}) + \mathbf{e}_0(\mathfrak{g},\mathfrak{q}_0)\,s + \frac{D_q}{K_q}\boldsymbol{\eta} \right) \right\|_2^2 .$$

The learning process can be accomplished in the same way as for the Cartesian case presented in Section 2.3.

As for the Cartesian formulation, the unit quaternion DMP system (4.1) converges to the goal with null velocity, when the elastic and damping constants K_q and D_q satisfy a particular identity.

Theorem 4.1 (Saveriano et al. [118])**.** *Dynamical System* (4.1) *with the error map* $\mathbf{e}_0$ *defined as* $\mathbf{e}_0(\mathfrak{q}, \mathfrak{p}) = \text{vec}(\mathfrak{q} * \overline{\mathfrak{p}})$ *globally asymptotically converges to*

$$(\mathfrak{q}, \boldsymbol{\eta}) = (\mathfrak{g}, \mathbf{0}).$$

Proof. As for the Cartesian case, we prove the result for the autonomous formulation of problem (4.1). The result for the general case follows from the fact that both $\mathbf{e}_0(\mathfrak{g}, \mathfrak{q}_0)\, s$ and $\mathbf{f}_q(s)$ goes to zero as t goes to infinity [89]. Moreover, without loss of generality, let us assume $\tau = 1$.
Thus, let us consider the problem

$$\begin{cases} \dot{\boldsymbol{\eta}} = K_q\, \mathbf{e}_0(\mathfrak{g}, \mathfrak{q}) - D_q\, \boldsymbol{\eta} & \text{(4.2a)} \\ \dot{\mathfrak{q}} = \dfrac{1}{2} \eta * \mathfrak{q} & \text{(4.2b)} \end{cases}$$

To compute the equilibria, let us start from (4.2b). we have that, in order to have $\dot{\mathfrak{q}} = 0$ for all $\mathfrak{q}$ we need $\eta = 0$.
Thus, we can now set $\boldsymbol{\eta} = \mathbf{0}$ in (4.2a). The equilibrium must thus satisfy $\mathbf{e}_0(\mathfrak{g}, \mathfrak{q}) = \text{vec}(\mathfrak{g} * \overline{\mathfrak{q}}) = \mathbf{0}$. Since we are working in $\mathbb{S}^3$, the only two quaternions with null vector component are $\mathfrak{q}_1 = 1$ and $\mathfrak{q}_2 = -1$, which represent the same orientation (indeed, we recall that, in general, $\mathfrak{q}$ and $-\mathfrak{q}$ represents the same orientations). From Proposition D.4 we have that, in unit quaternion space $\mathbb{S}^3$, $\mathfrak{q}^{-1} = \overline{\mathfrak{q}}$. Thus, we have that the equilibrium condition $\text{vec}(\mathfrak{g} * \overline{\mathfrak{q}}) = \mathbf{0}$ implies $\mathfrak{g} = \overline{\mathfrak{q}}$, which gives the equilibrium $\mathfrak{q} = \mathfrak{g}$. Thus, we proved that the only equilibrium of the dynamical system (4.2) (and, consequently, for (4.1)) is

$$(\mathfrak{q}, \boldsymbol{\eta}) = (\mathfrak{g}, \mathbf{0}).$$

We now prove the stability of the equilibrium. To do so, let us first introduce the following notation, that follows from the scalar-vector notation introduced in Appendix D:

$$\mathfrak{q} = [q, \mathbf{q}^\intercal]^\intercal, \qquad \mathfrak{g} = [g, \mathbf{g}^\intercal]^\intercal.$$

consider the Lyapunov candidate

$$V(\mathfrak{q}, \boldsymbol{\eta}) = (g - q)^2 + \|\mathbf{g} - \mathbf{q}\|^2 + \frac{1}{2} \frac{\|\boldsymbol{\eta}\|^2}{K_q} \tag{4.3}$$

The Lyapunov function candidate (4.3) is positive definite and vanishes only at the equilibrium $\mathfrak{q} = \mathfrak{g}$, $\eta = \mathbf{0}$. The time derivative of (4.3) is, using the *quaternion propagation* (D.13):

$$\begin{aligned}\dot{V}(\mathfrak{q}, \boldsymbol{\eta}) &= -2(g-q)\dot{q} - 2\langle \mathbf{g}-\mathbf{q}, \dot{\mathbf{q}}\rangle + \frac{1}{K_q}\langle \boldsymbol{\eta}, \dot{\boldsymbol{\eta}}\rangle \\ &= (g-q)\langle \mathbf{q}, \boldsymbol{\eta}\rangle + \Big\langle \mathbf{g}-\mathbf{q}, (q\mathbf{I}\mathrm{d}_3 - \mathbf{S}(\mathbf{q}))\boldsymbol{\eta}\Big\rangle + \frac{1}{K_q}\langle \boldsymbol{\eta}, \dot{\boldsymbol{\eta}}\rangle \end{aligned} \tag{4.4}$$

where $\mathbf{S}(\cdot)$ is the skew symmetrix matrix defined in (D.14) and satisfies $\mathbf{u}\times\mathbf{v} = \mathbf{S}(\mathbf{u})\,\mathbf{v}$ for any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$. Substituting (4.2a) in (4.4), we obtain

$$\begin{aligned}\dot{V}(\mathfrak{q}, \boldsymbol{\eta}) &= (g-q)\langle \mathbf{q}, \boldsymbol{\eta}\rangle + \Big\langle \mathbf{g}-\mathbf{q}, (q\mathbf{I}\mathrm{d}_3 - \mathbf{S}(\mathbf{q}))\boldsymbol{\eta}\Big\rangle + \frac{1}{K_q}\Big\langle \boldsymbol{\eta}, K_q \mathrm{vec}(\mathfrak{g} * \overline{\mathfrak{q}}) - D_q\boldsymbol{\eta}\Big\rangle \\ &= (g-q)\mathbf{q}^\intercal\boldsymbol{\eta} - (\mathbf{g}-\mathbf{q})^\intercal(q\mathbf{I}\mathrm{d}_3 - \mathbf{S}(\mathbf{q}))\boldsymbol{\eta} + (-g\mathbf{q}^\intercal + q\mathbf{g}^\intercal + (\mathbf{S}(\mathbf{q})\mathbf{g})^\intercal)\eta - \frac{D_q}{K_q}\|\boldsymbol{\eta}\|^2, \end{aligned} \tag{4.5}$$

where in the last step we used the explicited term $\mathrm{vec}(\mathfrak{g} * \overline{\mathfrak{q}})$ and written the scalar product as row-column vectors product. Next, by observing that, since matrix $\mathbf{S}(\mathbf{q})$ is skew symmetric, the following identity holds

$$(\mathbf{S}(\mathbf{q})\mathbf{g})^\intercal = g^\intercal\mathbf{S}^\intercal(\mathbf{q}) = -\mathbf{g}^\intercal\mathbf{S}(\mathbf{q}).$$

Now, by re-arranging the sum, we can rewrite (4.5) as

$$\dot{V}(\mathfrak{q}, \boldsymbol{\eta}) = (g\mathbf{q}^\intercal - q\mathbf{q}^\intercal - q\mathbf{g}^\intercal + \mathbf{g}^\intercal\mathbf{S}(\mathbf{q}) + q\mathbf{q}^\intercal - \mathbf{q}^\intercal\mathbf{S}(\mathbf{q}))\boldsymbol{\eta} - (g\mathbf{q}^\intercal + q\mathbf{g}^\intercal - \mathbf{g}^\intercal\mathbf{S}(\mathbf{q}))\boldsymbol{\eta} - \frac{D_q}{K_q}\|\boldsymbol{\eta}\|^2.$$

Since $\mathbf{S}(\mathbf{q})\mathbf{q} = \mathbf{q}^\intercal\mathbf{S}(\mathbf{q}) = \mathbf{0}$, and by removing all the opposite terms in the sum, we have that

$$\dot{V}(\mathfrak{q}, \boldsymbol{\eta}) = -\frac{D_q}{K_q}\|\boldsymbol{\eta}\|^2,$$

which implies that $\dot{V}(\mathfrak{q}, \boldsymbol{\eta}) \leq 0$ and vanishes if and only if $\boldsymbol{\eta} = \mathbf{0}$.
Thus, by LaSalle's invariance theorem, we have proven the stability of (4.2) when the error map $\mathbf{e}_0$ is set to be the vector map. □

Remark 4.1. Theorem 4.1 holds true even if we choose the error function $\mathbf{e}_0$ to be the logarithmic map: $\mathbf{e}_0(\mathfrak{q}, \mathfrak{p}) = 2\log(\mathfrak{q} * \overline{\mathfrak{p}})$. In this case, the result can be proved by using

$$V(\mathfrak{q}, \boldsymbol{\eta}) = (v_g - v_q)^2 + \left\|\mathbf{u}_g - \mathbf{u}_q\right\|^2 + \frac{1}{2}\|\boldsymbol{\eta}\|^2$$

as Lyapunov candidate, and selecting

$$K_q = \frac{\|\mathrm{vec}(\mathfrak{g} * \overline{\mathfrak{q}})\|}{2\arccos(\mathrm{scal}(\mathfrak{g} * \overline{\mathfrak{q}}))}\mathbf{I}\mathrm{d}_3$$

as stiffness gain. Map scal denotes the real part of a quaternion: $\text{scal}(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}) = a$. This non-linear stiffness gain has a singularity when $\mathfrak{g}$ and $\mathfrak{q}$ are aligned. Thus, for stability purposes, map $\mathbf{e}_0$ should be set to be the vector map, $\mathbf{e}_0(\mathfrak{q}\mathfrak{p}) = \text{vec}(\mathfrak{q} * \overline{\mathfrak{p}})$, instead of the logarithmic one, $\mathbf{e}_0(\mathfrak{q}\mathfrak{p}) = 2\log(\mathfrak{q} * \overline{\mathfrak{p}})$.

The integration of dynamical system (4.1) proceeds in two steps.
At first, $\boldsymbol{\eta}(t + \delta_t)$ is computed using any numerical integration scheme. In our implementation, we use the Exponential Euler method presented in Appendix A.1. Equation (4.1a) can be written as

$$\tau\dot{\boldsymbol{\eta}} = \boldsymbol{\Xi}_q\,\boldsymbol{\eta} + \boldsymbol{\beta}_q(\mathfrak{q}, s),$$

with

$$\boldsymbol{\Xi}_q = -D_q\,\mathbf{Id}_3$$

and

$$\boldsymbol{\beta}_q(\mathfrak{q}, s) = K_q(\mathbf{e}_0(\mathfrak{g}, \mathfrak{q}) - \mathbf{e}_0(\mathfrak{g}, \mathfrak{q}_0)s + \mathbf{f}_q(s)).$$

Thus, the numerical scheme to integrate (4.1a) with time step δ_t is:

$$\boldsymbol{\eta}(t + \delta_t) = \boldsymbol{\eta}(t) + \frac{\delta_t}{\tau}\boldsymbol{\Phi}_1\left(\frac{\delta_t}{\tau}\,\boldsymbol{\Xi}_q\right)\Big(\boldsymbol{\Xi}_q\,\boldsymbol{\eta}(t) + \boldsymbol{\beta}_q(\mathfrak{q}(t), s(t))\Big), \tag{4.6}$$

where map $\boldsymbol{\Phi}_1(\cdot)$ is defined in (A.5).
For the integration of (4.1b) we start from the following explicit scheme:

$$\mathfrak{q}(t + \delta_t) = \exp\left(\frac{\delta_t}{2}\,\frac{\boldsymbol{\eta}(t)}{\tau}\right) * \mathfrak{q}(t). \tag{4.7}$$

We can rewrite it as an implicit scheme by considering the time step to be negative, $\delta_t = -\delta_t$, obtaining that (4.7) now reads

$$\mathfrak{q}(t - \delta_t) = \exp\left(-\frac{\delta_t}{2}\,\frac{\eta(t)}{\tau}\right) * \mathfrak{q}(t).$$

This last expression can be written, by translating t by the quantity δ_t as

$$\mathfrak{q}(t) = \exp\left(-\frac{\delta_t}{2}\,\frac{\eta(t + \delta_t)}{\tau}\right) * \mathfrak{q}(t + \delta_t).$$

Thus, at each time step we can compute the next value for $\mathfrak{q}$ using the following integration scheme:

$$\mathfrak{q}(t + \delta_t) = \left(\exp\left(-\frac{\delta_t}{2}\,\frac{\eta(t + \delta_t)}{\tau}\right)\right)^{-1} * \mathfrak{q}(t), \tag{4.8}$$

where, we recall, quaternion η is the quaternion with null real part and vector component equals to $\boldsymbol{\eta}$. A single step of the integration of the quaternion DMPs (4.1) is summarized in Algorithm 4.1. Step 4 of the algorithm shows a normalization step. From a theoretical point of view, the result of the integration scheme (4.8) is a unit quaternion. However, when computing it numerically, the norm may be no unitary, so a normalization step is required to avoid divergence of the numerical method.

Algorithm 4.1 Quaternion DMPs integration - step.

Input: Vector field parameters $\tau, \boldsymbol{\Xi}_q$, and $\boldsymbol{\beta}_q(\cdot,\cdot)$, present solution $(\boldsymbol{\eta}(t), \mathfrak{q}(t), s)$, time-step size δ_t.

Output: New time-step soluztion $(\boldsymbol{\eta}(t+\delta_t), \mathfrak{q}(t+\delta_t))$

1: Integration of the angular velocity $\boldsymbol{\eta}$

$$\boldsymbol{\eta}(t+\delta_t) = \boldsymbol{\eta}(t) + \frac{\delta_t}{\tau}\boldsymbol{\Phi}_1\left(\frac{\delta_t}{\tau}\boldsymbol{\Xi}_q\right)\Big(\boldsymbol{\Xi}_q\,\boldsymbol{\eta}(t) + \boldsymbol{\beta}_q(\mathfrak{q}(t), s(t))\Big).$$

2: Integration of the canonical system

$$s(t+\delta_t) = \exp\left(\frac{-\alpha\,\delta_t}{\tau}\right)s(t).$$

3: Integration of the quaternion $\mathfrak{q}$

$$\mathfrak{q}(t+\delta_t) = \left(\exp\left(-\frac{\delta_t}{2}\,\frac{\boldsymbol{\eta}(t+\delta_t)}{\tau}\right)\right)^{-1} * \mathfrak{q}(t).$$

4: Normalization of the quaternion

$$\mathfrak{q}(t+\delta_t) = \frac{\mathfrak{q}(t+\delta_t)}{\|\mathfrak{q}(t+\delta_t)\|}.$$

4.2. DMPs Formulation for Poses

When using DMPs to control a robot's end-effector, both Cartesian and unit-quaternion DMPs must be considered so to model both the end-effector position and orientation. Each DMP system has to be synchronized with the other. Otherwise, the learned gesture would not be executed correctly since the Cartesian (i.e. position) component would evolve differently than the unit-quaternion (i.e. orientation) one. To do so, the same canonical system (2.2) should be shared between the Cartesian and the unit quaternion systems.

In summary, to control the *pose* of a robot (i.e. both its position and orientation), we use the following *poses DMPs* system

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g}-\mathbf{x}) - \mathbf{D}\,\mathbf{v} - \mathbf{K}(\mathbf{g}-\mathbf{x}_0)s + \mathbf{K}\mathbf{f}(s) & \text{(4.9a)} \\ \tau\dot{\mathbf{x}} = \mathbf{v} & \text{(4.9b)} \\ \tau\dot{\boldsymbol{\eta}} = K_q\,\mathbf{e}_0(\mathfrak{g},\mathfrak{q}) - K_q\,\mathbf{e}_0(\mathfrak{g},\mathfrak{q}_0)\,s - D_q\,\boldsymbol{\eta} + K_q\,\mathbf{f}_q(s) & \text{(4.9c)} \\ \tau\dot{\mathfrak{q}} = \dfrac{1}{2}\eta * \mathfrak{q} & \text{(4.9d)} \\ \tau\dot{s} = -\alpha s & \text{(4.9e)} \end{cases}$$

Equations (4.9a) and (4.9b) are the Cartesian DMPs (2.10); equations (4.9c) and (4.9d) are the unit-quaternion DMPs (4.1); and equation (4.9e) is the canonical

Algorithm 4.2 DMP pose integration - step

Input: DMPs parameters $\mathbf{K}, \mathbf{D}, \mathbf{x}_0, \mathbf{g}, K_q, D_q, \mathfrak{q}_0, \mathfrak{g}, \alpha$, and τ; forcing terms $\mathbf{f}, \mathbf{f}_q$, present state $\mathbf{x}(t), \mathbf{v}(t), \mathfrak{q}(t), \boldsymbol{\eta}(t)$, and $s(t)$; and time step δ_t

Output: Next-time solution $\mathbf{x}(t+\delta_t), \mathbf{v}(t+\delta_t), \mathfrak{q}(t+\delta_t), \boldsymbol{\eta}(t+\delta_t)$, and $s(t+\delta_t)$

1: Compute the Cartesian vector field parameters $\boldsymbol{\Xi}$ and $\boldsymbol{\beta}$ using (2.19)
2: Compute the next Cartesian component $[\mathbf{v}(t+\delta_t)^\intercal, \mathbf{x}(t+\delta_t)^\intercal]^\intercal$ using the exponential Euler method from Section A.1
3: Compute the next angular velocity component $\boldsymbol{\eta}(t+\delta_t)$ using (4.6)
4: Compute the next quaternion component $\mathfrak{q}(t+\delta_t)$ using (4.8)
5: Normalize the quaternion component $\mathfrak{q}(t+\delta_t)$
6: Compute the next canonical system value $s(t+\delta_t)$

system (2.2).

The learning process can be straightforwardly implemented. Indeed, it is enough to decompose the learning of the Cartesian component from the unit-quaternion one.

During the execution phase, each step of the numerical integrations can be achieved in four steps:

Step 1. Integrate the Cartesian compoent (4.9a) and (4.9b);

Step 2. Integrate the angular velocity component (4.9c);

Step 3. Integrate the quaternion compoent (4.9d), and normalize the resulting quaternion;

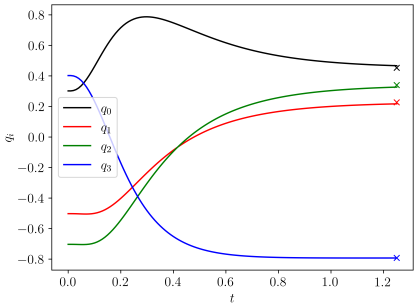
Step 4. Integrate the canonical system (4.9e).

The algorithm is summarized in Algorithm 4.2.

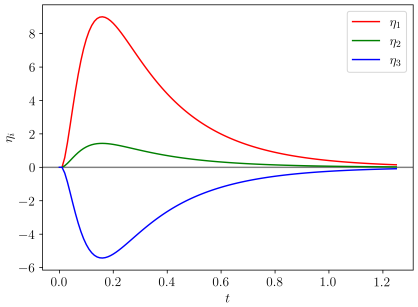
4.3. Experiments

To show the convergence and learning properties of the quaternion DMPs system (4.1) we present two tests. The first adopts a null forcing term $\mathbf{f}_q(s) \equiv \mathbf{0}$ and shows the convergence of the system to the desired goal configuration. In the second test, we learn one desired quaternion evolution and learn it. These tests show that the unit-quaternion DMP system is able both to learn the behavior and adapt to new desired goal positions.

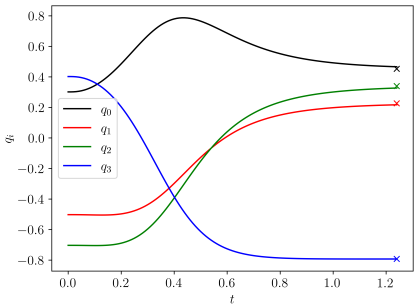
At first, we generate a quaternion DMP with null weights, $\boldsymbol{\omega}_i = \mathbf{0} \in \mathbb{R}^3, \forall i = 0, 1, \ldots, N$. The DMPs' elastic and damping terms are set to $K_q = 500$ and $D_q = 2\sqrt{K_q} \approx 44.72$. The canonical system (2.2) decrease rate is set to $\alpha = 4$. We set the starting quaternion configuration to $\mathfrak{q}_0 = 0.30151134 - 0.50251891\mathbf{i} - 0.70352647\mathbf{j} +$



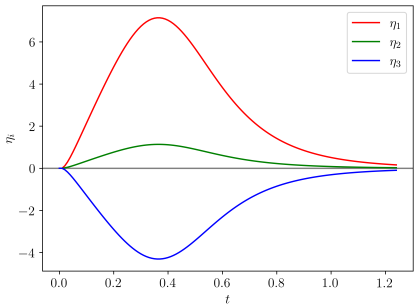
(a) Quaternion evolution, logarithmic map.



(b) Angular velocity evolution, logarithmic map.



(c) Quaternion evolution, vector map.

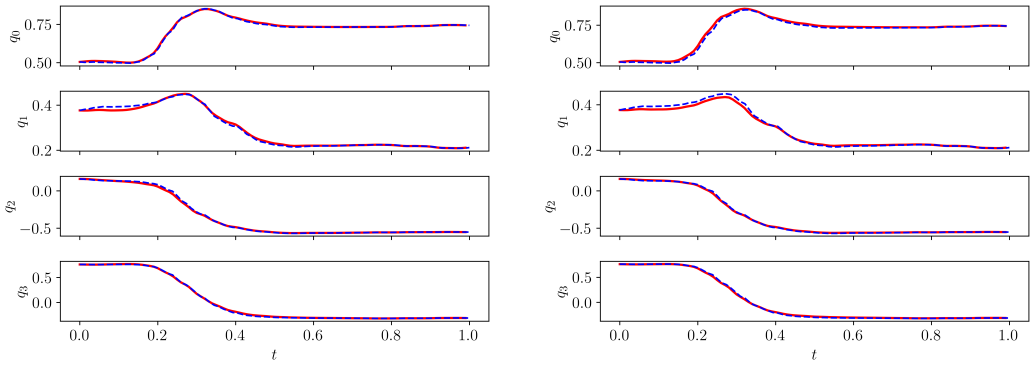


(d) Angular velocity evolution, vector map.

Figure 4.1: Evolution of a unit-quaternion DMP with null weights $\mathbf{f}_q \equiv \mathbf{0}$. In Figures 4.1a, 4.1c, the goal quaternion position is marked with an '×' for each of the four components.

$0.40201513\mathbf{k}$ and the goal to $\mathfrak{q}_g = 0.45291081 + 0.22645541\mathbf{i} + 0.33968311\mathbf{j} - 0.79259392\mathbf{k}$. Plots in Figure 4.1 show the results of the DMP evolution. As it can be seen, both when using, as error function, the logarithmic map $\mathbf{e}_0(\mathfrak{q}, \mathfrak{p}) = 2\log(\mathfrak{q} * \overline{\mathfrak{p}})$ and the vector map $\mathbf{e}_0(\mathfrak{q}, \mathfrak{p}) = \mathrm{vec}(\mathfrak{q} * \overline{\mathfrak{p}})$, convergence to the goal position is achieved. However, as it should be expected, the behaviors are different, even if slightly. In particular, the angular velocity $\boldsymbol{\eta}$ has both a faster and greater initial increase in all the components when using the logarithmic map.

Next, we test the ability of unit-quaternion DMPs (4.1) to learn the desired behavior. To do so, we use the *JIGSAW* dataset [39]. In particular, we select one of the three expert users ('E') performing a 'Needle Passing' task and extract a 'G8' gesture (i.e. "orienting needle"). In Figure 4.2 both the desired trajectory and the learned behavior are shown for each of the error maps (logarithmic and vector).



(a) Learning of a quaternion evolution - logarithmic map.

(b) Learning of a quaternion evolution - vector map.

Figure 4.2: Learning of a quaternion behavior. In both Figures, the blue dashed line shows the desired behavior, and the red solid line represents the evolution of the learned DMP.

4.4. Conclusions

In Minimally Invasive Surgery, wrist movements are crucial for executing multiple important tasks such as suturing and knot-tying. These types of movements require a change in the orientation of the end-effector, rather than in its position. Thus, automation of surgical tasks requires a method to learn trajectories in the space of orientations.

In this Chapter, we presented the DMP formulation in the space of unit-quaternions $\mathbb{S}^3$. Even tho other parametrizations of the orientation space can be used (such as Euler angles and 3×3 orthogonal matrices), unit quaternions are usually preferred in robotic contexts since it is the singularity-free parametrization that requires the smaller number of parameters [127].

We performed some synthetic experiments to show that the desirable properties of the Cartesian formulation of Dynamic Movement Primitives are true also for the unit-quaternion formulation. Indeed, any gesture can be learned and generalized to new goal positions. Moreover, the convergence of the system to the goal configuration is guaranteed.

CHAPTER 5

DMPs FOR ON-LINE CONTROL

In the previous Chapters, we discussed how to use DMPs to learn trajectories that can be generalized to new desired configurations. However, we never took into account various problems and difficulties that may occur in real robot applications. If the DMP is used to generate the trajectory off-line, no particular problems arise. Indeed, the DMP can be used to generate the trajectory, and then the planner of the robot seeks the most efficient way to follow the trajectory. On the other hand, if we aim to use the DMPs to control the robot on-line, some difficulties may happen. For instance, the trajectory generated from the DMP may be too fast to be executed due to the joint limits of the robot.
In this Chapter, we discuss a modification, called *phase stopping* [63, 127], of the canonical system (2.2) which result in an automatic decrease in the speed of the DMP when the robot is not able to execute the desired trajectory.

5.1. PHASE STOPPING

One easy, yet inefficient, way to deal with the limits of joints' velocity is to manually slow down the execution of the trajectory by increasing the temporal scaling factor τ. However, this method has two major drawbacks. First, it is hard to determine in advance how much the trajectory should be slowed down, and this may result in an excessively slow final execution. Second, if only a portion of the trajectory is too fast for the robot, one would prefer to slow down only that portion. This cannot be done by simply changing the value of the time scaling factor τ. Indeed, a modification of τ would change the speed of execution of the whole trajectory.
A more efficient way consists of the *phase stopping* [63, 127]. It is a modification

of the canonical system (2.2) which automatically slows down the evolution of the DMP when the present robot configuration is distant from the position planned from the DMP.

The original canonical system (2.2) can be replaced with

$$\tau \dot{s} = -\frac{\alpha}{1 + \tilde{\alpha}\,\|\tilde{\mathbf{x}} - \mathbf{x}\| + \tilde{\beta}\,\mathrm{dist}(\tilde{\mathfrak{q}} - \mathfrak{q})}\, s, \tag{5.1}$$

where $\mathbf{x}$ and $\mathfrak{q}$ are, respectively, the Cartesian and unit quaternion states of the DMP; and $\tilde{\mathbf{x}}$ and $\tilde{\mathfrak{q}}$ are the Cartesian and unit quaternion state of the robot. Constant $\alpha > 0$ gives the rate of deacy of s as in (2.2). Constants $\tilde{\alpha}, \tilde{\beta} \geq 0$ govern the speed decay of the canonical system depending on the error between Cartesian and unit-quaternion configurations respectively.

The idea behind this modification is that the evolution of the canonical system, which controls the evolution of the forcing term $\mathbf{f}$, automatically slows down when the robot is not able to properly follow the planned trajectory.

To validate this method we present the following synthetic test.

Example 5.1. In this test, we simulate a system with limited velocity in each direction. At each time step, we perform an integration step of the DMP. Then, we simulate a control $\mathbf{g}_k(\mathbf{x} - \tilde{\mathbf{x}})$ that, at each time step, moves the system to the new planned position

$$\tilde{\mathbf{x}}(t + \delta_t) = \tilde{\mathbf{x}}(t) + \mathbf{g}_k(\mathbf{x}(t + \delta_t) - \tilde{\mathbf{x}}(t)). \tag{5.2}$$

We remark that if $\mathbf{g}_k$ is the identity map then the system $\tilde{\mathbf{x}}$ has the same evolution than the planned trajectory $\mathbf{x}$. In this example, we set $\mathbf{g}_k$ to be a "limiting function" with components defined as

$$\mathbf{g}_k(\mathbf{y})|_i = [\min(\mathbf{y}|_i, k)]\,|_i.$$

Basically, function $\mathbf{g}_k$ bounds the velocity that the system $\tilde{\mathbf{x}}$ can achieve in each component.

To test the phase stopping, we start by learning the desired trajectory,

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} (t-1)\,(t-2)^2 \\ (t-1)^2\,(t-2) \end{bmatrix}, \qquad t \in [0.8, 2.2]\,.$$

The DMP's parameters are set to $K = 1050$, $D = 2\sqrt{K} \approx 64.81$, and $\alpha = 4$. During the execution, we set the integration time-step to $\delta_t = 0.01$ and the bound of function $\mathbf{g}_k$ to $k = 0.01$.

We then execute both the DMP and the control (5.2), both with the classical canonical system (2.2) and the phase stopping modification (5.1) with $\tilde{\alpha} = 200$.[1]

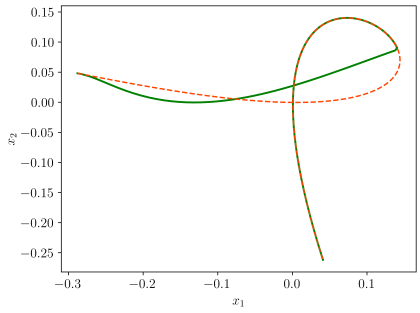
[1] Since we control only the position and not the orientation, we have $\tilde{\beta} = 0$.

Figure 5.1 shows the results of this test. Figure 5.1a shows that, without the phase stopping, the system is not able to keep up with the DMP system. On the other hand, Figure 5.1b shows that if the phase stopping is implemented, the system is able to properly follow the planned DMP. Figures 5.1c and 5.1d show how the DMP with phase stopping slows down to not stray from the system. Finally, Figure 5.1e shows the evolution of the canonical system with and without phase stopping.

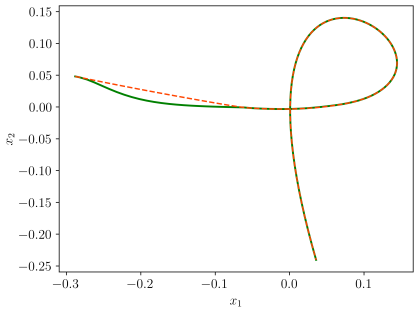
These tests show how DMPs can be used to control a robot in real-time in a reliable way. If we aim at using DMPs to generate trajectories off-line, and then execute them, phase stopping has not to be implemented, since the robot's controller will deal with the velocity of execution of the desired trajectory. However, in surgical scenarios, it is necessary to have a framework able to react to unforeseen situations. Thus, the ability of DMPs to be used as an on-line controller is crucial to use them in Robotic Minimally Invasive Surgery applications.

5.2. Conclusion

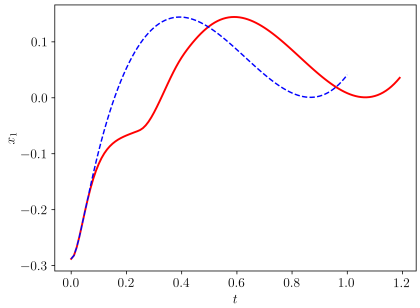
In this Chapter, we presented a modification of the canonical system (2.2), called *phse stopping*, that allows to automatically slow down the execution of the DMP when the robot is not able to follow the planned trajectory due to joints' limit. This is of crucial importance when using DMPs to control a real robot because, otherwise, the obtained execution would be different from the desired one, possibly failing the task. Phase stopping is tested on a synthetic test that shows how a DMP slows down when the robot is not able to follow it. The test shows that the DMP slows down when the robot position is far from the desired behavior. This allows the robot to properly follows the desired behavior, whithout deviating from it.



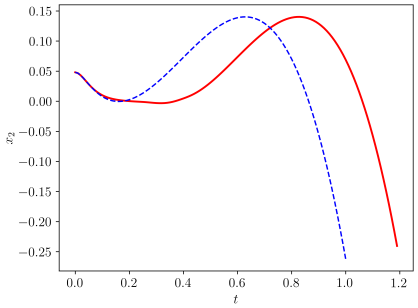
(a) Comparison between the planned DMP (in solid green) and the trajectory executed by the system (in dashed orange) without phase stopping.



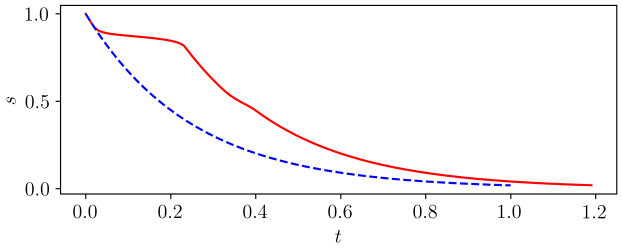
(b) Comparison between the planned DMP (in solid green) and the trajectory executed by the system (in dashed orange) with phase stopping.



(c) Comparison between the planned DMPs with (in solid red) and without (in dashed blue) phase stopping for the horizontal component.



(d) Comparison between the planned DMPs with (in solid red) and without (in dashed blue) phase stopping for the vertical component.



(e) Comparison between the canonical system evolution with (in solid red) and without (in dashed blue) phase stopping.

Figure 5.1: Results for Example 5.1

CHAPTER 6

TASK AUTOMATION USING DMPS

To validate the usage of Dynamic Movement Primitives in the automation of Robotic Minimally Invasive Surgery, we implemented two frameworks to execute a *Peg & Ring* task. This task is of particular interest in surgical robotics applications. Indeed, it is a standard training and skill-assessment exercise for surgeons [124, 55] that presents many challenges of real surgery, such as grasping and transferring [44]. Thus, automation of this task with the daVinci surgical robot will be an important first step in the automation of more complex surgical tasks.

In particular, we developed two frameworks: an *ontology-based* framework [44], validated on the Panda industrial manipulator, and an *answer set programming* framework [45], validated on the daVinci surgical robot.

The fact that both frameworks use DMPs to model the gestures and are capable to correctly perform a surgical-related task confirms that DMPs are a proper tool in the automation of Robotic Minimally Invasive Surgery.

6.1. THE TASK

In this Section, we describe in detail the Peg & Ring task, presenting different scenarios that may change its complexity. Clearly, a more complex formulation of the task is also more challenging to automate.

The *Peg & Ring* task consists of placing colored rings on the same-colored pegs in given color order. The setup consists of a (usually of circular shape) base in which the pegs are inserted so to remain vertical and of a set of rings posed on the base. We can think about three different levels of increasing complexity of the setup:

T1 Only the colored pegs are on the base, and the ring are placed on the base

without already being on a peg;

T2 The base has both colored and white (or gray) pegs; the rings are inserted in the gray pegs and the user must extract the rings before inserting it in the right peg;

T3 The base has both colored and non-colored pegs; the ring can be already inserted on any of the pegs (gray, right-colored, or wrong-colored), or be placed on the base. This formulation of the task is the most challenging of the three because the user (and, thus, the autonomous system) must plan in advance which rings have to be ignored (if already inserted in a right-colored peg), and which pegs have to be freed (if occupied by a wrong-colored ring).

The task can be executed in two different ways: single hand or bimanual. The first case is the simplest: the user has to grab the ring, carry it, and place it (or drop it) using always the same robot's end-effector. In the second case, the workspace is split into two halves and each arm should operate in one of them. Thus, when the ring is in the same half of the same-colored peg, the arm grabbing the ring is the same that place it. Otherwise, when a ring is placed in the opposite half of the workspace with respect to the ring, one arm has to grab the ring and move it towards the workspace's center, where the other arm will grab the ring and carry it towards the same-colored peg. Thus, in this case, there is an additional gesture: a *transferring* moment in which the ring is passed from one end-effector to the other.

6.2. Ontology-Based Framework

In [44] we proposed a knowledge-based model to automate the Peg & Ring task consisting of a modular framework with hierarchical reasoning. The framework consists of two levels: a *task level*, which implements the task reasoner, and a *control level*, which implements the dynamic controller with DMPs. At the task level (the higher in the hierarchy) an ontology encodes the prior knowledge as rules and verifies the preconditions for the execution of the main actions. At the control level (the lower) motion planning is implemented with DMPs. A block diagram summarizing the framework is shown in Figure 6.1.

Ontologies provide a knowledge representation of the key concepts related to the domain of discourse with properties, relationships, and constraints [49]. Moving from a description of the surgical task as a set of *actions* (i.e. elementary operations) and transition rules between them, the ontology specifies the properties of the task. Ontologies offer the opportunity to easily determine the task description by the users, which enhances the interpretability and reliability of the reasoner.

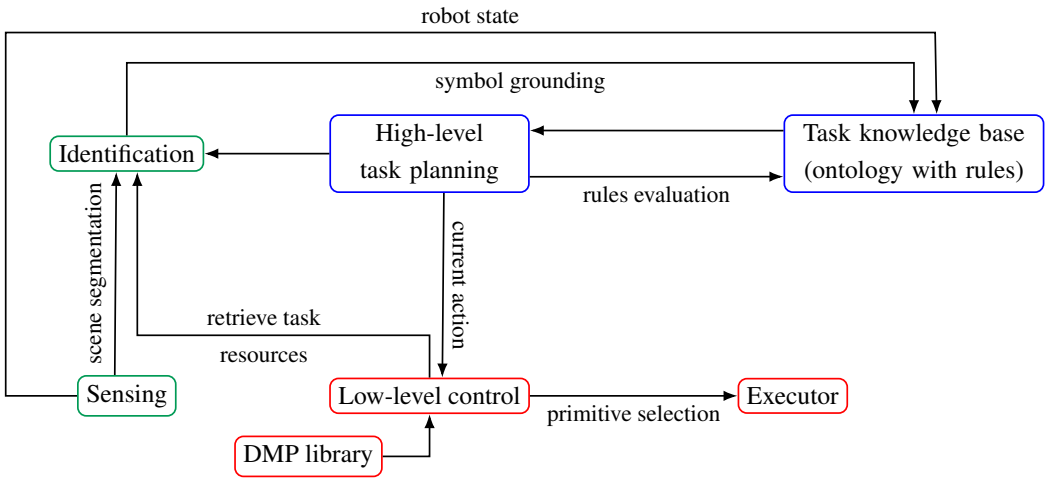


Figure 6.1: Diagram of the ontology-based framework for task automation. Red blocks represent the low-level control, blue blocks the high-level reasoning, and green blocks the perception modules.

This aspect is particularly important in MIS to guarantee the safety requirements in surgery [44].

This framework is implemented for a single-handed version of the task T1, in which three rings of different colors (red, blue, and green) are on the base and have to be moved onto the same-colored pegs. A Finite State Machine (FSM) is derived from the FSM presented in [55] for the *Peg Transfer* surgical task and adapted to the single manipulator case. The Peg Transfer task requires the user to pick, one at a time, six triangles with the non-dominant hand, transfer it to the dominant hand, and place the triangle onto any of the pegs. The Peg & Ring task is slightly more complicated since the user cannot decide the peg in which place the ring. We decided to adapt the FSM from [55] because the tasks, despite some differences, have a lot of similarities. For instance, the actions have similar semantic meanings. Each mode of the FSM is associated with one action. An action is a single gesture with a specific semantic meaning. In this task, three actions are performed:

1. `move_to_start`: the robot initially moves to a standard configuration at the beginning of the task;

2. `move_and_grasp`: the robot goes to the next ring in the sequence, and then grasp it;

3. `carry_and_leave`: the robot carries the grasped ring to the corresponding peg, and then opens the gripper to release the ring.

Additionally, a failure condition is raised when the ring falls while executing the `carry_and_leave` mode. In this case, the robot re-opens the gripper, and the FSM

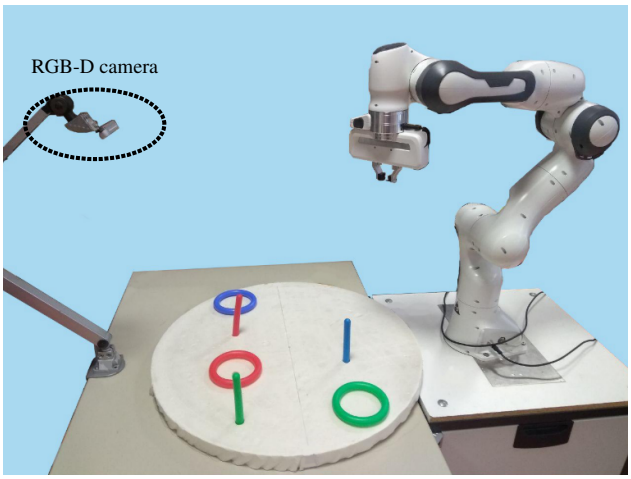


Figure 6.2: Setup for the automated Peg & Ring task with the Panda industrial manipulator.

comes back to the `move_and_grasp` mode so that the framework replans the task from the last ring.

To effectively handle the planning process at the task level, we exploit spatio-temporal reasoning with the ontology. Spatial reasoning is achieved by classifying objects in the environment. Classes and their instances (`pegs` and `rings`) have geometric properties (`spatial_pose`) and `id_color` semantic property, which identifies the specific color. The pose of the pegs is computed at the beginning of the task and stored since they are fixed. A `gripper` class instance is also generated to reason on the status of the robot's end-effector. We follow Allen's temporal interval algebra to do qualitative temporal planning between two actions, e.g. `move_and_grasp` takes place before `carry_and_leave`. To implement logical task planning, the preconditions and effects are implemented in a form of production rules.

The preconditions and effects are:

R1 *Rule-1 to start classification in ontology*:

- *Preconditions*: End-effector at initial configuration and gripper is open;
- *Effects*: Classification of objects, decomposition of state and action sequences, assignment of `id_colors` to rings.

R2 *Rule-2 for the* `move_and_grasp` *action*:

- *Preconditions*: Gripper is open, end-effector at standard location and `id_color`;
- *Effects*: The ring is grasped.

R3 *Rule-3 for the* `carry_and_leave` *action*:

- *Preconditions*: Status of execution is `move_and_grasp` and gripper is closed;
- *Effects*: the ring is on the peg.

R4 *Rule-4 for end of task*:

- *Preconditions*: Status of execution at the `carry_and_leave`, gripper is open and `id_color` is the last one;
- *Effects*: End of task.

R5 *Rule-5 for replanning at failure*:

- *Preconditions*: Status of execution at the `carry_and_leave`, gripper is open and tracking status is false;
- *Effects*: the ring is dropped, execute the `move_and_grasp` action.

The inferred action at each time step is communicated to the low-level reasoning module, while the `id_color` information is used to query the vision system.

The perception is handled by an RGB-D camera posed in front of the robot (see Figure 6.2). The camera takes the point cloud and the rectified color images as input. Then, point clouds are filtered to reduce the processing demand. RANSAC [37] is used to extract the parameters of the plane to segment the point clouds with the objects (i.e. pegs and rings). A Euclidean clustering algorithm separates the different objects in the scene.

From a low-level control perspective, each of the three gestures is modeled using a Cartesian DMP (orientation is kept fixed) with parameters $\mathbf{K} = K\mathbf{I}\mathrm{d}_3$ with $K = 1050$, $\mathbf{D} = D\mathbf{I}\mathrm{d}_3$ with $D = 2\sqrt{K} \approx 65$ in (2.10), and $\alpha = 3$ in (2.2). The DMP is learned from a set of (previously segmented) execution of the whole task. The learning set consists of five executions of the whole task, for a total of fifteen replications of both `move_and_grasp` and `carry_and_leave`, and five replications of `move_to_start`. An example of two trajectories for the `carry_and_leave` gesture and the learned DMP are shown in Figure 6.3. We implemented the *roto-dilatation invariant* DMPs presentd in Section 2.6. Obstacle avoidance is implemented using the volumetric static potential method presented in Section 3.2.1. Since the trajectories were learned directly on the same robot that will later execute the learned DMPs, the phase stopping (5.1) is not implemented.

Figure 6.4 shows how the two modules (ontologies and DMPs) are coordinated through sensors, guaranteeing continuous online adaptation.

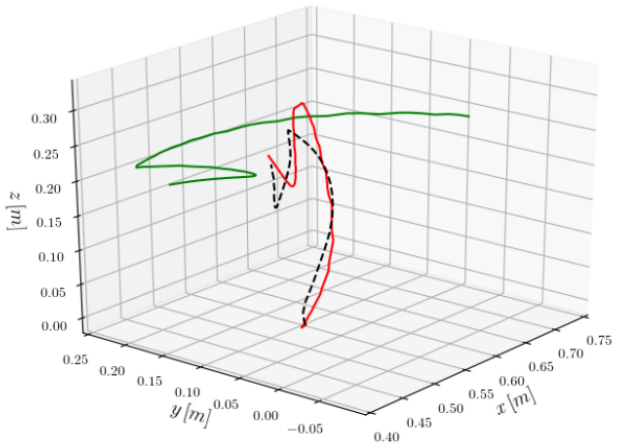


Figure 6.3: Example of DMPs for the automation of the Peg & Ring task with the Panda robot. The green and red solid lines are human demonstrations of the `carry_and_leave` gesture (for the green and red rings respectively). The black dashed line shows the learned DMP adapted to the start and goal positions of the red one.

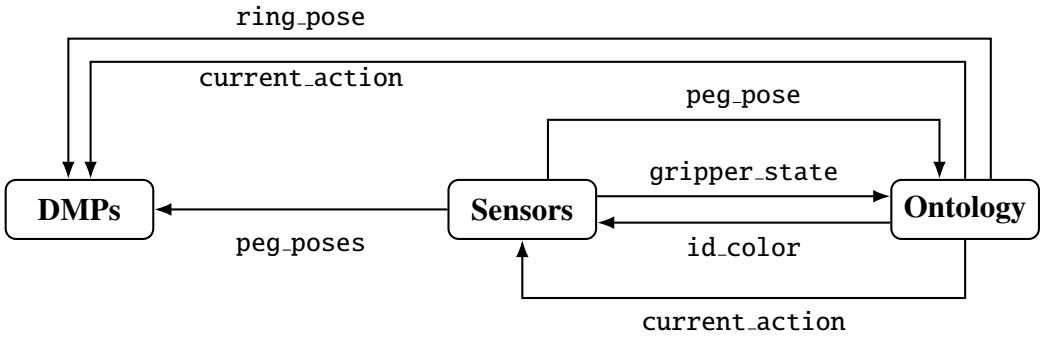


Figure 6.4: Structure of the general framework for the Peg & Ring task. The arrows show the ROS topics used for communication between the main modules.

6.2.1. Results

We tested the framework ten times with different rings' orders. Later, we tested the replanning by purposefully removing the ring from the gripper while carrying.

In the first experiment, the robot started from a pre-defined initial configuration to initiate reasoning. We designed the initial positions of the rings in order to execute three different kinds of motion during `carry_and_leave`: peg on the side of the ring (red), peg in a far diagonal position, and peg in front of the ring (green). In order to prove the repeatability of our framework, the full task was executed ten times with different orderings of the rings, and rings were always placed successfully. In Figure 6.5, the upper row shows the scene as seen from the cam-

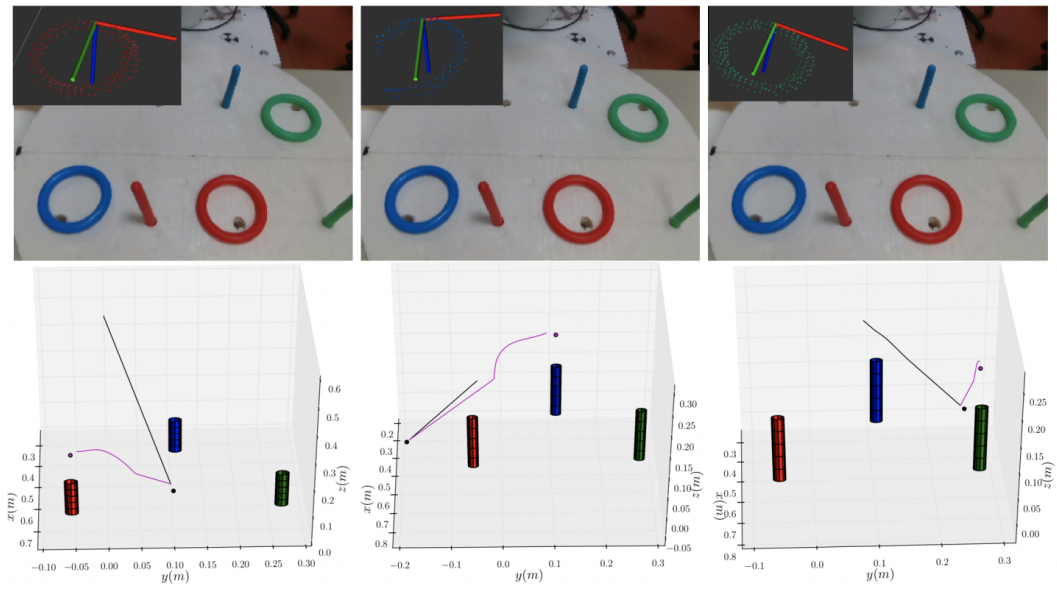


Figure 6.5: Results for one execution of the task. The top row shows the camera point of view and the segmented ring with the target grasping pose. The bottom row shows the trajectories and goal points for the `move_and_grasp` (in black) and `carry_and_leave` (in magenta) gestures.

era, with the segmented point cloud and the target grasping pose computed by the camera for each ring in the upper left corner of the base, retrieved from the RViz simulator. The lower row shows the executed trajectories for `move_and_grasp` and `carry_and_leave` for each ring. The trajectories deviate from the original DMPs because of the presence of the obstacles, especially in `carry_and_leave` for the enlargement of the obstacles.

In the second experiment, the ring was purposefully removed from the gripper of the robot while carrying. The system was able to detect the fall of the ring and trigger `drop`, and `move_and_grasp` again, proving the task-level adaptability to failures. Then, we moved the ring away from its original position while the robot was approaching it, to show the adaptive replanning of DMPs at the motion level. We tested both scenarios, that is ontology replanning and DMPs replanning, to prove repeatability a total of five times, and re-grasping was always successful.

6.2.2. Conclusion

In this Section, we presented a knowledge-based modular framework for automation of surgical procedures involving many actions, integrating task-level ontology reasoning with adaptive motion planning with DMPs. The ontology encodes prior task knowledge, providing interpretability and reliability of the autonomous execution, which are essential for the safety requirement in surgery. Continuously

querying the sensors, failures at the task level and changes in the environment can be handled by exploiting the adaptability of DMPs. The benefits of the framework were tested on a Peg & Ring task.

6.3. Answer Set Programming-Based Framework

In Section 6.2 we presented an improved framework in which we used, as the high-level task reasoner, a module based on *Answer Set Programming* (ASP) instead of ontologies. The logic-based reasoning module generates explainable plans and is able to recover from failure conditions, which are identified and explained by the situation awareness module interfacing to a human supervisor, for enhanced safety. DMPs are used, as the previous framework, to model each gesture necessary for the task execution.

In [44] we have proposed an ontology for the automation of the Peg & Ring task with an industrial manipulator. However, our experiments have evidenced the limits of ontologies, which can only reason on a static representation of the scenario, while real-time knowledge update for reactive planning is computationally inefficient. For this reason, ontologies have been mostly used as support to human monitoring in safety-critical applications, e.g. rehabilitation [31], and industry [106]. On the contrary, non-monotonic programming offers a more flexible framework for logic planning [30], allowing to update incomplete and dynamic knowledge with new observations. Examples in autonomous driving [42], aerospace [6] and industry [35] show the feasibility of non-monotonic reasoning in challenging safety-critical scenarios. While the most popular tool for non-monotonic planning is Prolog [23], we use the more recent framework of Answer Set Programming (ASP) [80] for its better computational efficiency and higher expressivity, allowing, for instance, preference reasoning for optimal planning [12]. Moreover, ASP syntax can be extended with temporal logic [95], reaching the same expressivity as the standard of planning domain languages [26].

Figure 6.6 shows a scheme of the framework. The exchange of information between its modules and the real system (robot + sensors) is shown. The flow of information towards an external human observer is also represented. The human can read the semantic conditions identified by sensors and the plan scheduled for execution at runtime, monitoring the correctness of the overall system. A description of the functions of each module and details about their integration follows. Figure 6.7 show the setup on which we test the proposed framework.

ASP [80] is an explainable Artificial Intelligence (AI) tool to reason on sensory information and on prior knowledge of the task provided by experts. An answer set

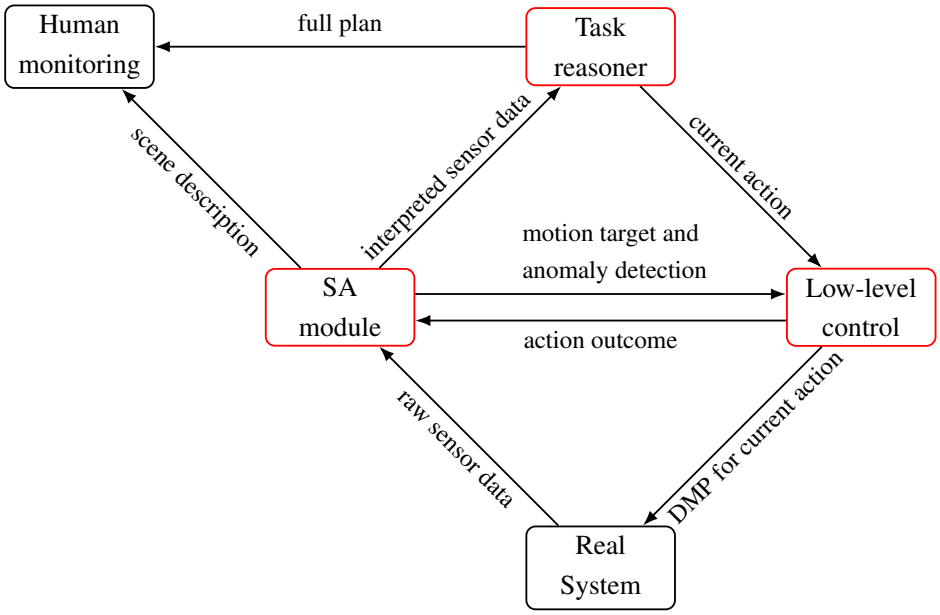


Figure 6.6: Block diagram of ASP-based framework for surgical task automation. Functional modules of the framework are depicted in red, the real system and the external human operator are depicted in black. Arrows show the stream of information between modules.

program defines the entities and the specifications of the task in terms of Boolean variables (called *atoms*) and logical implications (called *rules*). Entities are the objects involved and the actions, while specifications define rules that describe the effects and preconditions of actions, task constraint, and the goal. For the Peg & Ring task with the daVinci, entities are `Arm` (left and right end-effectors), `ring` and `peg` with their `Color` (red, green, blue, yellow and, in the peg case, grey). The task is executed in a bimanual fashion. Therefore, the arm placing (or dropping) the ring on the peg can sometimes be different from the arm grabbing the ring. To be more precise, when the ring is in the same reachability region as the same-colored peg, the arm placing the ring will be the same that grabbed it. On the other hand, when the ring and the same-colored peg are in different reachability regions, the arm placing the ring will differ from the arm grabbing it, and the ring will be passed from one arm to the other in the middle of the setup. Actions with preconditions and effects are defined as follows:

A1 `move(Arm, ring, Color)` is the action used to move towards a colored ring

- *Preconditions*: `reachable(Arm, ring, Color)`, check if it is possible to reach the ring of the given color with the arm;
- *Effect*: `at(Arm, ring, Color)`, moves the arm towards the ring of the given color.

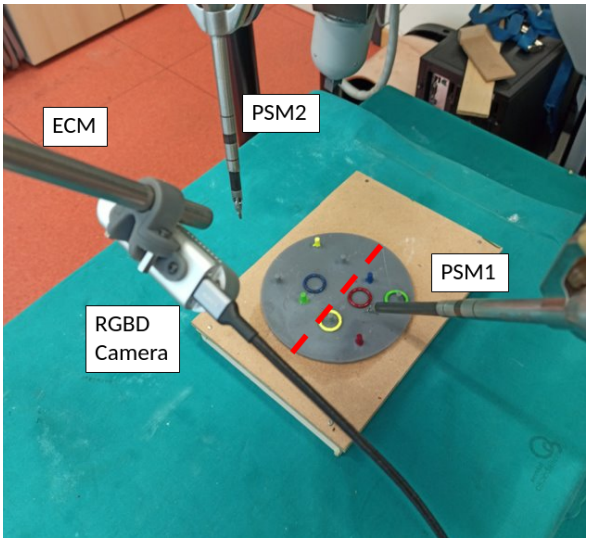


Figure 6.7: The setup for the Peg & Ring task on the daVinci. The dashed red line separates the reachability regions for the two arms.

A2 `move(Arm, peg, Color)` to move the arm towards a colored peg

- *Preconditions*: `reachable(Arm, peg, Color)`, check if it is possible to reach the peg of the given color with the arm;
- *Effect*:`at(Arm, peg, Color)`, moves the arm towards the peg of given color.

A3 `move(Arm, center)` to move the arm towards the transfer point (in which the ring will be transferred from one hand to the other)

- *Preconditions*: `in_hand(Arm, ring, Color)`, check if the ring is hold by the arm's gripper;
- *Effect*: `at(Arm, center)`, moves the arm towards the center of the setup.

A4 `grasp(Arm, ring, Color)` to grasp a ring of a given color

- *Preconditions*: `at(Arm, ring, Color)` check if the end-effector's gripper is at the grasping point selected for the ring of the given color;
- *Effect*: `in_hand(Arm, ring, Color)` closes the gripper, thus grabbing the ring.

A5 `release(Arm)` to open the gripper

- *Preconditions*: `closed_gripper(Arm)` checks if the arm's end-effector is closed;

- *Effect*: `not in_hand(Arm, ring, Color)`, i.e. the ring will no longer be in the arm end-effector.

A6 `extract(Robot, ring, Color)` to remove a colored ring from a peg

- *Preconditions*: `in_hand(Arm, ring, Color)` checks if the arm's end-effector is holding the ring;
- *Effect* `not on(ring, Color1, peg, Color2)`, i.e. the ring of color 1 will no longer be inserted in the peg of color 2.

The atom `reachable` states that a ring or a peg can be reached by an arm, depending on its relative position with respect to the center of the base. Some atoms are defined as *external*, namely they can be set by other programs, in order to allow integration with sensors. External atoms are `reachable`, `on`, `closed_gripper` and `in_hand`. Additionally, the atom `distance(Arm, ring, Color, Value)` is introduced to define the distance between rings and arms. External atoms are recognized by the situation awareness module at runtime. We also define executability constraints to implement user-defined specifications: a ring cannot be grasped by an arm with the closed gripper, a ring that is on a peg cannot be moved before extraction, and a ring cannot be placed on an occupied peg. These constraints emulate standard safety requirements in real surgery. The goal is defined as the constraint that all reachable rings are placed on their pegs, assuming `on(ring, Color, peg, Color)` is an effect of `at(Arm, peg, Color)`, `in_hand(Arm, ring, Color)`, and `release(Arm)`.

Given this task description, the ASP Solving Algorithm 6.1 based on SAT solving [101] is executed. First, *grounding* of the external atoms as received from sensors is performed. This assigns an initial truth value to the corresponding Boolean variables. Then, the solver checks the grounded preconditions, and matches them with the effects of possible actions, incrementing a discrete-time step until the goal is satisfied. We assume a one-step delay between preconditions, actions, and effects. Finally, the sequence of actions that minimizes the time horizon to reach the goal is returned. Notice that the specifications do not determine a fixed temporal sequence of the actions as in standard FSMs, but they only define high-level task-related knowledge that must be taken into account by the ASP solver to produce the fastest feasible plan. By default, we use the *aggregate* construct (`0 { Action: Pre-condition } 1`) from ASP to force the solver to return at most one action per time step. However, later we will relax this constraint to `0 { Action: Pre-condition } 1 :- arm(Arm)`, which allows one action *per robot* at each time step. Therefore, the reasoner will automatically decide whether

Algorithm 6.1 ASP Solving Algorithm

Input: ASP program with specifications, external atoms
Output: `plan`

Ground external atoms
`plan` = [], t = 1, `action` = `null`
while not goal **do**
  if `action` ≠ `null` **then**
    Ground effects of `action`
  end if
  Check pre-conditions for actions at time t
  if some actions are possible **then**
    Select `action` with effect closest to goal
    `plan.append(action(`t`))`
    $t++$
  else
    return Unsatisfiable
  end if
end while
return `plan`

Algorithm 6.2 Vision Algorithm

Input: Point Cloud P_{in} in real time
Output: Poses of rings $\texttt{pose}_{\texttt{ring}}$ and pegs $\texttt{pose}_{\texttt{peg}}$

for t = 1 to ∞ **do**
  Subsample $P_{in}(t)$ to $P_{sub}(t)$
  if t = 1 **then**
    Plane estimation
    return $\texttt{pose}_{\texttt{peg}}$
  else
    for `colorID`= 0 to 3 **do**
      Color Segmentation of $P_{sub}(t)$
      Euclidean clustering
      Ring identification ← RANSAC
      return $\texttt{pose}_{\texttt{ring}}(t)$[`colorID`]
    end for
  end if
end for

the arms should co-operate or act independently, reducing the time to complete the task.

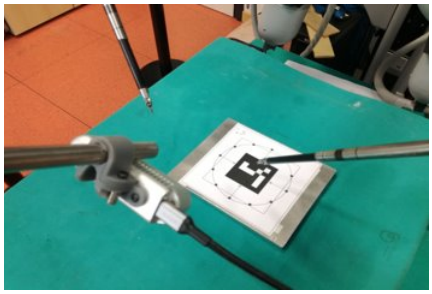
The Situation-Awareness (SA) module is in charge of the semantic interpretation of data from sensors, providing a high-level real-time description of the environment that can be easily read from human supervisors and enhances explainability and safety of the framework. Moreover, the SA module acts as an intermediate layer between task- and motion-level modules, improving the scalability and generality of the framework. The inputs to the SA module are the real-time poses of pegs and centers of rings from an RGBD camera, and the poses of the arms from kinematics. These poses are computed with respect to a common frame *world* for the camera and the robotic arms using hand-eye calibration. During the execution of the task, the Vision Algorithm 6.2 subsamples the point cloud from the scene in order to

guarantee real-time performances. The base and the pegs are assumed to be static during the whole execution, and they are identified only at the beginning of the task. The poses of all rings are retrieved at each time step. The identification of pegs and rings is performed in two steps. First, color segmentation allows identifying same-colored points. Then, Euclidean clustering allows separating the clouds of ring and peg. Finally, RAndom SAmple Consensus (RANSAC) is used to fit a torus shape on both clusters, and the best fitting cluster is identified as the ring, while the other cluster is identified as the peg.

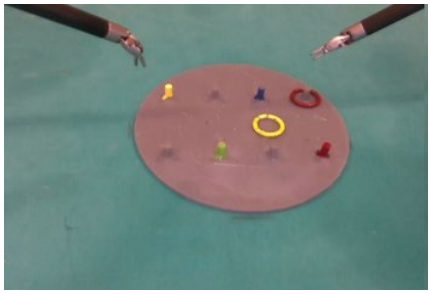
The output of the vision algorithm is used by the SA Algorithm (6.3) to provide external atoms to the task reasoner, check failure conditions and compute targets for the low-level control module. Atoms and failure conditions are identified from geometric information retrieved from the camera. For instance, `in_hand(Arm, ring, Color)` is recognized when the distance between the ring and the arm is below a threshold and the gripper is closed, while the fall of the ring is recognized when the distance between the arm and the ring increases over a threshold during motion. When an action is started by the low-level controller, the specific failure conditions and target pose are computed during the whole execution. In detail, when moving to a ring, the grasping point is selected as the point of the ring cloud most distant from the pegs. Given the position $\mathbf{r}$ of a point on the ring and the set X of positions of pegs, the function to be maximized is $\sqrt{\sum_{\mathbf{p}\in X} \|\mathbf{r}-\mathbf{p}\|_2^2}$. In this way, we guarantee collision-free grasping. Given the grasping point, the orientation is chosen such that the gripper approaches the ring orthogonally to the ring's plane. When moving to a peg, the target orientation is chosen orthogonal to the plane, so that the DMP can automatically recover in case the ring flips. Finally, during transfer between the PSMs[1], the target point for the free arm is chosen as the one opposite to the grasping point of the main arm. In case an anomaly is identified, the low-level control module is notified and the updated external atoms are sent to the task reasoner to compute a new plan.

At the low-level, we used pose DMPs (4.9). Parameters are set to $\mathbf{K} = K\,\mathbf{I}\mathrm{d}_3$, $K = 1050$ and $\mathbf{D} = D\,\mathbf{I}\mathrm{d}, D = 2\sqrt{K} \approx 64.81$ for the Cartesian component (4.9a) and (4.9b), $K_q = 1050$ and $D_q = 2\sqrt{K_q} \approx 64.81$ for the unit quaternion component (4.9c) and (4.9d), and $\alpha = 4$ for the canonical system (4.9e). The error map $\mathbf{e}_0$ in (4.9c) is set to be the vector map. Obstacle avoidance is implemented using the volumetric static potential method (3.14). DMPs are learned from real executions of the task by various users. Three users with different dexterity and familiarity with the setup performed five trials each of the tasks in teleoperation. The initial positions of rings and pegs and the order of the rings (red, green, blue, yellow) are the same for all

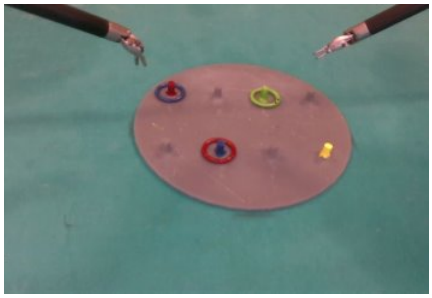
[1]Patient Side Manipulators



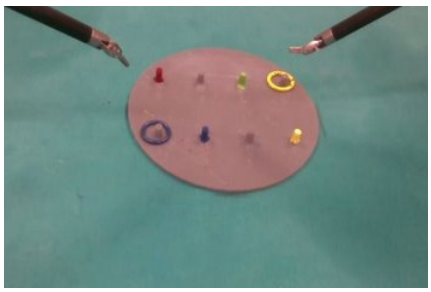
(a) Calibration.



(b) Scenario 1: one ring on the base, one on a gray peg.



(c) Scenario 2: one ring correclty placed and two rings wrongly placed.



(d) Scenario 3: both rings in gray peg. In this case, the two rings can be moved at the same time.

Figure 6.8: Custom calibration board (top-left) and the tested scenarios as seen from the Realsense.

executions. Rings are always transferred between the arms.

In this way, we get 120 executions of the `move(Arm, ring, Color)` gesture (at the beginning and during transfer for each ring), 60 executions of the `move(Arm, peg, Color)` and `move(Arm, center)` gestures. The learning process averages overall human trajectories, without any different weight or bias for executions from more expert users. Figure 6.9 shows the learned Cartesian DMP for the `move(Arm, ring, Color)` gesture as an example.

6.3.1. Results

We tested the framework on the da Vinci Research Kit (DVRK)[2]. The communication between modules of the framework relies on ROS infrastructure [108]. The task reasoning module is implemented using the state-of-the-art grounder and solver Clingo [41], which offers Python APIs for easy integration with ROS, as well as useful tools for incremental time-horizon solving, the definition of external atoms, and optimization statements.

[2] `https://github.com/jhu-dvrk/sawIntuitiveResearchKit`

We use an Intel RealSense d435 camera, which can see depth images from at least 0.105 m. The Point Cloud Library (PCL) is used as the standard tool to process the stream from the camera, offering integration with ROS and useful tools for RANSAC segmentation. We prefer an RGBD camera since the standard surgical endoscope has a smaller baseline between the stereo cameras, reducing the depth range of view and degrading the localization accuracy. Moreover, the adopted hand-eye calibration [116] with a marker on a custom calibration board (Figure 6.8a) allows to reach a state-of-the-art precision of 1.6 mm in pose detection, comparable with the intrinsic accuracy of the daVinci [51]. The RGBD camera is rigidly attached to the end-effector of the endoscopic arm of the daVinci (ECM) with a properly designed adapter.

We validate our framework in three different initial configurations, shown in Figure 6.8b–6.8d.

In scenario 6.8b we show the main different versions of the Peg & Ring task: the red ring is placed on a grey peg and must be extracted, and the yellow ring requires to be transferred. In spite of the calibration accuracy, the small size of the setup and light conditions originate vision errors. In this scenario, the reasoner is also able to replan when the first grasping of the yellow ring fails. Figure 6.10 shows the main steps of the execution, highlighting the semantic scene interpretation and plan generation. In this scenario, we exploit the preference reasoning in ASP to perform optimization and take the closest ring (red) first, using the `distance` variable defined above.

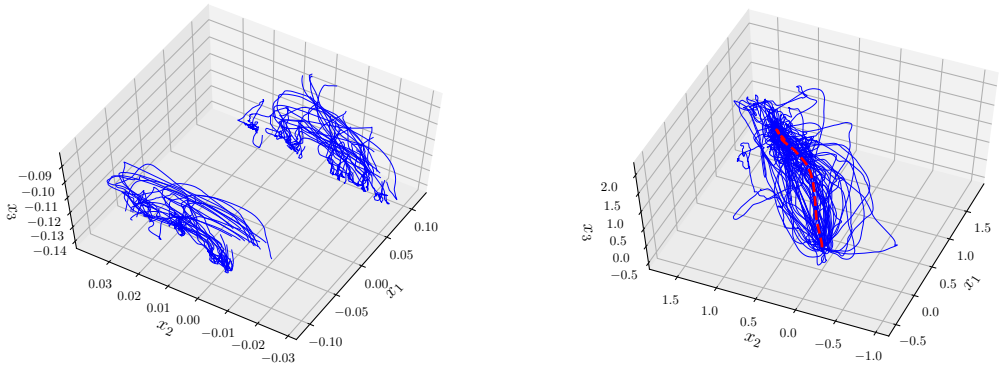
In scenario 6.8c, colored pegs are occupied, hence a ring must be temporarily placed on a grey peg. This operation is not encoded in the ASP program, but the ASP solver autonomously finds this solution as time-optimal from given constraints. Moreover, the SA module identifies the green ring as already placed, hence the reasoner ignores it.

Finally, in scenario 6.8d we test the simultaneous execution of the two arms to complete the task faster, using ASP aggregates as described above.

In Table 6.1 we show the task planning times for the tested scenarios. We also show the planning time for the standard scenario with all rings in the scene requiring transfer. This is the worst-case scenario since more actions are needed to reach the goal. The results prove the real-time capabilities of our task planner (1.78`s` in the worst-case scenario). We notice that, as one would expect, optimization increases the planning time.

6.3.2. Conclusions

In this Section, we presented a framework for the autonomous execution of surgical tasks based on Answer Set Programming. This novel framework is the first funda-



(a) Before rescaling.

(b) After rescaling.

Figure 6.9: The set of Cartesian trajectories for the `move_ring` gesture, for both PSMs, both before and after the rescaling via rotodilatation. The learned DMP in Figure 6.9b is shown in red dashed line.

Scenario	**Planning time [s]**
Scenario 6.8b (optimization)	0.108 (0.424)
Scenario 6.8c	0.133
Scenario 6.8d	0.113
Complete (optimization)	1.780 (8.636)

Table 6.1: Planning time for the ASP planner in the tested scenarios and in the worst-case scenario (complete) with all four rings requiring transfer.

mental step to address the problems of failure recovery, explainable plan generation, and situation awareness in the surgical scenario, which are required features for the acceptability of an autonomous surgical system. We focused on a more complex version of the Peg & ring task, a standard in the training program for surgeons, with the daVinci surgical robot. An ASP-based task reasoner is able to quickly coordinate the minimal set of actions in non-standard task conditions, integrating with semantic sensing of the scene, and respecting task-specific constraints. Motion trajectories are learned from teleoperated executions by users with different expertise, using the DMP framework to replicate human dexterity.

6.4. Conclusions

In this Chapter, we presented two frameworks to automate a surgical-related robotic task. We first presented the Peg & Ring task, which is widely used to train and evaluate surgeons. We precisely described the task and the possible variations that

can make it more challenging (ring placements, number of end-effectors involved, ...).

Both frameworks comprehend a high-level task reasoner, a low-level control module, and a sensing/situation awareness part. The main difference between the two frameworks is the task reasoner: in the first one, we used ontologies, while in the second we used Answer Set Programming. Both frameworks use DMPs as the low-level controller, allowing them to learn gestures from real task executions and use the learned behaviors to generate the trajectories necessary to accomplish the task.

One drawback of the implementation of both frameworks is the use of an RGB-D camera instead of a surgical endoscope. We used the RGB-D camera to reach sufficient precision, which cannot be achieved with the endoscope. Hence, at the moment, the application to real surgery is still limited. However, given the modularity of the frameworks, it would be easy to generalize them to different sensory sources.

Both frameworks are able to autonomously execute the task. On the ontology-based framework, we tested the capability of the system to recognize when the ring falls from the gripper. In this case, it was able to identify the failure condition and to replan the execution. Moreover, thanks to the convergence property of DMPs, it was able to reach the ring even when we moved it away, by continuously updating the DMPs' goal (i.e. the grasping position).

In the second framework, we used ASP as the high-level task reasoner instead of ontologies. This framework maintains many desirable properties of ontologies, such as the generation of explainable plans and the ability to recover from failures. Moreover, ASP can achieve a real-time knowledge update (on the other hand, ontologies work on a static representation of the scenario) resulting in better computational efficiency and higher expressivity. In particular, being able to adapt the knowledge in real-time, makes ASP a preferable choice in Minimally Invasive Surgery, where unexpected events may happen.

The ontology-based framework was tested on a simple version of the task: single-handed, with the pegs placed on the base, and using an industrial manipulator. On the other hand, the ASP-framework was tested on the daVinci surgical robot, performing a more complicated version of the Peg & Ring task. In more detail, the task was executed in a bimanual fashion. This introduces the additional gesture in which the ring has to be transferred from one end-effector to the other. Moreover, we tested the ASP-based framework in different scenarios of increasing difficulty. In the most complex case, some of the rings are initially placed on the wrong peg, and the system must first 'free' the peg. This scenario is more complex than the actual setup used to train and evaluate surgeons.

Even tho the Peg & Ring task may seem a simple task, its importance in surgical applications is well known. Indeed, it presents many challenges of real surgery, such as grasping and transferring, and it is widely used to train and test surgeons. Thus, automation of the Peg & Ring task is a crucial first step towards the automation of a more complex surgical task.
The fact that both frameworks use DMPs as the low-level controller, together with the fact that the Peg & Ring task is one of the most widely used in surgeons' training confirm that DMPs are capable of mimicking surgical gestures.

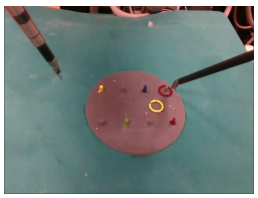
In the future, we aim to implement a more complex ASP-based framework to automate simple surgical tasks such as suturing and knot-tying.

Algorithm 6.3 SA Algorithm

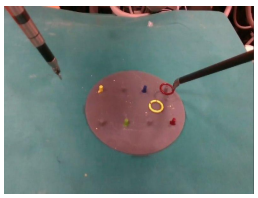
Input: Action, $\texttt{pose}_{\texttt{ring}}$, $\texttt{pose}_{\texttt{peg}}$

Output: Failure message, `target_pose`, `external_atoms`

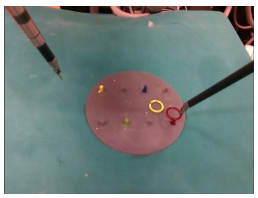
```
failure = True
if failure then
    external_atoms = compute_externals(pose_ring, pose_peg)
    return external_atoms
    failure = False
else if  thenexecuting_action
    while Action not ended do
        target_pose = compute_target(pose_peg)
        return target_pose
        if Action = move_ring then
            if pose_ring[colorID] is not retrieved then
                failure = True
                return failure
            end if
        else if Action = move_peg then
            if ring fallen or peg occupied then
                failure = True
                return failure
            end if
        else if Action = move_center then
            if ring fallen then
                failure = True
                return failure
            end if
        end if
    end while
end if
```



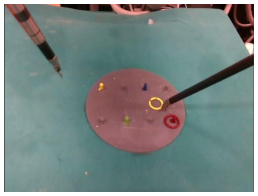
(a)
`move(psm1,ring,red,1)`



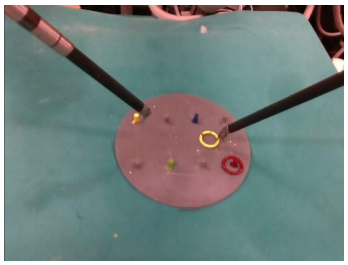
(b)
`extr(psm1,ring,red,3)`



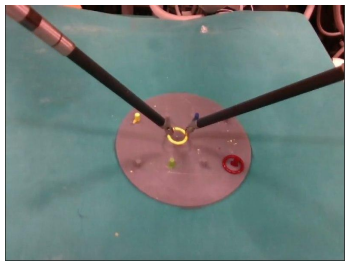
(c)
`move(psm1,peg,red,4)`



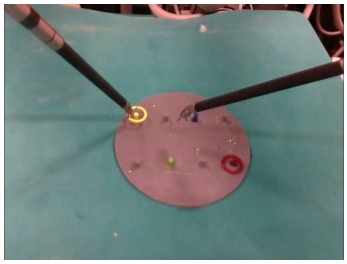
(d)
`move(psm1,ring,yel,6)`



(e) `move(psm1,ring,yel,9)`



(f) `move(psm1,center,11)`



(g) `move(psm2,peg,yel,14)`

Figure 6.10: Main actions of example execution in scenario 6.8b (`grasp`, `release` actions are omitted for simplicity). The initial plan is generated in a) after the grounding of the information from the SA module: `reachable(psm1,ring,red)`, `reachable(psm1,ring,yellow)`, `reachable(psm1,peg,red)`, `reachable(psm1,peg,blue)`, `reachable(psm2,peg,yellow)`, `reachable(psm2,peg,green)`, `on(ring,red,peg,grey)`. e) shows the replanning of action `move(psm1,ring,yellow)` when `ring_fallen` is received from SA module.

III

Auto Regressive Hidden Markov Model and Extensions

CHAPTER 7

AUTO REGRESSIVE HIDDEN MARKOV MODEL

In the first part of this thesis, we discussed the learning of surgical gestures using Dynamic Movement Primitives. We proved, in Chapter 6, that DMPs can be effectively used as the low-level controller when automating surgical-related tasks.

However, the problem of how to extract the gestures to learn the DMPs from surgical trials remains. Manual segmentation of complete surgical tasks into sub-tasks is a long and tedious process. Moreover, since gestures are executed using muscular memory and are of short duration, small errors in the manual segmentation are practically unavoidable.

For this reason, unsupervised segmentation is a well-studied topic in surgical robotics. Some approaches start with an over-segmentation of the data stream, and proceed by removing false positive cuts (thus merging two segments) relying on geometrical or statistical properties of the segments [1, 82, 97, 28, 83, 70]. Using the opposite strategy, some approaches start with a unique segment and perform further partitions, still based on geometric or statistical properties [123]. Other approaches look for movement repetitions within the time series to use as the basis for the segmentation [21]. Finally, semi-unsupervised approaches assume that a library of primitives is available, and uses it to perform the segmentation [94].

In this thesis, we propose a model-based approach. That is, we assume that there exists a common mathematical model underlying all the time series in our dataset, and we aim at extracting this model. Once the model is known, it can be used to extract the 'most probable' segmentation that generated the time series.

One of the most used models in this context is the Hidden Markov Model (HMMs) [17, 131, 128]. This type of model assumes that there exists a finite set of *modes* that cannot be observed (and are thus called *hidden* or *latent*) and that each mode

emits an *observation*.
An extension of HMMs is the *Auto-Regressive HMM* (AR-HMMs) [67, 33, 71, 48]. In this case, each latent mode encodes a linear model which is used to evolve the observed state.

In Section 7.1 we recall the theory of classical HMMs, and then, in Section 7.2, we will present AR-HMMs as a natural extension to HMMs.
Then, in the next Chapter, we will present some extensions of AR-HMM that allow making the model more generalizable.

In Appendix E we present a brief summary of the topics of probability theory needed to develop the HMM model.

In this Chapter, we will present in detail the development of the *Expectation-Maximization* (EM) *algorithm*, used to infer the model's parameters from data. Even though these formulae are well known, proving them from scratch will help in understanding in detail the latent variable models, and will provide a solid starting point to develop the EM algorithm for the generalization of the AR-HMM model that we present in Chapter 8.

7.1. Hidden Markov Model

Before introducing the Auto-Regressive Hidden Markov Model (AR-HMM), we present the simpler (but still widely used) Hidden Markov Model (HMM). Later, in Section 7.2 we will present the AR-HMM as an extension.

HMM is a probabilistic model in which the latent mode at each time t is an element of a finite set, while the observed state at each time may be both discrete or continuous. Three probability distributions are use to define an HMM: an *initial mode probability*, a *transition probability* between hidden modes, and an *emission probability*.

At first, we will precisely define the model. Later, in Section 7.1.1 we present the *Expectation-Maximization* algorithm, used to "learn" the model parameters from a single time series. In Section 7.44a we will extend the EM agorithm to the case of multiple observations. Then, In Section 7.1.3 we will discuss a strategy to initialize the model's parameters before executing the EM algorithm. Finally, in Section 7.1.4, we present the *Viterbi algorithm*, which is used to extract the most likely sequence of hidden modes given a time series.

Definition 7.1 (Hidden Markov Model)**.** An Hidden Markov Model (HMM) is a system $\mathcal{H} = \{\mathcal{S}, \mathcal{Y}, \Theta\}$ where [58]:

- $\mathcal{S} = \{1, 2, \dots, S\}$ is the set of (hidden) *modes*. The mode at time $t \in \{0, 1, \dots, T\}$ is denoted by z_t.

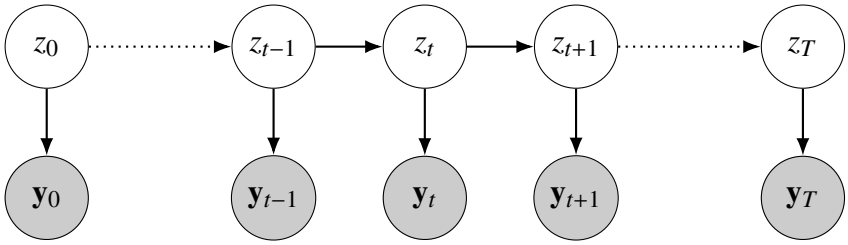


Figure 7.1: Graphical model representation of an Hidden Markov Model.

- Vector $\boldsymbol{\varpi} = [\varpi_i]_{i=1,2,\dots,S} \in \Theta$ defines the probability of the first mode:

  $$\Pr(z_0 = i) = \varpi_i, \quad i \in \mathcal{S}.$$

  It satisfies $\varpi_i \in [0,1], \forall i = 1,2,\dots,S$ and $\sum_{i=1}^{S} \varpi_i = 1$.

- Matrix $\mathbf{T} = [T|_{ij}]_{i=1,2,\dots,S}^{j=1,2,\dots,S} \in \Theta$ defines the transition probabilities between modes

  $$\Pr(z_{t+1} = j | z_t = i) = \mathbf{T}|_{ij}, \quad i,j \in \mathcal{S}.$$

  It satisfies $T|_{ij} \in [0,1], \forall i,j = 1,2,\dots,S$ and $\sum_{j=1}^{S} \mathbf{T}|_{ij} = 1, \forall i = 1,2,\dots,S$.

- $\mathcal{Y}$ is the set of possible observations. The observation at time $t \in \{0,1,\dots,T\}$ is denoted by $\mathbf{y}_t$. Set $\mathcal{Y}$ has usually two definitions, depending on the application study. It can be a finite set, $\mathcal{Y} = \{1,2,\dots,Y\}$, and the emission probability is given by the so-called *emission probability matrix* $\boldsymbol{\Omega} = [\boldsymbol{\Omega}|_{i\ell}]_{i=1,2,\dots,S}^{\ell=1,2,\dots,Y} \in \Theta$:

  $$\Pr(\mathbf{y}_t = \ell | z_t = i) = \boldsymbol{\Omega}|_{i\ell}, \qquad i \in \mathcal{S}, \quad \ell \in \mathcal{Y}.$$

  It satisfies $\boldsymbol{\Omega}|_{i\ell} \in [0,1], \forall i = 1,2,\dots,S, \ell = 1,2,\dots,Y$, and $\sum_{\ell=1}^{Y} \boldsymbol{\Omega}|_{i\ell} = 1, \forall i = 1,2,\dots,S$.
  Alternatively, set $\mathcal{Y}$ can be isomorph to $\mathbb{R}^d$, $d \in \mathbb{N}$. In this case, the emission probability is usually set to be a normal distribution with mean $\boldsymbol{\mu}_s$ and covariance matrix $\boldsymbol{\Sigma}_s$ so that

  $$p(\mathbf{y}_t | z_t) \sim \mathcal{N}(\boldsymbol{\mu}_{z_t} | \boldsymbol{\Sigma}_{z_t}).$$

The set Θ contains all the parameters of the model. In the case in which $\mathcal{Y}$ is finite, it is defined as

$$\Theta = \{\boldsymbol{\varpi}, \mathbf{T}, \boldsymbol{\Omega}\},$$

while for continuous emission space, the set Θ is defined as

$$\Theta = \{\boldsymbol{\varpi}, \mathbf{T}, \{\boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s\}_{s \in \mathcal{S}}\}.$$

A picture of HMM as graphical model is given in Figure 7.1 (For further details about graphical models, see Appendix E). We remark that the graphical model does not depend on the definition of the emission space $\mathcal{Y}$ since the graphical model shows only the conditional probabilities between the variables of the model and not their formulations.

7.1.1. Expectation-Maximization Algorithm

We now develop the *Expectation-Maximization* (EM) algorithm, introduced in Appendix E.3, for HMM. The EM algorithm is used to compute the set of parameters Θ that maximizes the likelihood of the observed data $p(\mathbf{Y}|\Theta)$.

By denoting the sequence of observations as $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_T)$, and the sequence of hidden modes as $\mathbf{z} = (z_0, z_1, \dots, z_T)$, we define the *complete data likelihood* of a HMM as

$$p(\mathbf{Y}, \mathbf{z}) = p(z_0) \prod_{t=0}^{T-1} p(z_{t+1}|z_t) \prod_{t=0}^{T} p(\mathbf{y}_t|z_t). \tag{7.1}$$

By taking the logarithm of (7.1) we obtain the *complete-data log-likelihood*

$$\log p(\mathbf{Y}, \mathbf{z}) = \log p(z_0) + \sum_{t=0}^{T-1} \log p(z_{t+1}|z_t) + \sum_{t=0}^{T} \log p(\mathbf{y}_t|z_t). \tag{7.2}$$

During the learning process, we aim at maximizing the function

$$Q(\Theta, \Theta^{\text{old}}) = \sum_{\mathbf{z} \in \mathcal{S}^{T+1}} p(\mathbf{z}|\mathbf{Y}, \Theta^{\text{old}}) \log p(\mathbf{Y}, \mathbf{z}|\Theta),$$

where $\mathcal{S}^{T+1}$ denotes the set of all possible sequences $\mathbf{z} = (z_0, z_1, \dots, z_T) \in \mathcal{S}^{T+1}$. By expanding term $\log p(\mathbf{Y}, \mathbf{z}|\Theta)$ using (7.2) in function Q we obtain (by suppressing the dependence on Θ for notation simplicity)

$$Q(\Theta, \Theta^{\text{old}}) = \sum_{\mathbf{z} \in \mathcal{S}^{T+1}} p(\mathbf{z}|\mathbf{Y}, \Theta^{\text{old}}) \left(\log p(z_0) + \sum_{t=0}^{T-1} \log p(z_{t+1}|z_t) + \sum_{t=0}^{T} \log p(\mathbf{y}_t|z_t) \right). \tag{7.3}$$

Now, we can write the sum over $\mathcal{S}^{T+1}$ as $T+1$ sums over $\mathcal{S}$. Thus, (7.3) reads

$$Q(\Theta, \Theta^{\text{old}}) = \sum_{z_0 \in \mathcal{S}} \cdots \sum_{z_t \in \mathcal{S}} \cdots \sum_{z_T \in \mathcal{S}} p(z_0.z_1, \dots, z_T|\mathbf{Y}, \Theta^{\text{old}}) \left(\log p(z_0) + \sum_{t=0}^{T-1} \log p(z_{t+1}|z_t) + \sum_{t=0}^{T} \log p(\mathbf{y}_t|z_t) \right).$$

Next, we can rewtite the nested summations as follows

$$\begin{aligned} Q(\Theta,\Theta^{\text{old}}) = &\sum_{z_0\in\mathcal{S}}\cdots\sum_{z_t\in\mathcal{S}}\cdots\sum_{z_T\in\mathcal{S}} p(z_0,z_1,\ldots,z_T|\mathbf{Y},\Theta^{\text{old}})\log p(z_0)+\\ &\sum_{t=0}^{T-1}\sum_{z_0\in\mathcal{S}}\cdots\sum_{z_t\in\mathcal{S}}\cdots\sum_{z_T\in\mathcal{S}} p(z_0,z_1,\ldots,z_T|\mathbf{Y},\Theta^{\text{old}})\log p(z_{t+1}|z_t)+\\ &\sum_{t=0}^{T}\sum_{z_0\in\mathcal{S}}\cdots\sum_{z_t\in\mathcal{S}}\cdots\sum_{z_T\in\mathcal{S}} p(z_0,z_1,\ldots,z_T|\mathbf{Y},\Theta^{\text{old}})\log p(\mathbf{y}_t|z_t). \end{aligned} \tag{7.4}$$

At this point, let us remark that, by marginalization of probability densities, we have that the following identities hold

$$\sum_{z_0\in\mathcal{S}}\cdots\sum_{z_t\in\mathcal{S}}\cdots\sum_{z_T\in\mathcal{S}} p(z_0,z_1,\ldots,z_T)\,f(z_t) = \sum_{z_t\in\mathcal{S}} p(z_t)\,f(z_t), \tag{7.5a}$$

$$\sum_{z_0\in\mathcal{S}}\cdots\sum_{z_t\in\mathcal{S}}\cdots\sum_{z_T\in\mathcal{S}} p(z_0,z_1,\ldots,z_T)\,g(z_t,z_{t'}) = \sum_{z_t\in\mathcal{S}}\sum_{z_{t'}\in\mathcal{S}} p(z_t,z_{t'})\,g(z_t,z_{t'}), \tag{7.5b}$$

for any smooth functions $f:\mathbb{R}\to\mathbb{R}$ and $g:\mathbb{R}^2\to\mathbb{R}$ and any indeces $t,t'\in\{0,1,\ldots,T\}$.

By using marginalization properties (7.5), we have that (7.4) can be simplified. Thus, we get that maximization of $Q(\Theta,\Theta^{\text{old}})$ over Θ is equivalent to the maximization of

$$\begin{aligned} \widetilde{Q}(\Theta,\Theta^{\text{old}}) = &\sum_{z_0\in\mathcal{S}}\log p(z_0|\Theta)\,p(z_0|\mathbf{Y},\Theta^{\text{old}})+\\ &\sum_{t=0}^{T-1}\sum_{z_t\in\mathcal{S}}\sum_{z_{t+1}\in\mathcal{S}}\log p(z_{t+1}|z_t,\Theta)\,p(z_t,z_{t+1}|\mathbf{Y},\Theta^{\text{old}})+\\ &\sum_{t=0}^{T}\sum_{z_t\in\mathcal{S}}\log p(\mathbf{y}_t|z_t,\Theta)\,p(z_t|\mathbf{Y},\Theta^{\text{old}}). \end{aligned} \tag{7.6}$$

At this point, the EM algorithm can be divided into its components: the *Expectation step* and the *Maximization step*. In the former, the quantities depending on the old set of parameters Θ^{old} are computed. The latter computes the new set of parameters Θ that maximizes $\widetilde{Q}(\Theta,\Theta^{\text{old}})$.

Expectation Step

In the E-step we aim at computing the so-called *marginals*

$$\gamma(z_t) \overset{\text{def}}{=} p(z_t|\mathbf{Y},\Theta^{\text{old}}),\quad t=0,1,\ldots,T, \tag{7.7a}$$

$$\xi(z_t,z_{t+1}) \overset{\text{def}}{=} p(z_t,z_{t+1}|\mathbf{Y},\Theta^{\text{old}}),\quad t=0,1,\ldots,T-1. \tag{7.7b}$$

We remark that, by marginalization, we get

$$\begin{aligned}\sum_{z_{t+1}\in\mathcal{S}} \xi(z_t, z_{t+1}) &= \sum_{z_{t+1}} p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) \\ &= p(z_t|\mathbf{Y}, \Theta^{\text{old}}),\end{aligned}$$

thus getting the identity

$$\sum_{z_{t+1}\in\mathcal{S}} \xi(z_t, z_{t+1}) = \gamma(z_t). \tag{7.8}$$

In the following, we will decompose marginals γ and ξ into the so-called *forward variables* α and *backward variables* β, which can be efficiently computed recursively.

Let us begin by considering marginal γ. By product rule of probability (E.2), we get

$$\begin{aligned}\gamma(z_t) &= p(z_t|\mathbf{Y}, \Theta^{\text{old}}) \\ &= \frac{p(\mathbf{Y}, z_t|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{p(\mathbf{y}_0, \dots, \mathbf{y}_t, \mathbf{y}_{t+1}, \dots, \mathbf{y}_T, z_t|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})}.\end{aligned}$$

By applying again the product rule to the numerator, and by taking advantage of the conditional independence property $\mathbf{y}_{t+1}, \dots, \mathbf{y}_T \perp\!\!\!\perp \mathbf{y}_0, \dots, \mathbf{y}_t | z_t, \Theta^{\text{old}}$ we get

$$\begin{aligned}\gamma(z_t) &= \frac{p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, \Theta^{\text{old}})\, p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{\alpha(z_t)\beta(z_t)}{p(\mathbf{Y}|\Theta^{\text{old}})},\end{aligned} \tag{7.9}$$

where the two factors at numerator are defined as

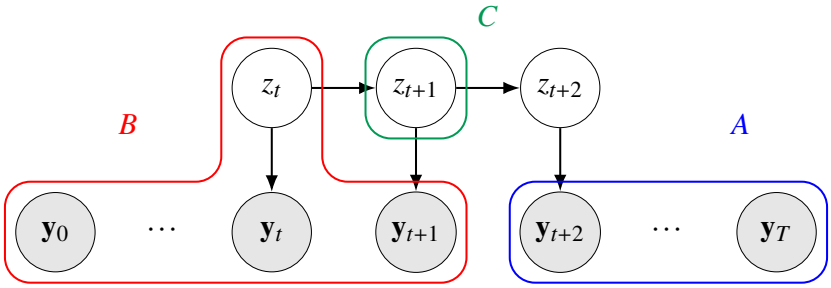
$$\alpha(z_t) \stackrel{\text{def}}{=} p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t|\Theta^{\text{old}}), \tag{7.10a}$$

$$\beta(z_t) \stackrel{\text{def}}{=} p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|z_t, \Theta^{\text{old}}), \tag{7.10b}$$

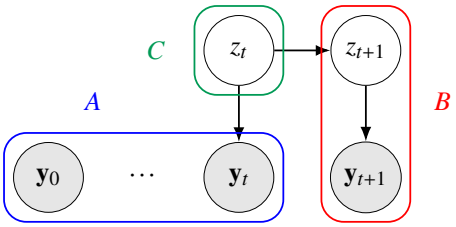
where α is called the *forward variable*, and β is called the *backward variable*.
Next, let us show how the forward and backward variables can be used to express the marginal ξ.

Proposition 7.1. *Marginal ξ can be written in terms of forward and backward variables* (7.10) *as*

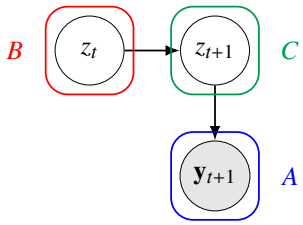
$$\xi(z_t, z_{t+1}) = \frac{\beta(z_{t+1})\, p(\mathbf{y}_{t+1}|z_{t+1}, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, \alpha(z_t)}{p(\mathbf{Y}|\Theta^{\text{old}})}. \tag{7.11}$$



(a) Conditional independence property (7.14): the arrows in the path from any element in the set A to any element of B meet only head-to-head or tail-to-tail in C.



(b) Conditional independence property (7.16): the arrows in the path from any element in the set A to any element of B meet only tail-to-tail in C.



(c) Conditional independence property (7.17): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.

Figure 7.2: Graphical proof of conditional independence properties of proof of Proposition 7.1. All the diagrams show conditional independence $A \perp\!\!\!\perp B|C$. For a recall on graphical model and conditional independence, see Appendix E.

Proof. By Bayes theorem (E.3) we have

$$\begin{aligned}\xi(z_t, z_{t+1}) &= p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) \\ &= \frac{p(\mathbf{Y}|z_t, z_{t+1}, \Theta^{\text{old}})\, p(z_t, z_{t+1}|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})}\end{aligned} \tag{7.12}$$

The first factor at numerator can be written, by using product rule (E.2) as

$$\begin{aligned}p(\mathbf{Y}|z_t, z_{t+1}, \Theta^{\text{old}}) &= p(\mathbf{y}_0, \ldots, \mathbf{y}_{t+1}, \mathbf{y}_{t+2}, \ldots, \mathbf{y}_T|z_t, z_{t+1}, \Theta^{\text{old}}) \\ &= p(\mathbf{y}_{t+2}, \ldots, \mathbf{y}_T|z_t, z_{t+1}, \mathbf{y}_0, \ldots, \mathbf{y}_{t+1}, \Theta^{\text{old}}) p(\mathbf{y}_0, \ldots, \mathbf{y}_{t+1}|z_t, z_{t+1}\Theta^{\text{old}}).\end{aligned} \tag{7.13}$$

By conditional independence property (see Figure 7.2a)

$$\mathbf{y}_{t+2}, \ldots, \mathbf{y}_T \perp\!\!\!\perp \mathbf{y}_0, \ldots, \mathbf{y}_{t+1}, z_t|z_{t+1}, \Theta^{\text{old}}, \tag{7.14}$$

we have that the first factor of (7.13) can be reduced to

$$p(\mathbf{y}_{t+2}, \ldots, \mathbf{y}_T|z_{t+1}, \Theta^{\text{old}}) = \beta(z_{t+1}).$$

The second factor of (7.13) can be written as, by using product rule (E.2), as

$$p(\mathbf{y}_0, \ldots, \mathbf{y}_{t+1} | z_t, z_{t+1} \Theta^{\text{old}}) = p(\mathbf{y}_0, \ldots, \mathbf{y}_t | z_t, z_{t+1}, \mathbf{y}_{t+1}, \Theta^{\text{old}}) p(\mathbf{y}_{t+1} | z_t, z_{t+1}, \Theta^{\text{old}}). \tag{7.15}$$

By conditional independence properties (see Figure 7.2b)

$$\mathbf{y}_0, \ldots, \mathbf{y}_t \perp\!\!\!\perp z_{t+1}, \mathbf{y}_{t+1} | z_t, \Theta^{\text{old}}, \tag{7.16}$$

and (see Figure 7.2c)

$$\mathbf{y}_{t+1} \perp\!\!\!\perp z_t | z_{t+1}, \Theta^{\text{old}}, \tag{7.17}$$

we can write the term (7.15) as

$$p(\mathbf{y}_0, \ldots, \mathbf{y}_{t+1} | z_t, z_{t+1} \Theta^{\text{old}}) = p(\mathbf{y}_0, \ldots, \mathbf{y}_t | z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1} | z_{t+1}, \Theta^{\text{old}}).$$

Thus, (7.13) reads

$$p(\mathbf{Y} | z_t, z_{t+1}, \Theta^{\text{old}}) = \beta(z_{t+1})\, p(\mathbf{y}_0, \ldots, \mathbf{y}_t | z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1} | z_{t+1}, \Theta^{\text{old}}).$$

The second factor in (7.12) can be written as, by using product rule,

$$p(z_t, z_{t+1} | \Theta^{\text{old}}) = p(z_{t+1} | z_t, \Theta^{\text{old}})\, p(z_t | \Theta^{\text{old}}).$$

Combining these decompositions, we obtain that marginal ξ can be written as

$$\xi(z_t, z_{t+1}) = \frac{\beta(z_{t+1}) p(\mathbf{y}_0, \ldots, \mathbf{y}_t | z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1} | z_{t+1})\, p(z_{t+1} | z_t, \Theta^{\text{old}})\, p(z_t | \Theta^{\text{old}})}{p(\mathbf{Y} | \Theta^{\text{old}})}.$$

At the numerator, we can use the product rule of probability to write

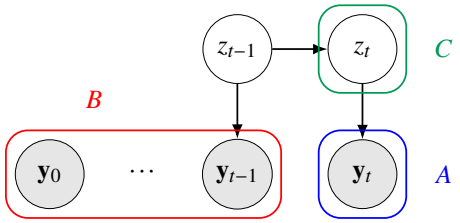
$$\begin{aligned} p(\mathbf{y}_0, \ldots, \mathbf{y}_t | z_t, \Theta^{\text{old}})\, p(z_t | \Theta^{\text{old}}) &= p(\mathbf{y}_0, \ldots, \mathbf{y}_t, z_t | \Theta^{\text{old}}) \\ &= \alpha(z_t). \end{aligned}$$

Thus, we have that marginal ξ can be written in terms of forward and backward variable as

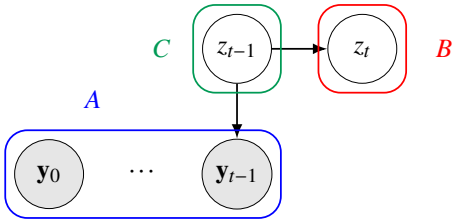
$$\xi(z_t, z_{t+1}) = \frac{\beta(z_{t+1})\, p(\mathbf{y}_{t+1} | z_{t+1})\, p(z_{t+1} | z_t, \Theta^{\text{old}})\, \alpha(z_t)}{p(\mathbf{Y} | \Theta^{\text{old}})}.$$

□

We now seek for recursion formula to make the computation of the forward and backward variables, $\alpha(z_t)$ and $\beta(z_t)$ efficient.



(a) Conditional independence property (7.20): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.



(b) Conditional independence property (7.22): the arrows in the path from any element in the set A to any element of B meet only tail-to-tail in C.

Figure 7.3: Graphical proof of conditional independence properties of proof of Proposition 7.2. All the diagrams show conditional independence $A \perp\!\!\!\perp B|C$.

Proposition 7.2. *Forward variable* $\alpha(z_t)$ *defined in* (7.10a) *can be recursively computed as*

$$\alpha(z_t) = p(\mathbf{y}_t|z_t, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \alpha(z_{t-1})\, p(z_t|z_{t-1}). \tag{7.18}$$

Proof. By applying the product rule of probability (E.2) to the definition of $\alpha(z_t)$,

$$\begin{aligned} \alpha(z_t) &= p(\mathbf{y}_0, \ldots, \mathbf{y}_t, z_t|\Theta^{\text{old}}) \\ &= p(\mathbf{y}_t|\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t, \Theta^{\text{old}})\, p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t|\Theta^{\text{old}}). \end{aligned} \tag{7.19}$$

The first factor can be simplified by taking advantage of the conditional independence property (see Figure 7.3a)

$$\mathbf{y}_t \perp\!\!\!\perp \mathbf{y}_0, \ldots, \mathbf{y}_{t-1}|z_t, \Theta^{\text{old}}, \tag{7.20}$$

thus getting

$$p(\mathbf{y}_t|\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t, \Theta^{\text{old}}) = p(\mathbf{y}_t|z_t, \Theta^{\text{old}}).$$

The second factor in (7.19) can be written using the sum rule of probability (E.1a) marginalizing over z_{t-1} as

$$p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t|\Theta^{\text{old}}) = \sum_{z_{t-1} \in \mathcal{S}} p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t, z_{t-1}|\Theta^{\text{old}}).$$

By using the product rule of probability to the argument inside the sum we can write it as

$$p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t, z_{t-1}|\Theta^{\text{old}}) = p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}|z_t, z_{t-1}, \Theta^{\text{old}})\, p(z_{t-1}, z_t|\Theta^{\text{old}}). \tag{7.21}$$

By applying the conditional independence property (see Figure 7.3b)

$$\mathbf{y}_0, \ldots, \mathbf{y}_{t-1} \perp\!\!\!\perp z_t | z_{t-1}, \Theta^{\text{old}} \tag{7.22}$$

to the first factor, and the product rule to the second, we get that (7.21) can be written as

$$\begin{aligned} p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_t, z_{t-1} | \Theta^{\text{old}}) &= p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1} | z_{t-1}, \Theta^{\text{old}})\, p(z_t | z_{t-1}, \Theta^{\text{old}}) p(z_{t-1} | \Theta^{\text{old}}) \\ &= p(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, z_{t-1}, \Theta^{\text{old}})\, p(z_t | z_{t-1}, \Theta^{\text{old}}) \\ &= \alpha(z_{t-1})\, p(z_t | z_{t-1}, \Theta^{\text{old}}). \end{aligned}$$

Thus, we get the recursive formula for the forward variable:

$$\alpha(z_t) = p(\mathbf{y}_t | z_t, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \alpha(z_{t-1})\, p(z_t | z_{t-1}).$$

□

The basis of recursion is

$$\begin{aligned} \alpha(z_0) &= p(\mathbf{y}_0, z_0 | \Theta^{\text{old}}) \\ &= p(\mathbf{y}_0 | z_0, \Theta^{\text{old}})\, p(z_0 | \Theta^{\text{old}}). \end{aligned}$$

Let us now develop the recursive formula for the backward variable $\beta(z_t)$.

Proposition 7.3. *Backward variable $\beta(z_t)$ defined in* (7.10b) *can be recursively computed as*

$$\beta(z_t) = \sum_{z_{t+1} \in \mathcal{S}} \beta(z_{t+1})\, p(\mathbf{y}_{t+1} | z_{t+1}, \Theta^{\text{old}})\, p(z_{t+1} | z_t, \Theta^{\text{old}}). \tag{7.23}$$

Proof. By marginalizing the definition of $\beta(z_t)$ (7.10b) over z_{t+1} via the sum rule of probability (E.1a) we get

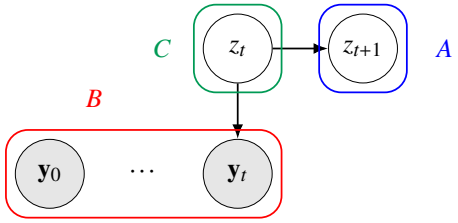
$$\begin{aligned} \beta(z_t) &= p(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T | \mathbf{y}_0, \ldots, \mathbf{y}_t, z_t, \Theta^{\text{old}}) \\ &= \sum_{z_{t+1} \in \mathcal{S}} p(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T, z_{t+1} | \mathbf{y}_0, \ldots, \mathbf{y}_t, z_t, \Theta^{\text{old}}). \end{aligned}$$

By applying the product rule, we get

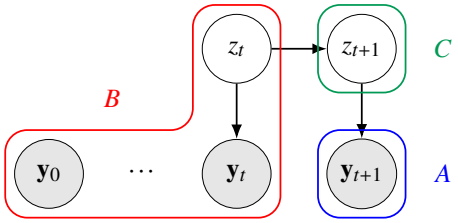
$$\beta(z_t) = \sum_{z_{t+1} \in \mathcal{S}} p(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T | \mathbf{y}_0, \ldots, \mathbf{y}_t, z_t, z_{t+1}, \Theta^{\text{old}})\, p(z_{t+1} | \mathbf{y}_0, \ldots, \mathbf{y}_t, z_t, \Theta^{\text{old}}).$$

By using the conditional independence property (see Figure 7.4a)

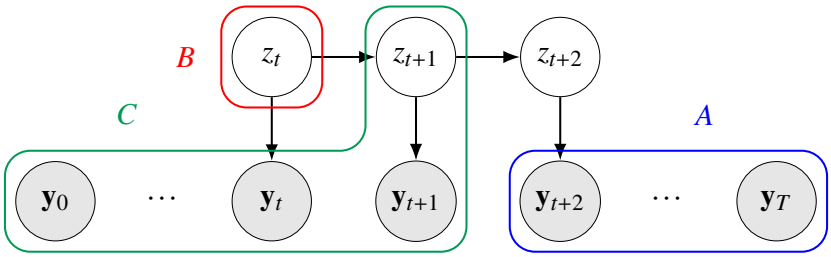
$$z_{t+1} \perp\!\!\!\perp \mathbf{y}_0, \ldots, \mathbf{y}_t | z_t, \Theta^{\text{old}} \tag{7.24}$$



(a) Conditional independence property (7.24): the arrows in the path from any element in the set A to any element of B meet only tail-to-tail in C.



(b) Conditional independence property (7.25): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.



(c) Conditional independence property (7.26): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.

Figure 7.4: Graphical proof of conditional independence properties of proof of Proposition 7.3. All the diagrams show conditional independence $A \perp\!\!\!\perp B|C$.

we can simplify the second factor as $p(z_{t+1}|z_t, \Theta^{\text{old}})$. Moreover, by using the product rule on the first factor, we can write $\beta(z_t)$ as

$$\beta(z_t) = \sum_{z_{t+1} \in \mathcal{S}} p(\mathbf{y}_{t+2}, \ldots, \mathbf{y}_T | \mathbf{y}_0, \ldots, \mathbf{y}_{t+1}, z_t, z_{t+1}, \Theta^{\text{old}}) p(\mathbf{y}_{t+1} | \mathbf{y}_0, \ldots, \mathbf{y}_t, z_t, z_{t+1}, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}}).$$

By using the conditional independence properties (see Figure 7.4b)

$$\mathbf{y}_{t+1} \perp\!\!\!\perp \mathbf{y}_0, \ldots, \mathbf{y}_t, z_t | z_{t+1}, \Theta^{\text{old}} \tag{7.25}$$

and (see Figure 7.4c)

$$\mathbf{y}_{t+2}, \ldots, \mathbf{y}_T \perp\!\!\!\perp z_t | \mathbf{y}_0, \ldots, \mathbf{y}_{t+1}, z_{t+1}, \Theta^{\text{old}}; \tag{7.26}$$

we obtain the recursive formula

$$\begin{aligned}\beta(z_t) &= \sum_{z_{t+1} \in \mathcal{S}} p(\mathbf{y}_{t+2}, \ldots, \mathbf{y}_T | \mathbf{y}_0, \ldots, \mathbf{y}_{t+1}, z_{t+1}, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \Theta^{\text{old}}) p(z_{t+1}|z_t, \Theta^{\text{old}}) \\ &= \sum_{z_{t+1} \in \mathcal{S}} \beta(z_{t+1})\, p(\mathbf{y}_{t+1}|z_{t+1}, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}}).\end{aligned}$$

□

The basis of recursion is given by observing that, for $t = T$, the following identity holds:

$$\gamma(z_T) = \frac{\alpha(z_T)}{p(\mathbf{Y}|\Theta^{\text{old}})},$$

which, combined with identity (7.9) gives $\beta(z_T) = 1$.

In practice, computing the forward and backward variables using these formulae will often result in numerical underflow. To solve this issue, we will now introduce the *scaled* (also known as *normalized*) forward and backward variables $\hat{\alpha}(z_t)$ and $\hat{\beta}(z_t)$.
Starting with the forward variable, we define the scaled version of $\alpha(z_t)$ as

$$\hat{\alpha}(z_t) \stackrel{\text{def}}{=} p(z_t|\mathbf{y}_0, \ldots, \mathbf{y}_t, \Theta^{\text{old}}).$$

By the product rule of probability (E.2), we have that scaled and "classical" forward variables are related as follows:

$$\begin{aligned} \hat{\alpha}(z_t) &= \frac{p(\mathbf{y}_0, \ldots, \mathbf{y}_t, z_t|\Theta^{\text{old}})}{p(\mathbf{y}_0, \ldots, \mathbf{y}_t|\Theta^{\text{old}})} \\ &= \frac{\alpha(z_t)}{p(\mathbf{y}_0, \ldots, \mathbf{y}_t|\Theta^{\text{old}})}. \end{aligned} \tag{7.27}$$

In order to relate the original and scaled forward variables, we define the scaling factor

$$c_t = p(\mathbf{y}_t|\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}, \Theta^{\text{old}}). \tag{7.28}$$

From the product rule of probability (E.2), we have that we can express the joint probability of the first $t + 1$ observations as

$$p(\mathbf{y}_0, \ldots, \mathbf{y}_t) = \prod_{\tau=0}^{t} c_\tau. \tag{7.29}$$

Thus, we can relate classical and scaled forward variables, by using the product rule, as

$$\begin{aligned} \alpha(z_t) &= p(\mathbf{y}_0, \ldots, \mathbf{y}_t, z_t|\Theta^{\text{old}}) \\ &= p(z_t|\mathbf{y}_0, \ldots, \mathbf{y}_t, \Theta^{\text{old}})\, p(\mathbf{y}_0, \ldots, \mathbf{y}_t|\Theta^{\text{old}}) \end{aligned}$$

obtaining

$$\alpha(z_t) = \left(\prod_{\tau=0}^{t} c_\tau \right) \hat{\alpha}(z_t).$$

Now, we can rewrite recursive relation (7.18) using this definition as

$$\underbrace{\left(\prod_{\tau=0}^{t} c_\tau\right)\hat{\alpha}(z_t)}_{\alpha(z_t)} = p(\mathbf{y}_t|z_t,\Theta^{\text{old}}) \sum_{z_{t-1}\in\mathcal{S}} \underbrace{\left(\prod_{\tau=0}^{t-1} c_\tau\right)\hat{\alpha}(z_{t-1})}_{\alpha(z_{t-1})} p(z_t|z_{t-1},\Theta^{\text{old}}).$$

The product $\prod_{\tau=0}^{t-1} c_\tau$ does not depend on z_{t-1} and can be moved outside the integral. Then, by dividing both sides by the quantity $\prod_{\tau=0}^{t-1} c_\tau$:

$$\frac{\prod_{\tau=0}^{t} c_\tau}{\prod_{\tau=0}^{t-1} c_\tau}\hat{\alpha}(z_t) = p(\mathbf{y}_t|z_t,\Theta^{\text{old}}) \sum_{z_{t-1}\in\mathcal{S}} \hat{\alpha}(z_{t-1}) p(z_t|z_{t-1},\Theta^{\text{old}}).$$

By simplifying, we obtain the recursive formula for the scaled forward variable

$$c_t\,\hat{\alpha}(z_t) = p(\mathbf{y}_t|z_t,\Theta^{\text{old}}) \sum_{z_{t-1}\in\mathcal{S}} \hat{\alpha}(z_{t-1}) p(z_t|z_{t-1},\Theta^{\text{old}}). \tag{7.30}$$

The basis of recursion is equal to, from (7.10a), $p(z_0|\mathbf{y}_0,\Theta^{\text{old}})$, which can be written, thanks to Bayes' theorem (E.3),

$$\begin{aligned}\hat{\alpha}(z_0) &= p(z_0|\mathbf{y}_0,\Theta^{\text{old}}) \\ &= \frac{p(\mathbf{y}_0|z_0,\Theta^{\text{old}})\,p(z_0|\Theta^{\text{old}})}{p(\mathbf{y}_0|\Theta^{\text{old}})}.\end{aligned}$$

The terms of the numerator are trivially given by the set of parameters, while the denominator can be obtained as the constant of normalization. Similarly, terms c_t in (7.30) can be obtained by normalizing its right hand side.

We now discuss the normalized backward variable. We start by defining it as the ratio

$$\begin{aligned}\hat{\beta}(z_t) &\overset{\text{def}}{=} \left(\prod_{\tau=t+1}^{T} c_\tau\right)^{-1} \beta(z_t) \\ &= \frac{p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T|z_t)}{p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T|\mathbf{y}_0,\ldots,\mathbf{y}_t)}.\end{aligned} \tag{7.31}$$

By applying the recursive formula (7.23) with the definition (7.31) we obtain

$$\underbrace{\left(\prod_{\tau=t+1}^{T} c_\tau\right)\hat{\beta}(z_t)}_{\beta(z_t)} = \sum_{z_{t+1}\in\mathcal{S}} \underbrace{\left(\prod_{\tau=t+2}^{T} c_\tau\right)\hat{\beta}(z_{t+1})}_{\beta(z_{t+1})}\, p(\mathbf{y}_{t+1}|z_{t+1},\Theta^{\text{old}})\, p(z_{t+1}|z_t,\Theta^{\text{old}}).$$

Similarly to the scaled forward variable recursion, we can move the product $\prod_{\tau=t+2}^{T} c_\tau$ outside the integral and simplify, obtaining the recursive formula

$$\hat{\beta}(z_t) = \frac{1}{c_{t+1}} \sum_{z_{t+1}\in\mathcal{S}} \hat{\beta}(z_{t+1})\, p(\mathbf{y}_{t+1}|z_{t+1},\Theta^{\text{old}})\, p(z_{t+1}|z_t,\Theta^{\text{old}}). \tag{7.32}$$

We will discuss the basis of recursion later.

From the development of the scaled variables, we observe that the likelihood function can be computed as

$$p(\mathbf{Y}|\Theta^{\text{old}}) = p(\mathbf{y}_0, \ldots, \mathbf{y}_T|\Theta^{\text{old}}) = \prod_{\tau=0}^{T} c_\tau.$$

From the scaled variables, we now show how to obtain the marginals γ and ξ. Starting with the marginal γ, we recall

$$\gamma(z_t) = \frac{\alpha(z_t)\beta(z_t)}{p(\mathbf{Y}|\Theta^{\text{old}})} = \frac{\alpha(z_t)\beta(z_t)}{\prod_{\tau=0}^{T} c_\tau}.$$

The product at the denominator can be splitted as

$$\prod_{\tau=0}^{T} c_\tau = \left(\prod_{\tau=0}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right)$$

thus giving

$$\begin{aligned}\gamma(z_t) &\frac{\alpha(z_t)\beta(z_t)}{\left(\prod_{\tau=0}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right)} \\ &= \frac{\alpha(z_t)}{\prod_{\tau=0}^{t} c_\tau} \frac{\beta(z_t)}{\prod_{\tau=t+1}^{T} c_\tau} \\ &= \hat{\alpha}(z_t)\hat{\beta}(z_t). \end{aligned} \tag{7.33}$$

This relation give us the basis of recursion for the scaled backward variable. Indeed, let us observe that, for $t = T$,

$$\gamma(z_T) = \hat{\alpha}(z_T),$$

thus giving

$$\hat{\beta}(z_T) = 1.$$

Let us now consider the marginal ξ. We recall

$$\begin{aligned}\xi(z_t, z_{t+1}) &= \frac{\beta(z_{t+1})p(z_{t+1}|z_t, \Theta^{\text{old}})p(\mathbf{y}_{t+1}|z_t, \Theta^{\text{old}})\alpha(z_t)}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{\beta(z_{t+1})p(z_{t+1}|z_t, \Theta^{\text{old}})p(\mathbf{y}_{t+1}|z_t\Theta^{\text{old}})\alpha(z_t)}{\left(\prod_{\tau=0}^{t} c_\tau\right) c_{t+1} \left(\prod_{\tau=t+2}^{T} c_\tau\right)}\end{aligned}$$

Thus getting the relation

$$\xi(z_t, z_{t+1}) = c_{t+1}^{-1}\hat{\beta}(z_{t+1})p(z_{t+1}|z_t, \Theta^{\text{old}})p(\mathbf{y}_{t+1}|z_t, \Theta^{\text{old}})\hat{\alpha}(z_t). \tag{7.34}$$

In summary, the Expectation step reduces to the recursive computation of scaled forward variable $\hat{\alpha}(z_t)$ and backward variable $\hat{\beta}(z_t)$, to then compute the marginals γ and ξ using (7.33) and (7.34). This recursive procedure takes the name of *Forward Backward algorithm* and is summarized in Algorithm 7.1

Algorithm 7.1 Forward Backward Algorithm for HMM

Input: Set of parameters Θ^{old}, observations $\mathbf{Y}$.

Output: Forward variable $\alpha(z_t)$, backward variable $\beta(z_t)$

▹ Compute the scaled forward variable $\hat{\alpha}(z_t)$

Compute $\hat{\alpha}(z_0) \propto p(\mathbf{y}_0|z_0, \Theta^{\text{old}})p(z_0|\Theta^{\text{old}})$.

Compute c_0 as the normalization term of $\hat{\alpha}(z_0)$.

for $t = 1, 2, \ldots, T$ **do**

    Compute $\hat{\alpha}(z_t) \propto p(\mathbf{y}_t|, z_t, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \hat{\alpha}(z_{t-1})\, p(z_t|z_{t-1}, \Theta^{\text{old}}))$.

    Compute c_t as the normalization term of $\hat{\alpha}(z_t)$.

end for

▹ Compute the scaled backward variable $\hat{\beta}(z_t)$

Inizialize $\hat{\beta}(z_T) = 1$.

for $t = T-1, T-2, \ldots, 0$ **do**

    Compute $\hat{\beta}(z_t) = c_{t+1}^{-1} \sum_{z_{t+1} \in \mathcal{S}} \hat{\beta}(z_{t+1}) p(\mathbf{y}_{t+1}|z_{t+1}, \Theta^{\text{old}}) p(z_{t+1}|z_t, \Theta^{\text{old}})$.

end for

Maximization Step

In the M-step, we aim at computing the new set of parameters Θ that maximizes $\widetilde{Q}(\Theta, \Theta^{\text{old}})$ in (7.6). To do so, we discuss maximization over initial state probability $\boldsymbol{\varpi}$, transition matrix $\mathbf{T}$, and emission probability separately. This can be done since these terms appear in different summations in (7.6) and maximization of $\widetilde{Q}$ can be achieved by maximizing each of the three components separately.

Starting with the *initial state probability*, we have to solve the following maximization problem:

$$\begin{aligned} \boldsymbol{\varpi}^\star &= \arg\max_{\boldsymbol{\varpi}} \sum_{i \in \mathcal{S}} \Big(\log \varpi_i \Pr(z_0 = i|\mathbf{Y}, \Theta^{\text{old}}) \Big), \\ \text{s.t.} \quad & \sum_{i \in \mathcal{S}} \varpi_i = 1. \end{aligned} \tag{7.35}$$

Proposition 7.4. *Problem* (7.35) *is solved by setting*

$$\boldsymbol{\varpi}|_i = \Pr(z_0 = i|\mathbf{Y}, \Theta^{\text{old}}). \tag{7.36}$$

Proof. From the Lagrange multiplier method, we define the Lagrangian

$$\mathcal{L}(\boldsymbol{\varpi}, \lambda) = \sum_{i \in \mathcal{S}} (\log \varpi_i \gamma_i(z_0)) - \lambda \left(\sum_{i \in \mathcal{S}} \varpi_i - 1 \right),$$

where, by definition (7.56a), $\gamma_i(z_0) = \Pr(z_0 = i|\mathbf{Y}, \Theta^{\text{old}})$. By computing the partial derivative w.r.t. ϖ_k, we get

$$\frac{\partial \mathcal{L}}{\partial \varpi_k} = \sum_{i=1}^{S} \left(\frac{\delta_i^k}{\varpi_i} \gamma_i(z_0) \right) - \lambda \left(\sum_{i=1}^{S} \delta_i^k \right)$$

$$= \frac{\gamma_k(z_0)}{\varpi_k} - \lambda,$$

where $\delta_{\cdot}^{\cdot}$ is the Dirac delta function:

$$\delta_i^k = \begin{cases} 1 & \text{if i = k} \\ 0 & \text{otherwise} \end{cases}.$$

By setting the partial derivative to zero, we get

$$\lambda = \frac{\gamma_k(z_0)}{\varpi_k}, \quad \forall k \in \{1, 2, \ldots, S\}. \tag{7.37}$$

By multiplying both side by ϖ_k and summing over k we obtain

$$\begin{aligned} &\sum_{k=1}^{S} \lambda \varpi_k = \sum_{k=1}^{S} \frac{\gamma_k(z_0)}{\varpi_k} \varpi_k \\ &\quad \Leftrightarrow \quad \lambda \underbrace{\sum_{k=1}^{S} \varpi_k}_{=1} = \underbrace{\sum_{k=1}^{S} \gamma_k(z_0)}_{=1}, \end{aligned}$$

from which $\lambda = 1$. Substituing this identity in (7.37), we get the update rule

$$\varpi_k = \gamma_k(z_0),$$

which is exactly (7.36). □

Talking about *transition probability*, we want to solve the maximization problem:

$$\begin{aligned} &\mathbf{T}^\star = \arg\max_{\mathbf{T}} \sum_{t=0}^{T-1} \sum_{i=1}^{S} \sum_{j=1}^{S} \log \mathbf{T}|_{ij} \Pr(z_t = i, z_{t+1} = j | \mathbf{Y}, \Theta^{\text{old}}), \\ &\text{s.t.} \quad \sum_j \mathbf{T}|_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, S\}. \end{aligned} \tag{7.38}$$

Proposition 7.5. *Problem* (7.38) *is solved by setting*

$$\mathbf{T}|_{k\ell} = \frac{\sum_{t=0}^{T-1} \Pr(z_t = k, z_{t+1} = \ell | \mathbf{Y}, \Theta^{\text{old}})}{\sum_{t=0}^{T-1} \Pr(z_t = k | \mathbf{Y}, \Theta^{\text{old}})}. \tag{7.39}$$

Proof. By defining $\xi_{ij}(t) = \Pr(z_t = i, z_{t+1} = j | \mathbf{Y}, \Theta^{\text{old}})$, and applying the Lagrange multiplier method, we define the Lagrangian

$$\mathcal{L}(\mathbf{T}, \boldsymbol{\lambda}) = \sum_{t=0}^{T-1} \sum_{i=1}^{S} \sum_{j=1}^{S} \log(\mathbf{T}|_{ij}) \xi_{ij}(t) - \sum_{i=1}^{S} \lambda_i \left(\sum_{j=1}^{S} \mathbf{T}|_{ij} - 1 \right).$$

Let us compute the partial derivative of the Lagrangian w.r.t. $\mathbf{T}|_{k\ell}$:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{T}|_{k\ell}} &= \sum_{t=0}^{T-1}\sum_{i=1}^{S}\sum_{j=1}^{S}\frac{\delta_i^k\delta_j^\ell}{\mathbf{T}|_{ij}}\xi_{ij}(t) - \sum_{i=1}^{S}\lambda_i\sum_{j=1}^{S}\delta_i^k\delta_j^l \\ &= \sum_{t=0}^{T-1}\frac{\xi_{k\ell}(t)}{\mathbf{T}|_{k\ell}} - \lambda_k.\end{aligned}$$

By setting the derivative to zero, we get

$$\lambda_k = \sum_{t=0}^{T-1}\frac{\xi_{k\ell}(t)}{\mathbf{T}|_{k\ell}}, \quad \forall l \in \{1,2,\dots,S\}. \tag{7.40}$$

By multiplying both side by $\mathbf{T}|_{k\ell}$ and summing over ℓ we get

$$\begin{aligned}&\sum_{\ell=1}^{S}\mathbf{T}|_{k\ell}\lambda_k = \sum_{\ell=1}^{S}\sum_{t=0}^{T-1}\frac{\xi_{k\ell}(t)}{\mathbf{T}|_{k\ell}}\mathbf{T}|_{k\ell} \\ \Leftrightarrow \quad &\lambda_k\underbrace{\sum_{l=1}^{S}\mathbf{T}|_{k\ell}}_{=1} = \sum_{t=0}^{T-1}\sum_{\ell=1}^{S}\xi_{k\ell}(t) \\ \Leftrightarrow \quad &\lambda_k = \sum_{t=0}^{T-1}\gamma_k(t),\end{aligned}$$

where in the last step we used (7.8). Substituting in (7.40) we obtain the update rule

$$\begin{aligned}\mathbf{T}|_{k\ell} &= \frac{\sum_{t=0}^{T-1}\xi_{k\ell}(t)}{\lambda_k} \\ &= \frac{\sum_{t=0}^{T-1}\xi_{k\ell}(t)}{\sum_{t=0}^{T-1}\gamma_k(t)},\end{aligned}$$

which coincides with (7.39) □

Maximization over the emission probability depends on the particular choice of emission model [8].

Proposition 7.6. *Maximization over the emission probability for the* discrete-emission HMM *is achieved by setting*

$$\mathbf{\Omega}|_{ij} = \frac{\sum_{t=0}^{T}\Pr(z_t = i|\mathbf{Y},\Theta^{\text{old}})\,\delta_{\mathbf{y}_t}^j}{\sum_{t=0}^{T}\Pr(z_t = i|\mathbf{Y},\Theta^{\text{old}})}. \tag{7.41}$$

Proof. Let us start by remarking that

$$p(\mathbf{y}|z) = \prod_{i=1}^{S}\prod_{j=1}^{Y}\omega_{ij}^{\delta_{\mathbf{y}}^j\delta_z^i}, \tag{7.42}$$

where $\omega_{ij} = \boldsymbol{\Omega}|_{ij}$.
We aim at maximizing

$$\widetilde{Q}_{\text{out}}(\boldsymbol{\Omega}) = \sum_{t=0}^{T}\sum_{i=1}^{S} \log(p(\mathbf{y}_t|z_t = i)) \Pr(z_t = i|\mathbf{Y}, \Theta^{\text{old}}). \tag{7.43}$$

By substituting (7.42) in (7.43) and by calling $\gamma_i(t) = \Pr(z_t = i|\mathbf{Y}, \Theta^{\text{old}})$, we get

$$\begin{aligned}\widetilde{Q}_{\text{out}}(\boldsymbol{\Omega}) &= \sum_{t=0}^{T}\sum_{i=1}^{S} \gamma_i(t) \log\left(\prod_{j=1}^{Y} \omega_{ij}^{\delta_{\mathbf{y}}^j \delta_z^i}\right) \\ &= \sum_{t=0}^{T}\sum_{i=1}^{S} \gamma_i(t) \sum_{j=1}^{Y} \log\left(\omega_{ij}^{\delta_{\mathbf{y}}^j \delta_z^i}\right).\end{aligned}$$

We thus obtain the maximization problem

$$\begin{aligned}&\boldsymbol{\Omega}^\star = \arg\max_{\boldsymbol{\Omega}} \sum_{t=0}^{T}\sum_{i=1}^{S} \gamma_i(t) \sum_{j=1}^{Y} \log\left(\omega_{ij}^{\delta_{\mathbf{y}}^j \delta_z^i}\right) \\ &\text{s.t.} \quad \sum_{j=1}^{Y} \omega_{ij} = 1 \quad \forall i \in \{1, 2, \ldots, S\}.\end{aligned}$$

We aim to solve this maximization problem via the Lagrange multiplier method. The Lagrangian is

$$\mathcal{L}(\boldsymbol{\Omega}, \boldsymbol{\lambda}) = \left(\sum_{t=0}^{T}\sum_{i=1}^{S} \gamma_i(t) \sum_{j=1}^{Y} \log\left(\omega_{ij}^{\delta_{\mathbf{y}}^j \delta_z^i}\right)\right) - \left(\sum_{i=1}^{S} \lambda_i \left(\sum_{j=1}^{Y} \omega_{ij} - 1\right)\right).$$

By taking the partial derivative w.r.t. $\omega_{k\ell}$ we obtain

$$\frac{\partial \mathcal{L}}{\partial \omega_{k\ell}} = \sum_{t=0}^{T} \gamma_k(t) \frac{\delta_{\mathbf{y}_t}^{\ell}}{\omega_{k\ell}} - \lambda_k.$$

By setting the derivative to zero, we get

$$\lambda_k = \sum_{t=0}^{T} \gamma_k(t) \frac{\delta_{\mathbf{y}_t}^{\ell}}{\omega_{k\ell}}.$$

Multiplying both side by $\omega_{k\ell}$ and summing over ℓ we get

$$\begin{aligned}&\sum_{\ell=1}^{Y} \lambda_k \omega_{k\ell} = \sum_{t=0}^{T}\sum_{\ell=1}^{Y} \gamma_k(t) \frac{\delta_{\mathbf{y}_t}^{\ell}}{\omega_{k\ell}} \omega_{k\ell} \\ &\quad \Leftrightarrow \quad \lambda_k = \sum_{t=0}^{T}\sum_{\ell=1}^{Y} \gamma_k(t) \delta_{\mathbf{y}_t}^{\ell}.\end{aligned}$$

Substituting we obtain

$$\begin{aligned}
&\sum_{t=0}^{T}\gamma_k(t)\frac{\delta^{\ell}_{\mathbf{y}_t}}{\omega_{k\ell}}-\sum_{t=0}^{T}\sum_{\ell=1}^{Y}\gamma_k(t)\delta^{\ell}_{\mathbf{y}_t}\\
&\quad\Leftrightarrow\quad \frac{1}{\omega_{k\ell}}\sum_{t=0}^{T}\gamma_k(t)\delta^{\ell}_{\mathbf{y}_t}-\sum_{t=0}^{T}\gamma_k(t)\sum_{\ell=1}^{Y}\delta^{\ell}_{\mathbf{y}_t}\\
&\quad\Leftrightarrow\quad \frac{1}{\omega_{k\ell}}\sum_{t=0}^{T}\gamma_k(t)\delta^{\ell}_{\mathbf{y}_t}-\sum_{t=0}^{T}\gamma_k(t).
\end{aligned}$$

By setting this last quantity to zero, we get

$$\omega_{k\ell}=\frac{\sum_{t=0}^{T}\gamma_k(t)\,\delta^{\ell}_{\mathbf{y}_t}}{\sum_{t=0}^{T}\gamma_k(t)},$$

which gives (7.41) □

Proposition 7.7. *Maximization over the emission probability for the* Gaussian-emission HMM *is achieved by setting*

$$\boldsymbol{\mu}_k=\frac{\sum_{t=0}^{T}\Pr(z_t=k|\mathbf{Y},\Theta^{\text{old}})\,\mathbf{y}_t}{\sum_{t=0}^{T}\Pr(z_t=k|\mathbf{Y},\Theta^{\text{old}})}, \tag{7.44a}$$

$$\boldsymbol{\Sigma}_k=\frac{\sum_{t=0}^{T}\Pr(z_t=k|\mathbf{Y},\Theta^{\text{old}})\,(\mathbf{y}_t-\boldsymbol{\mu}_k)^{\intercal}(\mathbf{y}_t-\boldsymbol{\mu}_k)}{\sum_{t=0}^{T}\Pr(z_t=k|\mathbf{Y},\Theta^{\text{old}})}. \tag{7.44b}$$

Proof. Since the emission probability of each mode i, $i=1,2,\ldots,S$ is a normal distribution, $(\mathbf{y}|z=i)\sim\mathcal{N}(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)$, we have

$$p(\mathbf{y}|z=i)=\frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}_i|^{1/2}}\exp\left(-\frac{1}{2}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)^{\intercal}\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)\right)$$

We aim at maximize, w.r.t. $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ function

$$\begin{aligned}
\widetilde{Q}_{\text{out}}(\Theta,\Theta^{\text{old}})&=\sum_{t=0}^{T}\sum_{i=1}^{S}\Pr(z_t=i|\mathbf{Y},\Theta^{\text{old}})\log p(\mathbf{y}_t|z_t=i)\\
&=\sum_{t=0}^{T}\sum_{i=1}^{S}\gamma_i(t)\log\left(\frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}_i|^{1/2}}\exp\left(-\frac{1}{2}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)^{\intercal}\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)\right)\right),
\end{aligned} \tag{7.45}$$

where $\gamma_i(t)=\Pr(z_t=i|\mathbf{Y},\Theta^{\text{old}})$. By logarithm rule, we can write (7.45) as follows:

$$\begin{aligned}
\widetilde{Q}_{\text{out}}(\Theta,\Theta^{\text{old}})&=\sum_{t=0}^{T}\sum_{i=1}^{S}\gamma_i(t)\left(-\log\left((2\pi)^{d/2}\,|\boldsymbol{\Sigma}_i|^{1/2}\right)+\log\left(\exp\left(-\frac{1}{2}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)^{\intercal}\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)\right)\right)\right)\\
&=\sum_{t=0}^{T}\sum_{i=1}^{S}\gamma_i(t)\left(-\frac{d}{2}\log(2\pi)-\frac{1}{2}\log(|\boldsymbol{\Sigma}_i|)-\frac{1}{2}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)^{\intercal}\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{y}_t-\boldsymbol{\mu}_i)\right).
\end{aligned}$$

Since $^d/_2 \log(2\pi)$ is a constant, we can ignore it when maximizing $\widetilde{Q}_{\text{out}}$. Thus we aim at maximize the following function w.r.t. $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$:

$$\widetilde{\widetilde{Q}}_{\text{out}}(\Theta, \Theta^{\text{old}}) = \sum_{t=0}^{T} \sum_{i=1}^{S} \gamma_i(t) \left(-\frac{1}{2} \log(|\boldsymbol{\Sigma}_i|) - \frac{1}{2} (\mathbf{y}_t - \boldsymbol{\mu}_i)^\intercal \, \boldsymbol{\Sigma}_i^{-1} \, (\mathbf{y}_t - \boldsymbol{\mu}_i) \right). \quad (7.46)$$

Maximization over $\boldsymbol{\mu}_i$ is achieved by setting $\partial_{\mu_k} \widetilde{\widetilde{Q}}_{\text{out}}$ to zero:

$$\begin{aligned}
&\frac{\partial \widetilde{\widetilde{Q}}_{\text{out}}(\Theta, \Theta^{\text{old}})}{\partial \boldsymbol{\mu}_k} = \mathbf{0} \\
\Leftrightarrow \quad & \sum_{t=0}^{T} \sum_{i=1}^{S} \gamma_i(t) \left(\boldsymbol{\Sigma}^{-1} (\mathbf{y}_t - \mu_i) \right) \delta_i^k = \mathbf{0} \\
\Leftrightarrow \quad & \sum_{t=0}^{T} \gamma_k(t) \boldsymbol{\Sigma}_k^{-1} (\mathbf{y}_t - \boldsymbol{\mu}_k) = \mathbf{0} \\
\Leftrightarrow \quad & \boldsymbol{\mu}_k \sum_{t=0}^{T} \gamma_k(t) = \sum_{t=0}^{T} \gamma_k(t) \mathbf{y}_t,
\end{aligned}$$

from which the update rule follows:

$$\boldsymbol{\mu}_k = \frac{\sum_{t=0}^{T} \gamma_k(t) \mathbf{y}_t}{\sum_{t=0}^{T} \gamma_k(t)},$$

which gives (7.44a).

Next, we aim at maximize (7.46) w.r.t. $\boldsymbol{\Sigma}_k$. To do so, we rewrite (7.46) as follows [8]:

$$\begin{aligned}
\widetilde{\widetilde{Q}}_{\text{out}}(\Theta, \Theta^{\text{old}}) &= \sum_{i=1}^{S} \left(\frac{1}{2} \log\left(\left| \boldsymbol{\Sigma}_i^{-1} \right| \right) \sum_{t=0}^{T} \gamma_i(t) - \frac{1}{2} \sum_{t=0}^{T} \gamma_i(t) \text{tr}\left(\boldsymbol{\Sigma}_i^{-1} (\mathbf{y}_t - \boldsymbol{\mu}_i)^\intercal (\mathbf{y}_t - \boldsymbol{\mu}_i) \right) \right) \\
&= \sum_{i=1}^{S} \left(\frac{1}{2} \log\left(\left| \boldsymbol{\Sigma}_i^{-1} \right| \right) \sum_{t=0}^{T} \gamma_i(t) - \frac{1}{2} \sum_{t=0}^{T} \gamma_i(t) \, \text{tr}\left(\boldsymbol{\Sigma}_i^{-1} \boldsymbol{\Psi}_i(t) \right) \right) \quad (7.47)
\end{aligned}$$

where we defined $\boldsymbol{\Psi}_i(t) = (\mathbf{y}_t - \boldsymbol{\mu}_i)^\intercal (\mathbf{y}_t - \boldsymbol{\mu}_i)$. We now must set the partial derivative of (7.47) w.r.t. $\boldsymbol{\Sigma}_k^{-1}$ equal to zero. Let us compute the partial derivative:

$$\begin{aligned}
\frac{\partial \widetilde{\widetilde{Q}}_{\text{out}}(\Theta, \Theta^{\text{old}})}{\partial \boldsymbol{\Sigma}_k^{-1}} &= \frac{1}{2} \sum_{t=0}^{T} \gamma_k(t) (2\boldsymbol{\Sigma}_k - \text{diag}(\boldsymbol{\Sigma}_k)) - \frac{1}{2} \sum_{t=0}^{T} \gamma_k(t) (2\boldsymbol{\Psi}_k(t) - \text{diag}(\boldsymbol{\Psi}_k(t))) \\
&= \frac{1}{2} \sum_{t=0}^{T} \gamma_k(t) \left(2\mathbf{M}_k(t) - \text{diag}(\mathbf{M}_k(t)) \right) \\
&= 2\boldsymbol{\Xi} - \text{diag}(\boldsymbol{\Xi}),
\end{aligned}$$

where $\mathbf{M}_k(t) = \boldsymbol{\Sigma}_k - \boldsymbol{\Psi}_k(t)$, and $\boldsymbol{\Xi} = {}^1/_2 \sum_{t=0}^{T} \gamma_k(t) \mathbf{M}_k(t)$. By setting this derivative to zero, we get

$$2\boldsymbol{\Xi} - \text{diag}(\boldsymbol{\Xi}) = \mathbf{0} \quad \Leftrightarrow \quad \boldsymbol{\Xi} = \mathbf{0},$$

Algorithm 7.2 Maximization Step for HMM

Input: Marginals $\gamma_i(t)$, $i \in \mathcal{S}$, $t \in \{1,2,T\}$ and $\xi_{ij}(t)$, $i,j \in \mathcal{S}$, $t \in 1,2,\dots,T-1$; observed data $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_T)$, $\mathbf{y}_t \in \mathbb{R}^d$, $t = 0,1,\dots,T$

Output: New set of parameter Θ

▹ Maximization over the initial mode probability density

1: Set

$$\varpi|_i = \gamma_i(1).$$

▹ Maximization over the transition probability matrix

2: Set

$$\mathbf{T}|_{k\ell} = \frac{\sum_{t=1}^{T-1} \xi_{k\ell}(t)}{\sum_{t=1}^{T-1} \gamma_k(t)}.$$

▹ Maximization over the AR dynamic parameters

3: **for** $s \in \mathcal{S} = \{1,2,\dots,S\}$ **do**

4: Set

in the case of discrete-emissions HMM

$$\mathbf{\Omega}|_{k\ell} = \frac{\sum_{t=0}^{T} \gamma_k(t)\, \delta^{\ell}_{\mathbf{y}_t}}{\sum_{t=0}^{T} \gamma_k(t)}.$$

in the case of Gaussian-emissions HMM

$$\mathbf{\Sigma}_k = \frac{\sum_{t=0}^{T} \gamma_k(t)\, (\mathbf{y}_t - \boldsymbol{\mu}_k)^{\intercal} (\mathbf{y}_t - \boldsymbol{\mu}_k)}{\sum_{t=0}^{T} \gamma_k(t)},$$

$$\boldsymbol{\mu}_k = \frac{\sum_{t=0}^{T} \gamma_k(t)\, \mathbf{y}_t}{\sum_{t=0}^{T} \gamma_k(t)}.$$

5: **end for**

which, by definition of $\mathbf{\Xi}$, reads

$$\begin{aligned} &\sum_{t=0}^{T} \gamma_k(t)\, \mathbf{M}_k(t) = \mathbf{0} \\ \Leftrightarrow \quad &\sum_{t=0}^{T} \gamma_k(t)\, (\mathbf{\Sigma}_k - \mathbf{\Psi}_k(t)) = \mathbf{0} \\ \Leftrightarrow \quad &\mathbf{\Sigma}_k = \frac{\sum_{t=0}^{T} \gamma_k(t)\, \mathbf{\Psi}_k(t)}{\sum_{t=0}^{T} \gamma_k(t)}, \end{aligned}$$

which, by expanding $\mathbf{\Psi}_k(t)$ gives (7.44b). □

We remark that, since the values for the marginal γ depend on the old set of parameters Θ^{old}, the mean $\boldsymbol{\mu}_k$ in (7.44b) is the 'old' value, and not that given from (7.44a). In practice, when implementing the maximization step, it is convenient to

update the covariance matrices first, and then update the means.

The M-step is summarized in Algorithm 7.2.

7.1.2. EM Algorithm for Multiple Observations

The EM algorithm development we presented assumes that a single stream of observation $\mathbf{Y}$ is available to perform the algorithm. Let us assume that a set of sequences of observations is given:

$$\Upsilon = \{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(K)}\}.$$

By assuming that each observation is independent to all other demonstrations given the set of parameters:

$$\mathbf{Y}^{(k)} \perp\!\!\!\perp \Upsilon \setminus \{\mathbf{Y}^{(k)}\} \,|\, \Theta,$$

we have that the maximization of the likelihood $p(\Upsilon|\Theta)$ can be achieved by maximizing the auxilary function [79]

$$\breve{Q}(\Theta, \Theta^{\text{old}}) = \sum_{k=1}^{K} \widetilde{Q}^{(k)}(\Theta, \Theta^{\text{old}}), \tag{7.48}$$

where $\widetilde{Q}^{(k)}$ is obtained from (7.3) by adding the superscrit $^{(k)}$ to $\mathbf{Y}$. By expanding (7.48), we get

$$\begin{aligned}\breve{Q}(\Theta, \Theta^{\text{old}}) = &\sum_{k=1}^{K} \sum_{z_1 \in \mathcal{S}} \log p(z_1|\Theta) p\left(z_1|\mathbf{Y}^{(k)}, \Theta^{\text{old}}\right) + \\ &\sum_{k=1}^{K} \sum_{t=0}^{T^{(k)}-1} \sum_{z_{t+1} \in \mathcal{S}} \log p\left(\mathbf{y}_{t+1}^{(k)}|z_{t+1}, \mathbf{y}_t^{(k)}, \Theta\right) p\left(z_{t+1}|\mathbf{Y}^{(k)}, \Theta^{\text{old}}\right) + \\ &\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}-1} \sum_{z_t \in \mathcal{S}} \sum_{z_{t+1} \in \mathcal{S}} \log p(z_{t+1}|z_t, \Theta) p\left(z_t, z_{t+1}|\mathbf{Y}^{(k)}, \Theta^{\text{old}}\right).\end{aligned}$$

We remark that each demostration $\mathbf{Y}^{(k)}$ in Υ may have different length w.r.t. all other demonstrations. Thus, each demonstration $\mathbf{Y}^{(k)}$ span over time indexes $0, 1, \ldots, T^{(k)}$.

The expectation step follows as presented in Section 7.1.1. The only difference is that a set of marginals $\gamma^{(k)}(z_t)$ and $\xi^{(k)}(z_t, z_{t+1})$ (and, subsequently, of forward and backward variables $\alpha^{(k)}(z_t)$ and $\beta^{(k)}(z_t)$) have to be computed for each demostration $\mathbf{Y}^{(k)}$ in Υ.

Maximization over the parameters is easy to extend. Indeed, since the summation is over the index of demonstration k, we can move the differentiations over

the parameters inside the summation itself when solving the maximization problems (see proofs of Proposition 7.4, Proposition 7.5, Proposition 7.6, and Proposition 7.7).
Maximization of the initial mode probability is achieved by setting

$$\varpi|_i = \frac{1}{K}\sum_{k=1}^{K} Pr(z_1 = i|\mathbf{Y}^{(k)}, \Theta^{\text{old}}). \tag{7.49a}$$

Maximization over the transition probability matrix is obtained by setting

$$\mathbf{T}|_{mn} = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T^{(k)}-1} \Pr(z_t = m, z_{t+1} = n|\mathbf{Y}^{(k)}, \Theta^{\text{old}})}{\sum_{k=1}^{K}\sum_{t=1}^{T^{(k)}-1} \Pr(z_t = m|\mathbf{Y}^{(k)}, \Theta^{\text{old}})}. \tag{7.49b}$$

Finally, maximization over the emission probabilities is achieved by setting

$$\mathbf{\Omega}|_{mn} = \frac{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}} \gamma_m^{(k)}(t)\delta^n_{\mathbf{y}_t^{(k)}}}{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}} \gamma_m(t)}, \tag{7.49c}$$

in the case of discrete emissions, and

$$\mathbf{\Sigma}_s = \frac{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}} \gamma_s(t)(\mathbf{y}_t^{(k)} - \boldsymbol{\mu}_s)^\intercal(\mathbf{y}_t^{(k)} - \boldsymbol{\mu}_s)}{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}} \gamma_s(t)}, \tag{7.49d}$$

$$\boldsymbol{\mu}_s = \frac{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}} \gamma_s(t)\mathbf{y}_t^{(k)}}{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}} \gamma_s(t)}. \tag{7.49e}$$

in the case of Gaussian emissions.

7.1.3. Parameter Initialization

The EM algorithm is a maximization procedure that aims at finding the set of parameters Θ that maximizes the likelihood $p(\mathbf{Y}|\Theta)$ of the observed data. However, this maximization problem has, in general, multiple local maxima. This means that, by applying the EM algorithm, it is unlikely to converge to the global maximum. Moreover, being an iterative method, the EM algorithm will reach a maximum that is 'near' the initial set of parameters. Thus, to obtain a good estimate of the parameters set Θ, a proper initialization is required.
In this Section, we discuss a way to obtain a proper initial estimate of parameters [77, 138] based on the *k-means clustering algorithm*, recalled in Appendix F.

The initialization works as follows. A first, the k-means algorithm is run by setting the number of clusters equal to the number of hidden modes of the HMM $k = S$.

After applying the clustering algorithm on the dataset $\Upsilon = \{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(K)}\}$, we obtain a candidate segmentation (i.e. a sequence of hidden modes) $\mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_T^{(k)})$ for each time series $\mathbf{Y}^{(k)} = (\mathbf{y}_0^{(k)}, \mathbf{y}_1^{(k)}, \ldots, \mathbf{y}_T^{(k)})$.
From this, we can initialize the initial mode probability ϖ as

$$\Pr(z_0 = i) = \frac{\#\text{ of times } z_0 = i}{K}. \tag{7.50}$$

Transition probabilities $\mathbf{T}$ is initialized as

$$\Pr(z_{t+1} = j | z_t = i) = \frac{\#\text{ of observed transitions from } i \text{ to } j}{\text{total \# of transitions}}. \tag{7.51}$$

Initialization of the emission probability for Gaussian emission is performed as follows. The mean μ_s and covariance matrix $\mathbf{\Sigma}_s$ for mode $s \in \{1, 2, \ldots, S\}$ are compute as the average and covariance of the emissions clustered in cluster s.

7.1.4. Viterbi Algorithm

Once the HMM's parameters have been learned, we aim at using the model to determine the sequence of modes $\mathbf{z} = (z_0, z_1, \ldots, z_T)$ that generated the observed sequence $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T)$. Clearly, we cannot compute the exact sequence. However, we can compute the sequence of modes that maximize the joint probability distribution $p(\mathbf{Y}, \mathbf{z})$:

$$\hat{\mathbf{z}} = \arg \max_{\mathbf{z} \in \mathcal{S}^{T+1}} p(\mathbf{Y}, \mathbf{z}). \tag{7.52}$$

We start by noticing that maximization problem 7.52 is equivalent to (since the logarithm is an increasing function)

$$\hat{\mathbf{z}} = \arg \max_{\mathbf{z} \in \mathcal{S}^{T+1}} \log p(\mathbf{Y}, \mathbf{z}).$$

We now define the variables [9]

$$\omega(z_t) \stackrel{\text{def}}{=} \max_{(z_0, \ldots, z_{t-1}) \in \mathcal{S}^t} \log p((\mathbf{y}_0, \ldots, \mathbf{y}_t), (z_0, \ldots, z_t)). \tag{7.53}$$

Function ω can be initialized as

$$\omega(z_0) = \log p(z_0) + \log p(\mathbf{y}_0 | z_0),$$

and recursively computed as

$$\omega(z_{t+1}) = \log p(\mathbf{y}_{t+1} | z_{t+1}) + \max_{z_t \in \mathcal{S}} (\log p(z_{t+1} | z_t) + \omega(z_t)).$$

From definition (7.53) we have that $\log p(\mathbf{Y}, \hat{\mathbf{z}}) = \arg\max_{z_T \in \mathcal{S}} \omega(z_T)$.

The *Viterbi algorithm* [132] is a procedure to find the most probable sequence of hidden modes given the sequence of observations. The algorithm takes as input a hidden-variable model $\mathcal{M}$ and a stream of observations $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_T)$, and returns the sequence of hidden modes $\mathbf{z} = (z_0, z_1, \dots, z_T) \in \mathcal{S}^{T+1}$ that maximizes $p(\mathbf{Y}|\mathbf{z}, \mathcal{M})$.
The algorithm generates two 2-dimensional tables $\mathtt{T}_1, \mathtt{T}_2$ of size $(T+1)\times S$. Element $\mathtt{T}_1|_{ij}$ of $\mathtt{T}_1$ contains the probability of the most likely path so far $\hat{\mathbf{z}} = (\hat{z}_1, \hat{z}_2, \dots, \hat{z}_i)$ with $\hat{z}_i = s_j$ that generates $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_2, \dots, \mathbf{y}_i)$. Element $\mathtt{T}_2|_{ij}$ of $\mathtt{T}_2$ contains the mode $\hat{z}_{i-1}$ of the most likely path so far $\hat{\mathbf{z}} = (\hat{z}_1, \hat{z}_2, \dots, \hat{z}_{i-1}, \hat{z}_i = j)$. The components of $\mathtt{T}_1, \mathtt{T}_2$ are recursively computed as

$$\begin{aligned}
\mathtt{T}_1|_{ij} &= \max_k \left(\mathtt{T}_{i-1,k} \Pr(z_{t+1} = j | z_t = k) \Pr(\mathbf{y}_t | z_t = s)\right), \\
\mathtt{T}_2|_{ij} &= \arg\max_k \left(\mathtt{T}_{i-1,k} \Pr(z_{t+1} = j | z_t = k) \Pr(\mathbf{y}_t | z_t = s)\right) \\
&= \arg\max_k \left(\mathtt{T}_{i-1,k} \Pr(z_{t+1} = j | z_t = k)\right).
\end{aligned}$$

We remark that last identity follows from the fact that $\Pr(\mathbf{y}_j | z_j = i)$ is positive and independent of index k. The basis of recursion is

$$\begin{aligned}
\mathtt{T}_1|_{1\,s} &= \Pr(z_0 = s) \Pr(\mathbf{y}_0 | z_0 = s), \\
\mathtt{T}_2|_{1\,s} &= 0.
\end{aligned}$$

The algorithm is summarized in Algorithm 7.3.

7.2. Auto-Regressive Hidden Markov Model

The Hidden Markov Model is widely used in speech recognition [65], natural language modeling [87], handwriting recognition[99], and for the analysis of biological sequences (e.g. proteins and DNA) [72, 32, 5].
However, since each observation is not related to the previous, the HMM is not able to capture correlations between the observed variables [9]. For this reason, in this Section, we present an improvement of the HMM that adds a relation between two consecutive nodes. In this way, each hidden mode of the model defines how the observation evolves, instead of governing directly what the observation is.

Auto-Regressive Hidden Markov Model (AR-HMM) is an extension of the classical Hidden Markov Models (HMM) in which the observations are related by an Auto-Regressive (AR) dynamic. In this model, the latent mode encodes the (linear) vector field governing the evolution of the observed state.

Definition 7.2. An *Auto-Regressive Hidden Markov Model* (AR-HMM) is a model $\mathcal{H} = \{\mathcal{S}, \mathcal{Y}, \Theta = \{\varpi, \mathbf{T}, \{\mathbf{A}_s, \mathbf{b}_s\}_{s\in\mathcal{S}}\}\}$ where:

Algorithm 7.3 Viterbi Algorithm - HMM

Input: HMM parameters Θ, observation sequence $\mathbf{Y} = (\mathbf{y}_0, \ldots, \mathbf{y}_T)$.

Output: Most likely sequence of hidden modes $\hat{\mathbf{z}} = (\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_T)$.

for $s = 1, , 2 \ldots S$ **do**
  $\mathrm{T}_1|_{0\,s} = \Pr(z_0 = s) \Pr(\mathbf{y}_0 | z_0 = s)$
  $\mathrm{T}_2|_{0\,s} = 0$
end for
for $t = 1, 2, \ldots, T$ **do**
  for do$s = 1, 2, \ldots, S$
    $\mathrm{T}_1|_{s\,t} = \max_k (\mathrm{T}_1|_{k\,t-1} \Pr(z_t = s | z_{t-1} = k)\, p(\mathbf{y}_t | z_t = s))$
    $\mathrm{T}_2|_{s\,t} = \arg\max_k (\mathrm{T}_1|_{k\,t-1} \Pr(z_t = s | z_{t-1} = k)\, p(\mathbf{y}_t | z_t = s))$
  end for
end for
$\hat{z}_T = \arg\max_k \mathrm{T}_1|_{k\,T}$
for $t = T, T-1, \ldots, 1$ **do**
  $\hat{z}_{t-1} = \mathrm{T}_2|_{\hat{z}_t, t}$
end for

- $\mathcal{S} = \{1, 2, \ldots, S\}$ is the set of (hidden) *modes*. The mode at time $t \in \{1, 2, \ldots, T\}$ is denoted by z_t.
- Vector $\boldsymbol{\varpi} = [\varpi_i]_{i=1,2\ldots,S} \in \Theta$ define the *initial probability* of hidden mode:

  $$\Pr(z_1 = i) = \boldsymbol{\varpi}_i, \quad i \in \mathcal{S}.$$

  Vector $\boldsymbol{\varpi}$ satisfies $\varpi_i \in [0, 1], \forall i = 1, 2, \ldots, S$ and $\sum_{i=1}^{S} \boldsymbol{\varpi}_i = 1$.
- Matrix $\mathbf{T} = [\mathbf{T}|_{i\,j}]_{i=1,2,\ldots,S}^{j=1,2,\ldots,S} \in \Theta$ defines the *transition probabilities* between hidden modes:

  $$\Pr(z_{t+1} = j | z_t = i) = \mathbf{T}|_{i\,j}, \quad i, j \in \mathcal{S}.$$

  Matrix $\mathbf{T}$ satisfies $\mathbf{T}|_{i\,j} \in [0, 1], \forall i, j = 1, 2, \ldots, S$ and $\sum_{j=1}^{S} \mathbf{T}_{i\,j} = 1, \forall i = 1, 2, \ldots, S$.
- $\mathcal{Y} = \mathbb{R}^d$ is the set of observations. The observation at time $t \in \{0, 1, \ldots, T\}$ is denoted by $\mathbf{y}_t \in \mathbb{R}^d$. The emissions are modeled by the following AR affine dynamics with Gaussian white noise:

  $$\mathbf{y}_{t+1} | z_{t+1}, \mathbf{y}_t \sim \mathcal{N}(\mathbf{A}_{z_{t+1}} \mathbf{y}_t + \mathbf{b}_{z_{t+1}}, \boldsymbol{\Sigma}_{z_{t+1}}).$$

The set Θ contains all the parameters of the model:

$$\Theta = \{\boldsymbol{\varpi}, \mathbf{T}, \boldsymbol{\Sigma}_s, \{\mathbf{A}_s, \mathbf{b}_s\}_{s \in \mathcal{S}}\}.$$

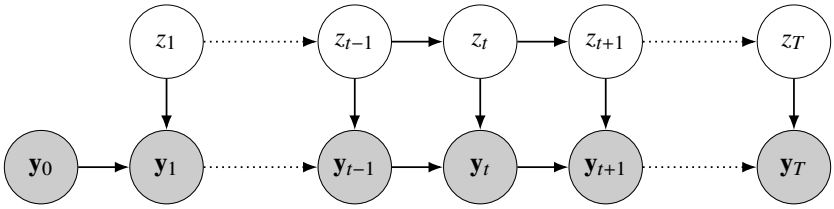


Figure 7.5: Graphical representation of an AR-HMM.

The graphical model representation is given in Figure 7.5.

In the following, we develop the Expectation-Maximization and Viterbi algorithms as we did for the classical HMM in Section 7.1.

7.2.1. Expectation-Maximization Algorithm

In the following, we will develop the *Expectation-Maximization* (EM) algorithm in the case of AR-HMM.
The complete-data likelihood of an AR-HMM is

$$p(\mathbf{Y}, \mathbf{z}) = p(z_1) \prod_{t=1}^{T-1} p(z_{t+1}|z_t) \prod_{t=0}^{T-1} p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t),$$

where $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T)$ denotes the sequence of observations, and $\mathbf{z} = (z_1, z_2, \ldots, z_T)$ denotes the sequence of hidden modes. The complete data log-likelihood is

$$\log p(\mathbf{Y}, \mathbf{z}) = \log p(z_1) + \sum_{t=1}^{T-1} \log p(z_{t+1}|z_t) + \sum_{t=0}^{T-1} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t). \tag{7.54}$$

During the learning process, we aim at maximizing

$$Q(\Theta, \Theta^{\text{old}}) = \sum_{\mathbf{z} \in \mathcal{S}^T} p(\mathbf{z}|\mathbf{Y}, \Theta^{\text{old}}) \log p(\mathbf{Y}, \mathbf{z}|\Theta),$$

where $\mathcal{S}^T$ denotes the set of all possible sequences $\mathbf{z}$. By expanding the term $\log p(\mathbf{Y}, \mathbf{z}|\Theta)$ in function Q we obtain (by suppressing the dependence on Θ for notation simplicity)

$$Q(\Theta, \Theta^{\text{old}}) = \sum_{\mathbf{z} \in \mathcal{S}^T} p(\mathbf{z}|\mathbf{Y}, \Theta^{\text{old}}) \left(\log p(z_1) + \sum_{t=1}^{T-1} \log p(z_{t+1}|z_t) + \sum_{t=0}^{T-1} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t) \right).$$

Next, we can expand the summation over the domain $\mathcal{S}^T$ as multiple sums over $\mathcal{S}$

$$Q(\Theta, \Theta^{\text{old}}) = \sum_{z_1 \in \mathcal{S}} \cdots \sum_{z_t \in \mathcal{S}} \cdots \sum_{z_T \in \mathcal{S}} p(z_1, z_2, \ldots, z_T|\mathbf{Y}, \Theta^{\text{old}})$$

$$\left(\log p(z_1) + \sum_{t=1}^{T-1} \log p(z_{t+1}|z_t) + \sum_{t=0}^{T-1} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t)\right).$$

Now, we can rearrange the sums to obtain

$$\begin{aligned} Q(\Theta, \Theta^{\text{old}}) = &\sum_{z_1 \in \mathcal{S}} \cdots \sum_{z_T \in \mathcal{S}} p(z_1, z_2, \ldots, z_T|\mathbf{Y}, \Theta^{\text{old}}) \log p(z_1) + \\ &\sum_{t=1}^{T-1} \sum_{z_1 \in \mathcal{S}} \cdots \sum_{z_T \in \mathcal{S}} p(z_1, z_2, \ldots, z_T|\mathbf{Y}, \Theta^{\text{old}}) \log p(z_{t+1}|z_t) + \\ &\sum_{t=0}^{T-1} \sum_{z_1 \in \mathcal{S}} \cdots \sum_{z_T \in \mathcal{S}} p(z_1, z_2, \ldots, z_T|\mathbf{Y}, \Theta^{\text{old}}) \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t). \end{aligned}$$

By marginalizing, we have that maximization over Q is equivalent to maximize the following function:

$$\begin{aligned} \widetilde{Q}(\Theta, \Theta^{\text{old}}) = &\sum_{z_1 \in \mathcal{S}} \log p(z_1|\Theta) p(z_1|\mathbf{Y}, \Theta^{\text{old}}) + \\ &\sum_{t=0}^{T-1} \sum_{z_{t+1} \in \mathcal{S}} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta) p(z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\ &\sum_{t=1}^{T-1} \sum_{z_t \in \mathcal{S}} \sum_{z_{t+1} \in \mathcal{S}} \log p(z_{t+1}|z_t, \Theta) p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}). \end{aligned} \tag{7.55}$$

At this point, the EM algorithm can be split in its two components: the *Expectation step* in which quantities depending on Θ^{old} are computed, and the *Maximization step* in which a new set of parameters Θ is computed in order to maximize $\widetilde{Q}(\Theta, \Theta^{\text{old}})$.

Expectation Step

During the E-step, we aim at computing the marginals γ and ξ as defined in (7.7):

$$\gamma(z_t) \stackrel{\text{def}}{=} p(z_t|\mathbf{Y}, \Theta^{\text{old}}), \quad t = 1, 2, \ldots, T; \tag{7.56a}$$

$$\xi(z_t, z_{t+1}) \stackrel{\text{def}}{=} p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}), \quad t = 1, 2, \ldots, T-1. \tag{7.56b}$$

As for the HMM case, we aim at writing marginals in (7.56) in terms of forward and backward variables. The definition of the forward variable α is the same as in (7.10a), while the definition of the backward variable β is slightly different.

Let us consider the marginal γ. By applying the product rule of probability two times we get

$$\gamma(z_t) = \frac{p(\mathbf{Y}, z_t|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})}$$

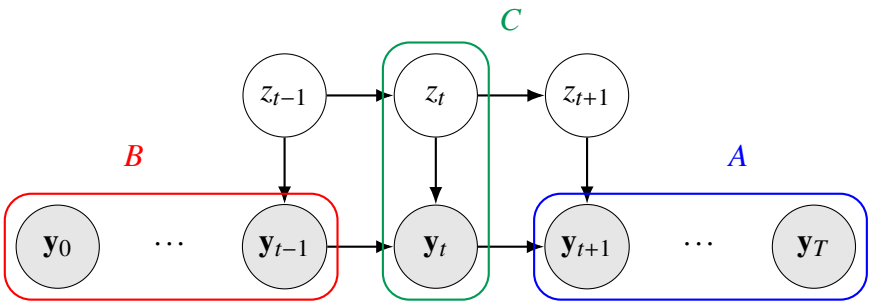


Figure 7.6: Graphical proof of conditional independence property (7.57). The conditional independence $A \perp\!\!\!\perp B|C$ follows from the fact that all paths from A to B meet head-to-tail in C. For a recall on graphical model and conditional independence, see Appendix E.

$$= \frac{p(\mathbf{y}_0, \dots, \mathbf{y}_t, \mathbf{y}_{t+1}, \dots, \mathbf{y}_T, z_t | \Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})}$$
$$= \frac{p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t | \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T | \mathbf{y}_0, \dots, \mathbf{y}_t, z_t, \Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})}$$

Now, by conditional independence property (see Figure 7.6)

$$\mathbf{y}_{t+1}, \dots, \mathbf{y}_T \perp\!\!\!\perp \mathbf{y}_0, \dots, \mathbf{y}_{t-1} | \mathbf{y}_t, z_t, \Theta^{\text{old}} \tag{7.57}$$

to the first factor at numerator, we can write $\gamma(z_t)$ as

$$\gamma(z_t) = \frac{\alpha(z_t)\beta(z_t)}{p(\mathbf{Y}|\Theta^{\text{old}})}. \tag{7.58}$$

Forward variable $\alpha(z_t)$ is defined as in (7.10a):

$$\alpha(z_t) \stackrel{\text{def}}{=} p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t | \Theta^{\text{old}}). \tag{7.59a}$$

Backward variable $\beta(z_{t+1})$ is slightly different from (7.10b) and reads, in the case of AR-HMM

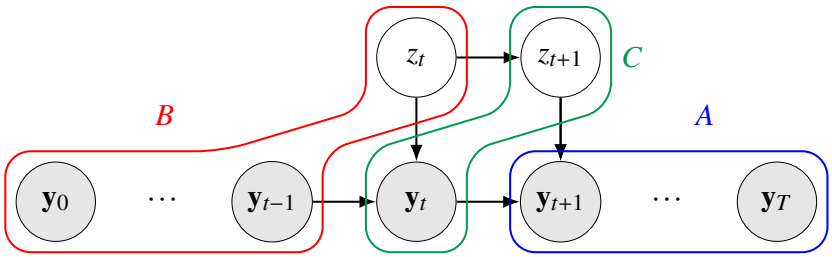
$$\beta(z_t) \stackrel{\text{def}}{=} p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T | z_t, \mathbf{y}_t, \Theta^{\text{old}}). \tag{7.59b}$$

The difference between formulation (7.59b) and (7.10b) is that in the HMM case the probability is conditioned only on the hidden mode at time t, while in the AR-HMM case it is conditioned also the the observed value at time t.

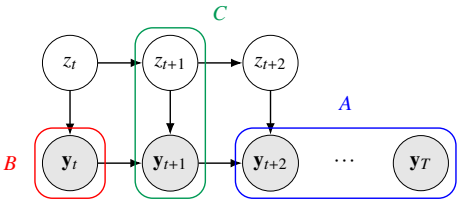
Marginal ξ can be written in terms of the forward and backward variables with a formulation similar to the standard HMM case (7.11).

Proposition 7.8. *Marginal ξ defined in* (7.56b) *can be expressed in terms of forward and backward variables α and β as in* (7.59) *as*

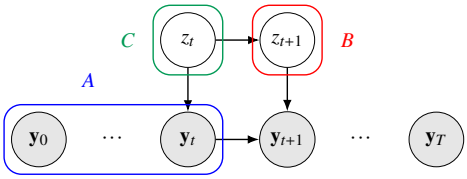
$$\xi(z_t, z_{t+1}) = \frac{\alpha(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\beta(z_{t+1})}{p(\mathbf{Y}|\Theta^{\text{old}})}. \tag{7.60}$$



(a) Conditional independence property (7.62): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.



(b) Conditional independence property (7.63): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.



(c) Conditional independence property (7.65): the arrows in the path from any element in the set A to any element of B either meet only tail-to-tail in C; or they meet in $\mathbf{y}_{t+1}$ head-to-head, and neither $\mathbf{y}_{t+1}$ nor any its descendant are in C.

Figure 7.7: Graphical proof of conditional independence properties of proof of Proposition 7.8. All the diagrams show conditional independence $A \perp\!\!\!\perp B|C$.

Proof. By applying two times the product rule of probability (E.2) we get

$$\begin{aligned}\xi(z_t, z_{t+1}) &= \frac{p(\mathbf{Y}, z_t, z_{t+1}|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{p(\mathbf{y}_0, \dots, \mathbf{y}_t, \mathbf{y}_{t+1}, \dots, \mathbf{y}_T, z_t, z_{t+1}|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}, \Theta^{\text{old}})\, p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}|\Theta^{\text{old}})}{p(\mathbf{Y}|\Theta^{\text{old}})}.\end{aligned} \tag{7.61}$$

Let us now discuss the two factors at the numerator. Starting with $p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}, \Theta^{\text{old}})$, we can apply the conditional independence property (see Figure 7.7a)

$$\mathbf{y}_{t+1}, \dots, \mathbf{y}_T \perp\!\!\!\perp \mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_t|\mathbf{y}_t, z_{t+1}, \Theta^{\text{old}}, \tag{7.62}$$

and simplify

$$p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}, \Theta^{\text{old}}) = p(\mathbf{y}_{t+1}, \mathbf{y}_{t+2}, \dots, \mathbf{y}_T|\mathbf{y}_t, z_{t+1}, \Theta^{\text{old}}).$$

Now, by product rule, we can write it as

$$p(\mathbf{y}_{t+1}, \mathbf{y}_{t+2}, \dots, \mathbf{y}_T|\mathbf{y}_t, z_{t+1}, \Theta^{\text{old}}) = p(\mathbf{y}_{t+2}, \dots, \mathbf{y}_T|\mathbf{y}_t, \mathbf{y}_{t+1}, z_{t+1}, \Theta^{\text{old}}) p(\mathbf{y}_{t+1}|\mathbf{y}_t, z_{t+1}, \Theta^{\text{old}}).$$

Next, by conditional independence property (see Figure 7.7b)

$$\mathbf{y}_{t+2}, \dots, \mathbf{y}_T \perp\!\!\!\perp \mathbf{y}_t|\mathbf{y}_{t+1}, z_{t+1}, \Theta^{\text{old}}, \tag{7.63}$$

we get

$$p(\mathbf{y}_{t+1}, \mathbf{y}_{t+2}, \dots, \mathbf{y}_T|\mathbf{y}_t, z_{t+1}, \Theta^{\text{old}}) = p(\mathbf{y}_{t+2}, \dots, \mathbf{y}_T|\mathbf{y}_{t+1}, z_{t+1}, \Theta^{\text{old}}) p(\mathbf{y}_{t+1}|\mathbf{y}_t, z_{t+1}, \Theta^{\text{old}}). \tag{7.64}$$

The first factor of (7.64) is equivalent to $\beta(z_{t+1})$, while the second factor is given by the AR dynamic of the model.

Let us now focus on the second factor at numerator in (7.61). By product rule of probability, it can be written as

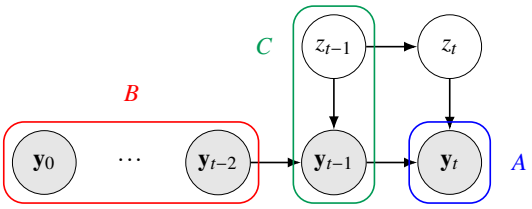
$$p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}|\Theta^{\text{old}}) = p(\mathbf{y}_0, \dots, \mathbf{y}_t|z_t, z_{t+1}, \Theta^{\text{old}})\, p(z_t, z_{t+1}|\Theta^{\text{old}}).$$

By applying the conditional independence property (see Figure 7.7c)

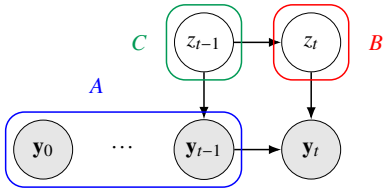
$$\mathbf{y}_0, \dots, \mathbf{y}_t \perp\!\!\!\perp z_{t+1}|z_t, \Theta^{\text{old}} \tag{7.65}$$

to the first factor, and product rule to the second, we get

$$p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}|\Theta^{\text{old}}) = p(\mathbf{y}_0, \dots, \mathbf{y}_t|z_t, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(z_t|\Theta^{\text{old}}).$$



(a) Conditional independence property (7.67): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.



(b) Conditional independence property (7.68): the arrows in the path from any element in the set A to any element of B either meet only head-to-tail in C; or they meet in $\mathbf{y}_t$ head-to-head, and neither $\mathbf{y}_t$ nor any its descendant are in C.

Figure 7.8: Graphical proof of conditional independence properties of proof of Proposition 7.9. All the diagrams show conditional independence $A \perp\!\!\!\perp B|C$.

By applying one last time the product rule of probability, we get

$$p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t, z_{t+1}|\Theta^{\text{old}}) = p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t|\Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}}).$$

The first term in the above equation is, by definition, $\alpha(z_t)$. Finally, by combining these decompositions to (7.61) we get that marginal ξ can be written as

$$\xi(z_t, z_{t+1}) = \frac{\alpha(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\beta(z_{t+1})}{p(\mathbf{Y}|\Theta^{\text{old}})}.$$

□

As for classical HMM, we now seek for recursive formulae to efficiently compute the forward and backward variables α and β (7.59). These recursion formulae are similar to the HMM case but the AR dynamic will enter both the recursion for α and β.

Proposition 7.9. *The forward variable $\alpha(z_t)$ for AR-HMM can be recursively computed as*

$$\alpha(z_t) = p(\mathbf{y}_t|z_t, \mathbf{y}_{t-1}, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \alpha(z_{t-1}) p(z_t|z_{t-1}, \Theta^{\text{old}}). \tag{7.66}$$

Proof. By the product rule of probability (E.2), we can write $\alpha(z_t)$ as

$$\begin{aligned}\alpha(z_t) &= p(\mathbf{y}_0, \dots, \mathbf{y}_t, z_t|\Theta^{\text{old}}) \\ &= p(\mathbf{y}_t|\mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_t, \Theta^{\text{old}})\, p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_t|\Theta^{\text{old}})\end{aligned}$$

We can simplify the first factor by using the conditional independence property (see Figure 7.8a)

$$\mathbf{y}_t \perp\!\!\!\perp \mathbf{y}_0, \dots, \mathbf{y}_{t-2} | \mathbf{y}_{t-1}, z_t, \Theta^{\text{old}} \tag{7.67}$$

thus getting

$$p(\mathbf{y}_t | \mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_t, \Theta^{\text{old}}) = p(\mathbf{y}_t | \mathbf{y}_{t-1}, z_t, \Theta^{\text{old}}).$$

The second factor can by written as, by marginalizing over z_{t-1}, as

$$p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_t | \Theta^{\text{old}}) = \sum_{z_{t-1} \in \mathcal{S}} p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_{t-1}, z_t | \Theta^{\text{old}}).$$

We can use again the product rule obtaining

$$\sum_{z_{t-1} \in \mathcal{S}} p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_{t-1}, z_t | \Theta^{\text{old}}) = \sum_{z_{t-1} \in \mathcal{S}} p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1} | z_{t-1}, z_t, \Theta^{\text{old}}) p(z_{t-1}, z_t | \Theta^{\text{old}}).$$

By using the product rule on the second factor, and the conditional independence property (see Figure 7.8b)

$$\mathbf{y}_0, \dots, \mathbf{y}_{t-1} \perp\!\!\!\perp z_t | z_{t-1}, \Theta^{\text{old}} \tag{7.68}$$

on the first one, we get

$$\sum_{z_{t-1} \in \mathcal{S}} p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1}, z_{t-1}, z_t | \Theta^{\text{old}}) = \sum_{z_{t-1} \in \mathcal{S}} p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1} | z_{t-1}, \Theta^{\text{old}})\, p(z_t | z_{t-1}, \Theta^{\text{old}}) p(z_{t-1} | \Theta^{\text{old}}).$$

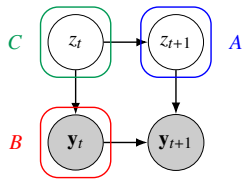
In the last r.h.s. the product $p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1} | z_{t-1}, \Theta^{\text{old}}) p(z_{t-1} | \Theta^{\text{old}})$ is equivalent to $\alpha(z_{t-1})$.
By combining these result, we obtain the recursion formula

$$\alpha(z_t) = p(\mathbf{y}_t | z_t, \mathbf{y}_{t-1}, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \alpha(z_{t-1}) p(z_t | z_{t-1}, \Theta^{\text{old}}).$$

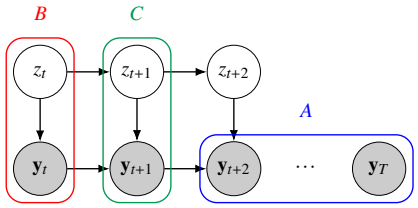
□

Proposition 7.10. *The backward variable β in* (7.59b) *can be recursively computed as*

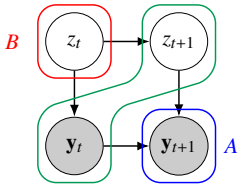
$$\beta(z_t) = \sum_{z_{t+1} \in \mathcal{S}} \beta(z_{t+1})\, p(\mathbf{y}_{t+1} | z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\, p(z_{t+1} | z_t, \Theta^{\text{old}}). \tag{7.69}$$



(a) Conditional independence property (7.70): the arrows in the path from any element in the set A to any element of B either meet only head-to-tail in C; or they meet in $\mathbf{y}_{t+1}$ head-to-head, and neither $\mathbf{y}_{t+1}$ nor any its descendant are in C.



(b) Conditional independence property (7.71): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.



(c) Conditional independence property (7.72): the arrows in the path from any element in the set A to any element of B meet only head-to-tail in C.

Figure 7.9: Graphical proof of conditional independence properties of proof of Proposition 7.10. All the diagrams show conditional independence $A \perp\!\!\!\perp B|C$.

Proof. Starting with marginalization over z_{t+1} we get

$$\begin{aligned}\beta(z_t) &\stackrel{\text{def}}{=} p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T|\mathbf{y}_t,z_t,\Theta^{\text{old}}) \\ &= \sum_{z_{t+1}\in\mathcal{S}} p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T,z_{t+1}|\mathbf{y}_t,z_t,\Theta^{\text{old}}).\end{aligned}$$

Now, let us apply the product rule of probability to get

$$\sum_{z_{t+1}\in\mathcal{S}} p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T,z_{t+1}|\mathbf{y}_t,z_t,\Theta^{\text{old}}) = \sum_{z_{t+1}\in\mathcal{S}} p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T|\mathbf{y}_t,z_t,z_{t+1},\Theta^{\text{old}})\, p(z_{t+1}|\mathbf{y}_t,z_t,\Theta^{\text{old}}).$$

Next, we can apply the conditional independence property (see Figure 7.9a)

$$z_{t+1} \perp\!\!\!\perp \mathbf{y}_t|z_t,\theta^{\text{old}} \tag{7.70}$$

to the second factor and by applying the product rule (E.2) to the first one, we get

$$\begin{aligned}&\sum_{z_{t+1}\in\mathcal{S}} p(\mathbf{y}_{t+1},\ldots,\mathbf{y}_T|\mathbf{y}_t,z_t,z_{t+1},\Theta^{\text{old}})\, p(z_{t+1}|\mathbf{y}_t,z_t,\Theta^{\text{old}}) = \\ &\qquad \sum_{z_{t+1}\in\mathcal{S}} p(\mathbf{y}_{t+2},\ldots,\mathbf{y}_T|\mathbf{y}_t,\mathbf{y}_{t+1},z_t,z_{t+1},\Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|\mathbf{y}_t,z_t,z_{t+1},\Theta^{\text{old}})p(z_{t+1}|z_t,\Theta^{\text{old}}).\end{aligned}$$

Finally, we notice that by conditional independence property (see Figure 7.9b)

$$\mathbf{y}_{t+2},\ldots,\mathbf{y}_T \perp\!\!\!\perp \mathbf{y}_t,z_t|\mathbf{y}_{t+1},z_{t+1},\Theta^{\text{old}} \tag{7.71}$$

the first factor is equivalent to $\beta(z_{t+1}) = p(\mathbf{y}_{t+2}, \dots, \mathbf{y}_T | \mathbf{y}_{t+1}, z_{t+1}, \Theta^{\text{old}})$. Moreover, by conditional independence property (Figure 7.9c)

$$\mathbf{y}_{t+1} \perp\!\!\!\perp z_t | z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}} \tag{7.72}$$

we get the recursive formula

$$\beta(z_t) = \sum_{z_{t+1} \in \mathcal{Z}} \beta(z_{t+1})\, p(\mathbf{y}_{t+1} | z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\, p(z_{t+1} | z_t, \Theta^{\text{old}}).$$

□

As for the HMM case, recursive formulae (7.66) and (7.69) often result in numerical underflow when implemented. for this reason, we will now define the *scaled* forward and backward variable for AR-HMM.
Starting with the forward variable, we define the *scaled forward variable* as in (7.27)

$$\begin{aligned} \hat{\alpha}(z_t) &\stackrel{\text{def}}{=} p(z_t | \mathbf{y}_0, \dots, \mathbf{y}_t, \Theta^{\text{old}}) \\ &= \frac{\alpha(z_t)}{p(\mathbf{y}_0, \dots, \mathbf{y}_t | \Theta^{\text{old}})}. \end{aligned} \tag{7.73}$$

By defining the *scaling factor* c_t as in (7.28)

$$c_t = p(\mathbf{y}_t | \mathbf{y}_0, \dots, \mathbf{y}_{t-1}, \Theta^{\text{old}}), \tag{7.74}$$

and recalling (7.29):

$$p(\mathbf{y}_0, \dots, \mathbf{y}_t) = \prod_{\tau=0}^{t} c_\tau,$$

we have

$$\begin{aligned} \alpha(z_t) &= p(z_t | \mathbf{y}_0, \dots, \mathbf{y}_t, \Theta^{\text{old}})\, p(\mathbf{y}_0, \dots, \mathbf{y}_{t-1} | \Theta^{\text{old}}) \\ &= \hat{\alpha}(z_t) \prod_{\tau=0}^{t} c_\tau, \end{aligned}$$

from which the recursive formula for the forward variable (7.66) can be written as

$$\underbrace{\left(\prod_{\tau=0}^{t} c_\tau \right) \hat{\alpha}(z_t)}_{\alpha(z_t)} = p(\mathbf{y}_t | z_t, \mathbf{y}_{t-1}, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \underbrace{\left(\prod_{\tau=0}^{t-1} c_\tau \right) \hat{\alpha}(z_t)}_{\alpha(z_{t-1})}\, p(z_t | z_{t-1}, \Theta^{\text{old}}).$$

By simplifying we obtain the recursive relation for the scaled forward variable

$$\hat{\alpha}(z_t) = c_t^{-1}\, p(\mathbf{y}_t | z_t, \mathbf{y}_{t-1}, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \hat{\alpha}(z_{t-1})\, p(z_t | z_{t-1}, \Theta^{\text{old}}). \tag{7.75}$$

We remark that constant c_t can be easily computed as the constant of normalization in (7.75).

Similarly, we define the *scaled backward variable* as

$$\beta(z_t) = \hat{\beta}(z_t) \prod_{\tau=t+1}^{T} c_\tau,$$

from which

$$\begin{aligned}\hat{\beta}(z_t) &= \frac{\beta(z_t)}{\prod_{\tau=t+1}^{T} c_\tau} \\ &= \frac{p(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T | \mathbf{y}_t, z_t, \Theta^{\text{old}})}{p(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T | \mathbf{y}_0, \ldots, \mathbf{y}_t, \Theta^{\text{old}})}. \end{aligned} \tag{7.76}$$

By combining (7.76) and (7.69) we get

$$\underbrace{\left(\prod_{\tau=t+1}^{T} c_\tau\right)\hat{\beta}(z_t)}_{\beta(z_t)} = \sum_{z_{t+1}\in\mathcal{S}} \underbrace{\left(\prod_{\tau=t+2}^{T} c_\tau\right)\hat{\beta}(z_{t+1})}_{\beta(z_{t+1})} p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}}).$$

By simplifying, we get the *recursive relation for the backward variables* in the case of AR-HMM

$$\hat{\beta}(z_t) = c_{t+1}^{-1} \sum_{z_{t+1}\in\mathcal{S}} \hat{\beta}(z_{t+1})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}}). \tag{7.77}$$

We now have to express marginals γ and ξ in terms of scaled forward and backward variables.
From (7.58) we have

$$\begin{aligned}\gamma(z_t) &= \frac{\alpha(z_t)\beta(z_t)}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{\alpha(z_t)\beta(z_t)}{\prod_{\tau=0}^{T} c_\tau} \\ &= \frac{\alpha(z_t)\beta(z_t)}{\left(\prod_{\tau=0}^{t} c_\tau\right)\left(\prod_{\tau=t+1}^{T} c_\tau\right)} \\ &= \hat{\alpha}(z_t)\hat{\beta}(z_t). \end{aligned} \tag{7.78}$$

Similarly, from (7.60), we obtain

$$\begin{aligned}\xi(z_t, z_{t+1}) &= \frac{\alpha(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\beta(z_{t+1})}{p(\mathbf{Y}|\Theta^{\text{old}})} \\ &= \frac{\alpha(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\beta(z_{t+1})}{\left(\prod_{\tau=0}^{t} c_\tau\right) c_{t+1} \left(\prod_{\tau=t+2}^{T} c_\tau\right)} \end{aligned}$$

$$= c_{t+1}^{-1}\, \hat{\alpha}(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}}) \hat{\beta}(z_{t+1}) \tag{7.79}$$

We now discuss the basis of recursion for (7.66) adn (7.69). Let us start with the forward scaled variable. By applying the Bayes' theorem (E.3) to definition (7.73) we have

$$\begin{aligned}\hat{\alpha}(z_1) &= p(z_1|\mathbf{y}_0, \mathbf{y}_1, \Theta^{\text{old}}) \\ &= \frac{p(\mathbf{y}_1|\mathbf{y}_0, z_1, \Theta^{\text{old}})\, p(z_1|\mathbf{y}_0, \Theta^{\text{old}})}{p(\mathbf{y}_1|\mathbf{y}_0, \Theta^{\text{old}})}.\end{aligned}$$

By using conditional independence property $z_1 \perp\!\!\!\perp \mathbf{y}_0|\Theta^{\text{old}}$, and observing that $p(\mathbf{y}_1|\mathbf{y}_0, \Theta^{\text{old}}) = c_1$ we obtain

$$\begin{aligned}\hat{\alpha}(z_1) &= \frac{p(\mathbf{y}_1|\mathbf{y}_0, z_1, \Theta^{\text{old}})\, p(z_1|\Theta^{\text{old}})}{c_1} \\ &\propto p(\mathbf{y}_1|\mathbf{y}_0, z_1, \Theta^{\text{old}})\, p(z_1|\Theta^{\text{old}})\end{aligned} \tag{7.80}$$

From an implementative point of view, $\hat{\alpha}(z_1)$ can be computed as the right hand side of (7.80) and sucessively normalized, obtaining c_1 as the constant of normalization.

The basis of recursion for the scaled backward variable $\hat{\beta}$ follows from identy (7.78). By definition (7.56a)

$$\gamma(z_T) \stackrel{\text{def}}{=} p(z_T|\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T, \Theta^{\text{old}}).$$

Moreover, by definition (7.73)

$$\hat{\alpha}(z_T) \stackrel{\text{def}}{=} p(z_T|\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T, \Theta^{\text{old}}),$$

follows the identity

$$\gamma(z_T) = \hat{\alpha}(z_T).$$

By combining this result with identity (7.78) $\gamma(z_t) = \hat{\alpha}(z_t)\hat{\beta}(z_t)$ for any $t = 1, 2, \ldots, T$ we obtain that the basis of recursion for $\hat{\beta}$ is

$$\hat{\beta}(z_T) = 1. \tag{7.81}$$

The process of computing the scaled forward and backward variables takes the name of *forward-backward algorithm* and is summarized in Algorithm 7.4.

In summary, the Expectation step reduces to the application of the forward-backward algorithm to compute variables $\hat{\alpha}$ and $\hat{\beta}$, followed by the computation of marginals γ and ξ, as summarized in Algorithm 7.5

Algorithm 7.4 Forward-Backward Algorithm for AR-HMM.

Input: Model parameters Θ^{old}, stream of observations $\mathbf{Y} = (\mathbf{y}_0, \ldots, \mathbf{y}_T)$.

Output: Scaled forward and backward variables $\hat{\alpha}(z_t)$ and $\hat{\beta}(z_t)$ and scaling factors c_t.

▹ Compute the forward variables and the scaling factors.

1: Initialize the forward variable using (7.80):

$$\hat{\alpha}(z_1) \propto p(\mathbf{y}_1|\mathbf{y}_0, z_1, \Theta^{\text{old}})\, p(z_1|\Theta^{\text{old}}).$$

2: Compute c_1 as the normalizing factor.

3: **for** $t = 2, 3, \ldots, T$ **do**

4: Compute the forward variable using (7.75):

$$\hat{\alpha}(z_t) \propto p(\mathbf{y}_t|z_t, \mathbf{y}_{t-1}, \Theta^{\text{old}}) \sum_{z_{t-1} \in \mathcal{S}} \hat{\alpha}(z_{t-1}) p(z_t|z_{t-1}, \Theta^{\text{old}}).$$

5: Compute c_t as the normalizing faactor.

6: **end for**

▹ Compute the backward variables.

7: Initialize the scaled backward variable using (7.81):

$$\hat{\beta}(z_T) = 1.$$

8: **for** $t = T-1, T-2, \ldots, 1$ **do**

9: Compute the scaled backward vairbales using (7.77):

$$\hat{\beta}(z_t) = c_{t+1}^{-1} \sum_{z_{t+1} \in \mathcal{S}} \hat{\beta}(z_{t+1})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}}).$$

10: **end for**

Maximization Step

In the M-step, we aim at computing the new set of parameters Θ that maximizes $\widetilde{Q}(\Theta, \Theta^{\text{old}})$ in (7.55). To do so, we discuss maximization over initial state probability $\boldsymbol{\varpi}$, transition matrix $\mathbf{T}$, and the AR dynamic parameters separately. This can be done since these terms appear in different summations in (7.55) and maximization of $\widetilde{Q}$ can be achieved by maximizing each of the three components independently.

Starting with the *initial state probability*, we have to solve the following maxi-

Algorithm 7.5 Expectation Step for AR-HMM.

Input: Set of parameters Θ^{old}, stream of observations $\mathbf{Y}$.

Output: Marginals $\gamma(z_t) \stackrel{\text{def}}{=} p(z_t|\mathbf{Y}, \Theta^{\text{old}})$ and $\xi(z_t, z_{t+1}) \stackrel{\text{def}}{=} p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}})$.

1: Apply the forward backward algorithm 7.4 to compute the forward and scaled variables:

$$\hat{\alpha}(z_t), \hat{\beta}(z_t), c_t = \texttt{forward\_backward}\ (\Theta^{\text{old}}, \mathbf{Y}).$$

▷ Compute the marginals.

2: **for** $t = 1, 2, \ldots, T-1$ **do**

3: Compute marginal γ using (7.78):

$$\gamma(z_t) = \hat{\alpha}(z_t)\hat{\beta}(z_t).$$

4: Compute marginal ξ using (7.79):

$$\xi(z_t, z_{t+1}) = c_{t+1}^{-1} \hat{\alpha}(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}}) \hat{\beta}(z_{t+1})$$

5: **end for**

6: $\gamma(z_T) = \hat{\alpha}(z_T)$.

mization problem:

$$\begin{aligned} \boldsymbol{\varpi}^\star &= \arg\max_{\boldsymbol{\varpi}} \sum_{i \in \mathcal{S}} \Big(\log \varpi_i \Pr(z_1 = i|\mathbf{Y}, \Theta^{\text{old}}) \Big), \\ \text{s.t.} \quad & \sum_{i \in \mathcal{S}} \varpi_i - 1 = 0. \end{aligned} \tag{7.82}$$

Proposition 7.11. *Problem* (7.82) *is solved by setting*

$$\boldsymbol{\varpi}|_i = Pr(z_1 = i|\mathbf{Y}, \Theta^{\text{old}}). \tag{7.83}$$

Proof. The proof follows the same steps of the proof of Proposition 7.4. The only difference lies in the fact that subscript $_0$ has to be substituted with subscript $_1$. □

Talking about *transition probability*, we want to solve the maximization problem:

$$\begin{aligned} \mathbf{T}^\star &= \arg\max_{\mathbf{T}} \sum_{t=1}^{T-1} \sum_{i=1}^{S} \sum_{j=1}^{S} \log \mathbf{T}|_{ij} \Pr(z_t = i, z_{t+1} = j|\mathbf{Y}, \Theta^{\text{old}}), \\ \text{s.t.} \quad & \sum_{j} \mathbf{T}|_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, S\}. \end{aligned} \tag{7.84}$$

Proposition 7.12. *Problem* (7.84) *is solved by setting*

$$\mathbf{T}|_{k\ell} = \frac{\sum_{t=1}^{T-1} \Pr(z_t = k, z_{t+1} = \ell|\mathbf{Y}, \Theta^{\text{old}})}{\sum_{t=1}^{T-1} \Pr(z_t = k|\mathbf{Y}, \Theta^{\text{old}})}. \tag{7.85}$$

Proof. The proof follows the same steps as proof of Proposition 7.5. The only difference is that summation over t spans from 1 to $T-1$ instead of from 0 and $T-1$. □

Proposition 7.13. *Maximization over AR dynamic parameters $\mathbf{A}_s, \mathbf{b}_s, \mathbf{\Sigma}_s,\ s \in \mathcal{S}$ is achieved by setting*

$$[\mathbf{A}_s, \mathbf{b}_s] = \left(\sum_{t=0}^{T-1} \Pr(z_t = s|\mathbf{Y}, \Theta^{\text{old}})\, \mathbf{y}_{t+1} \hat{\mathbf{y}}_t^\intercal \right) \left(\sum_{t=0}^{T-1} \Pr(z_t = s|\mathbf{Y}, \Theta^{\text{old}})\, \hat{\mathbf{y}}_t \hat{\mathbf{y}}_t^\intercal \right)^{-1} \tag{7.86a}$$

$$\mathbf{\Sigma}_s = \frac{\sum_{t=0}^{T-1} \Pr(z_t = s|\mathbf{Y}, \Theta^{\text{old}})\, (\mathbf{y}_{t+1} - (\mathbf{A}_s \mathbf{y}_t + \mathbf{b}_s))\, (\mathbf{y}_{t+1} - (\mathbf{A}_s \mathbf{y}_t + \mathbf{b}_s))^\intercal}{\sum_{t=0}^{T-1} \Pr(z_t = s|\mathbf{Y}, \Theta^{\text{old}})} \tag{7.86b}$$

where $\hat{\mathbf{y}}_t$ is defined as $\hat{\mathbf{y}}_t = [\mathbf{y}_t^\intercal, 1]^\intercal$

Proof. We aim at maximize, w.r.t. the linear dynamic's parameters and the covariance matrices,

$$\begin{aligned} \widetilde{Q}_{\text{out}}(\Theta, \Theta^{\text{old}}) &= \sum_{t=0}^{T-1} \sum_{z_{t+1} \in \mathcal{S}} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta) p(z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) \\ &= \sum_{t=0}^{T-1} \sum_{s=1}^{S} \gamma_s(t+1) \log p(\mathbf{y}_{t+1}|z_{t+1} = s, \mathbf{y}_t, \Theta). \end{aligned} \tag{7.87}$$

Now, we have that the next state probability is a Gaussian distribution

$$p(\mathbf{y}_{t+1}|z_{t+1} = i, \mathbf{y}_t, \Theta) = \mathcal{N}(\mathbf{y}_{t+1}|\mathbf{A}_s \mathbf{y}_t + \mathbf{b}_s, \mathbf{\Sigma}_s).$$

The linear dynamic can be expressed in a more compact way as

$$\mathbf{A}_s \mathbf{y}_t + \mathbf{b}_s = \widetilde{\mathbf{A}}_s \hat{\mathbf{y}}_t,$$

where we defined

$$\begin{aligned} \widetilde{\mathbf{A}}_s &\stackrel{\text{def}}{=} [\mathbf{A}_s, \mathbf{b}_s], \\ \hat{\mathbf{y}}_t &= \left[\mathbf{y}_t^\intercal, 1\right]^\intercal, \end{aligned}$$

thus getting

$$p(\mathbf{y}_{t+1}|z_{t+1} = i, \mathbf{y}_t, \Theta) = \mathcal{N}(\mathbf{y}_{t+1}|\widetilde{\mathbf{A}}_s \hat{\mathbf{y}}_t, \mathbf{\Sigma}_s). \tag{7.88}$$

With these definitions, we can rewrite (7.87) as

$$\begin{aligned}
\widetilde{Q}_{\text{out}}(\Theta,\Theta^{\text{old}}) &= \sum_{t=0}^{T-1}\sum_{s=1}^{S}\gamma_s(t+1)\log\left(\frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_s|^{1/2}}\exp\left(-\frac{1}{2}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}_s\hat{\mathbf{y}}_t\right)^{\intercal}\boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}_s\hat{\mathbf{y}}_t\right)\right)\right)\\
&= \sum_{t=0}^{T-1}\sum_{s=1}^{S}\gamma_s(t+1)\left(-\frac{d}{2}\log(2\pi)\right)+\\
&\quad \sum_{t=0}^{T-1}\sum_{s=1}^{S}\gamma_s(t+1)\left(-\frac{1}{2}\log|\boldsymbol{\Sigma}_s|\right)+\\
&\quad \sum_{t=0}^{T-1}\sum_{s=1}^{S}\gamma_s(t+1)\left(-\frac{1}{2}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}_s\hat{\mathbf{y}}_t\right)^{\intercal}\boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}_s\hat{\mathbf{y}}_t\right)\right).
\end{aligned}$$

Since the first term does not depend on the parameters set Θ we obtain that we have to maximize

$$\widetilde{\widetilde{Q}}(\Theta,\Theta^{\text{old}}) = \sum_{t=0}^{T-1}\sum_{s=1}^{S}\gamma_s(t+1)\left(-\frac{1}{2}\log|\boldsymbol{\Sigma}_s|\right)-\frac{1}{2}\sum_{t=0}^{T-1}\sum_{s=1}^{S}\gamma_s(t+1)\left(\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}_s\hat{\mathbf{y}}_t\right)^{\intercal}\boldsymbol{\Sigma}_s^{-1}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}_s\hat{\mathbf{y}}_t\right)\right). \tag{7.89}$$

Maximization can be performed independently over each mode $s \in \{1,2,\ldots,S\}$, which allows us to simplify the notation. By dropping the dependence on index s we aim at maximize

$$\widehat{Q}(\Theta,\Theta^{\text{old}}) = \sum_{t=0}^{T-1}\gamma(t+1)\left(-\frac{1}{2}\log|\boldsymbol{\Sigma}|\right)-\frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\left(\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)^{\intercal}\boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)\right). \tag{7.90}$$

Maximization over the linear dynamic matrix $\widetilde{\mathbf{A}}$ is achived by setting $\partial_{\widetilde{\mathbf{A}}}\widehat{Q} = \mathbf{0}$. Let us compute

$$\begin{aligned}
\frac{\partial\widehat{Q}(\Theta,\Theta^{\text{old}})}{\partial\widetilde{\mathbf{A}}} &= -\frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\frac{\partial}{\partial\widetilde{\mathbf{A}}}\left(\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)^{\intercal}\boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)\right)\\
&= -\frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\left(-2\boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)\hat{\mathbf{y}}_t^{\intercal}\right)\\
&= \boldsymbol{\Sigma}^{-1}\sum_{t=0}^{T-1}\gamma(t+1)\left(\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)\hat{\mathbf{y}}_t^{\intercal}\right).
\end{aligned}$$

By setting it to zero, we obtain

$$\sum_{t=0}^{T-1}\gamma(t+1)\left(\left(\mathbf{y}_{t+1}-\widetilde{\mathbf{A}}\hat{\mathbf{y}}_t\right)\hat{\mathbf{y}}_t^{\intercal}\right) = \mathbf{0}$$

$$\Leftrightarrow \quad \sum_{t=0}^{T-1} \gamma(t+1)\, \mathbf{y}_{t+1}\, \hat{\mathbf{y}}_t^\intercal = \sum_{t=0}^{T-1} \gamma(t+1)\, \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t\, \hat{\mathbf{y}}_t^\intercal$$
$$\Leftrightarrow \quad \sum_{t=0}^{T-1} \gamma(t+1)\, \mathbf{y}_{t+1}\, \hat{\mathbf{y}}_t^\intercal = \widetilde{\mathbf{A}} \sum_{t=0}^{T-1} \gamma(t+1)\, \hat{\mathbf{y}}_t\, \hat{\mathbf{y}}_t^\intercal$$

from which [20]

$$\widetilde{\mathbf{A}} = \left(\sum_{t=0}^{T-1} \gamma(t+1)\, \mathbf{y}_{t+1}\, \hat{\mathbf{y}}_t^\intercal \right) \left(\sum_{t=0}^{T-1} \gamma(t+1)\, \hat{\mathbf{y}}_t\, \hat{\mathbf{y}}_t^\intercal \right)^{-1},$$

which, by ripristinating the index s that we dropped before for notation simplicity, gives (7.86a).

Next, we discuss maximization of (7.90) w.r.t. covariance matrix $\mathbf{\Sigma}$. To do so, let us compute the derivative

$$\frac{\partial \widehat{Q}(\Theta, \Theta^{\text{old}})}{\partial \mathbf{\Sigma}^{-1}} = -\frac{1}{2} \sum_{t=0}^{T-1} \gamma(t+1) \frac{\partial}{\partial \mathbf{\Sigma}^{-1}} (\log |\mathbf{\Sigma}|) - \frac{1}{2} \sum_{t=0}^{T-1} \gamma(t+1) \frac{\partial}{\partial \mathbf{\Sigma}^{-1}} \left(\left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right)^\intercal \mathbf{\Sigma}^{-1} \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right) \right). \tag{7.91}$$

Let us discuss the two terms. The first summation becomes

$$\begin{aligned} -\frac{1}{2} \sum_{t=0}^{T-1} \gamma(t+1) \frac{\partial}{\partial \mathbf{\Sigma}^{-1}} (\log |\mathbf{\Sigma}|) &= \frac{1}{2} \frac{\partial}{\partial \mathbf{\Sigma}^{-1}} \left(\log |\mathbf{\Sigma}|^{-1} \right) \sum_{t=0}^{T-1} \gamma(t+1) \\ &= \frac{1}{2} \frac{\partial}{\partial \mathbf{\Sigma}^{-1}} \left(\log |\mathbf{\Sigma}^{-1}| \right) \sum_{t=0}^{T-1} \gamma(t+1) \\ &= \frac{1}{2} \mathbf{\Sigma} \sum_{t=0}^{T-1} \gamma(t+1). \end{aligned}$$

The second term in (7.91) reads

$$\frac{1}{2} \sum_{t=0}^{T-1} \gamma(t+1) \frac{\partial}{\partial \mathbf{\Sigma}^{-1}} \left(\left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right)^\intercal \mathbf{\Sigma}^{-1} \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right) \right) = \frac{1}{2} \sum_{t=0}^{T-1} \gamma(t+1) \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right) \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right)^\intercal.$$

Combining these results, (7.91) reads

$$\frac{\partial \widehat{Q}(\Theta, \Theta^{\text{old}})}{\partial \mathbf{\Sigma}^{-1}} = \frac{1}{2} \mathbf{\Sigma} \sum_{t=0}^{T-1} \gamma(t+1) - \frac{1}{2} \sum_{t=0}^{T-1} \gamma(t+1) \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right) \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right)^\intercal.$$

By setting this quantity to zero, we get

$$\mathbf{\Sigma} = \frac{\sum_{t=0}^{T-1} \gamma(t+1) \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right) \left(\mathbf{y}_{t+1} - \widetilde{\mathbf{A}}\, \hat{\mathbf{y}}_t \right)^\intercal}{\sum_{t=0}^{T-1} \gamma(t+1)},$$

which, by ripristinating the index s that we dropped before for notation simplicity, gives (7.86b) □

Algorithm 7.6 Maximization Step for AR-HMM.

Input: Marginals $\gamma_i(t)$, $i \in \mathcal{S}$, $t \in \{1,2,T\}$ and $\xi_{ij}(t)$, $i,j \in \mathcal{S}$, $t \in 1,2,\dots,T-1$; observed data $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_T)$, $\mathbf{y}_t \in \mathbb{R}^d$, $t = 0,1,\dots,T$

Output: New set of parameter $\Theta = \{\varpi, \mathbf{T}, \mathbf{A}_s, \mathbf{b}_s, s \in \mathcal{S}\}$

▷ Maximization over the initial mode probability density

1: Set $\varpi|_i = \gamma_i(1)$.

▷ Maximization over the transition probability matrix

2: Set $\mathbf{T}|_{k\ell} = \frac{\sum_{t=1}^{T-1} \xi_{k\ell}(t)}{\sum_{t=1}^{T-1} \gamma_k(t)}$.

▷ Maximization over the AR dynamic parameters

3: **for** $s \in \mathcal{S} = \{1,2,\dots,S\}$ **do**

4: Set

$$\mathbf{\Sigma}_s = \frac{\sum_{t=0}^{T-1} \gamma_s(t+1)\,(\mathbf{y}_{t+1} - (\mathbf{A}_s\mathbf{y}_t + \mathbf{b}_s))(\mathbf{y}_{t+1} - (\mathbf{A}_s\mathbf{y}_t + \mathbf{b}_s))^\intercal}{\sum_{t=0}^{T-1} \gamma_s(t+1)}.$$

5: Set, where $\hat{\mathbf{y}} = [\mathbf{y}^\intercal, 1]^\intercal$,

$$\widetilde{\mathbf{A}}_s = \left(\sum_{t=0}^{T-1} \gamma_s(t+1)\,\mathbf{y}_{t+1}\hat{\mathbf{y}}_t^\intercal\right)\left(\sum_{t=0}^{T-1} \gamma_s(t+1)\,\hat{\mathbf{y}}_t\hat{\mathbf{y}}_t^\intercal\right)^{-1},$$

6: Set, for $k = 1,2,\dots,d$ and $\ell = 1,2,\dots,d$, $\mathbf{A}_s|_{k\ell} = \widetilde{\mathbf{A}}_s|_{k\ell}$ and $\mathbf{b}_s|_k = \widetilde{\mathbf{A}}_s|_{kd+1}$.

7: **end for**

We remark that, since the values for the marginal γ depend on the old set of parameters Θ^{old}, the AR dynamic's parameters $\mathbf{A}_s, \mathbf{b}_s$ in (7.86b) are the 'old' values, and not those given from (7.86a). In practice, when implementing the maximization step, it is convenient to update the covariance matrices first, and then update the AR dynamic parameters.

The M-step is summarized in Algorithm 7.6.

7.2.2. EM Algorithm for Multiple Observations

The EM algorithm development presented in Section 7.2.1 assumes that a single stream of observation $\mathbf{Y}$ is available to perform the algorithm. Let us assume that a set of sequences of observations is given:

$$\Upsilon = \{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \dots, \mathbf{Y}^{(K)}\}.$$

By assuming that each observation is independent to all other demonstrations given the set of parameters:

$$\mathbf{Y}^{(k)} \perp\!\!\!\perp \Upsilon \setminus \{\mathbf{Y}^{(k)}\} \,|\, \Theta,$$

we have that the maximization of the likelihood $p(\Upsilon|\Theta)$ can be achieved by maximizing the auxilary function [79]

$$\breve{Q}(\Theta, \Theta^{\text{old}}) = \sum_{k=1}^{K} \widetilde{Q}^{(k)}(\Theta, \Theta^{\text{old}}), \tag{7.92}$$

where $\widetilde{Q}^{(k)}$ is obtained from (7.55) by adding the superscrit $^{(k)}$ to $\mathbf{Y}$. By expanding (7.92), we get

$$\begin{aligned}\breve{Q}(\Theta, \Theta^{\text{old}}) = &\sum_{k=1}^{K} \sum_{z_1 \in \mathcal{S}} \log p(z_1|\Theta) p\left(z_1|\mathbf{Y}^{(k)}, \Theta^{\text{old}}\right) + \\ &\sum_{k=1}^{K} \sum_{t=0}^{T^{(k)}-1} \sum_{z_{t+1} \in \mathcal{S}} \log p\left(\mathbf{y}_{t+1}^{(k)}|z_{t+1}, \mathbf{y}_t^{(k)}, \Theta\right) p\left(z_{t+1}|\mathbf{Y}^{(k)}, \Theta^{\text{old}}\right) + \\ &\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}-1} \sum_{z_t \in \mathcal{S}} \sum_{z_{t+1} \in \mathcal{S}} \log p(z_{t+1}|z_t, \Theta) p\left(z_t, z_{t+1}|\mathbf{Y}^{(k)}, \Theta^{\text{old}}\right).\end{aligned}$$

We remark that each demostration $\mathbf{Y}^{(k)}$ in Υ may have different length w.r.t. all other demonstrations. Thus, each demonstration $\mathbf{Y}^{(k)}$ span over time indexes $0, 1, \ldots, T^{(k)}$.

The expectation step follows as presented in Section 7.2.1. The only difference is that a set of marginals $\gamma^{(k)}(z_t)$ and $\xi^{(k)}(z_t, z_{t+1})$ (and, subsequently, of forward and backward variables $\alpha^{(k)}(z_t)$ and $\beta^{(k)}(z_t)$) have to be computed for each demostration $\mathbf{Y}^{(k)}$ in Υ.

Maximization over the parameters is easy to extend. Indeed, since the summation is over the index of demonstration k, we can move the differentiations over the parameters inside the summation itself when solving the maximization problems (see proofs of Proposition 7.11, Proposition 7.12, and Proposition 7.13).
Maximization of the initial mode probability is achieved by setting

$$\boldsymbol{\varpi}|_i = \frac{1}{K} \sum_{k=1}^{K} Pr(z_1 = i|\mathbf{Y}^{(k)}, \Theta^{\text{old}}). \tag{7.93a}$$

Maximization over the transition probability matrix is obtained by setting

$$\mathbf{T}|_{mn} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}-1} \Pr(z_t = m, z_{t+1} = n|\mathbf{Y}^{(k)}, \Theta^{\text{old}})}{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}-1} \Pr(z_t = m|\mathbf{Y}^{(k)}, \Theta^{\text{old}})}. \tag{7.93b}$$

Finally, maximization over the linear dynamic parameters

$$[\mathbf{A}_s, \mathbf{b}_s] = \left(\sum_{k=1}^{K} \sum_{t=0}^{T^{(k)}-1} \Pr(z_t = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}}) \mathbf{y}_{t+1}^{(k)} \hat{\mathbf{y}}_t^{(k)\intercal} \right) \left(\sum_{k=1}^{K} \sum_{t=0}^{T^{(k)}-1} \Pr(z_t = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}}) \hat{\mathbf{y}}_t^{(k)} \hat{\mathbf{y}}_t^{(k)\intercal} \right)^{-1}, \tag{7.93c}$$

$$\mathbf{\Sigma}_s = \frac{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1} \Pr(z_t = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}})\left(\mathbf{y}_{t+1}^{(k)} - \left(\mathbf{A}_s\mathbf{y}_t^{(k)} + \mathbf{b}_s\right)\right)\left(\mathbf{y}_{t+1}^{(k)} - \left(\mathbf{A}_s\mathbf{y}_t^{(k)} + \mathbf{b}_s\right)\right)^{\intercal}}{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1} \Pr(z_t = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}})}. \tag{7.93d}$$

7.2.3. Parameters Initialization

As pointed out in 7.1.3 for the HMM case, the EM algorithm converges to a local maximum in the parameter space. This is true also for the AR-HMM. Thus, proper initialization is required.

Similarly to HMM, the process is based on using k-means clustering to obtain a first segmentation of the data-set $\mathbf{\Upsilon} = \{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(K)}\}$ to use to initialize the model parameters Θ.

The first difference from the classical HMM model is the definition on which data perform the clustering. Indeed, in HMM makes sense to perform clustering directly on data $\mathbf{y}_t^{(k)}$ since each hidden mode directly emits an observed state. On the other hand, in AR-HMM each hidden mode describes the vector field mapping the previous state into the present one. For this reason, clustering results more efficient when it is performed on the 'velocity' dataset

$$\widetilde{\mathbf{Y}}^{(k)} = (\mathbf{y}_t^{(k)} - \mathbf{y}_{t-1}^{(k)})_{t=1,2,\ldots T}, \tag{7.94a}$$

or on datasets comprehending both positions and velocities

$$\widetilde{\mathbf{Y}}^{(k)} = \left(\begin{bmatrix} \mathbf{y}_t^{(k)} \\ \mathbf{y}_t^{(k)} - \mathbf{y}_{t-1}^{(k)} \end{bmatrix}\right)_{t=1,2,\ldots T}. \tag{7.94b}$$

Once the clustering has been performed, the update of the initial mode probability and transition probability is performed as in the classical HMM case as in (7.50) and (7.51).

The linear dynamic parameters $\mathbf{A}_s$ and $\mathbf{b}_s$, and the covariance matrices of the error $\mathbf{\Sigma}_s$ are computed by performing linear regression similarly to the M-step. In particular, the initialization is performed as in (7.93) where probability $\Pr(z_t = s|\widetilde{\mathbf{Y}}^{(k)}, \Theta^{\text{old}})$ is a Dirac delta distribution. It has value one if the clustering assign mode s to z_t, and zero otherwise.

7.2.4. Viterbi Algorithm

In this Section, we present the Viterbi algorithm for AR-HMM. The basic idea follows the same argument of Section 7.1.4.

Once the AR-HMM's set of parameters Θ have been learned, we aim at using the model to determine the sequence of modes $\mathbf{z} = (z_1, \ldots, z_T)$ that most likely generated the observed sequence $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T)$. We thus aim at computing

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z} \in \mathcal{S}^T} p(\mathbf{Y}, \mathbf{z}),$$

or, equivalently,

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z} \in \mathcal{S}^T} \log p(\mathbf{Y}, \mathbf{z}).$$

We now define the variables [9]

$$\omega(z_t) \stackrel{\text{def}}{=} \max_{(z_1, \ldots, z_{t-1}) \in \mathcal{S}^{t-1}} \log p((\mathbf{y}_0, \ldots, \mathbf{y}_t), (z_1, \ldots, z_t)). \tag{7.95}$$

Function ω can be initialized as

$$\omega(z_1) = \log p(z_1) + \log p(\mathbf{y}_1 | z_1, \mathbf{y}_0),$$

and recursively computed as

$$\omega(z_{t+1}) = \log p(\mathbf{y}_{t+1} | z_{t+1}, \mathbf{y}_t) + \max_{z_t \in \mathcal{S}} \left(\log p(z_{t+1} | z_t) + \omega(z_t) \right).$$

From definition (7.95) we have that $\log p(\mathbf{Y}, \hat{\mathbf{z}}) = \arg\max_{z_T \in \mathcal{S}} \omega(z_T)$.

Now, we have to find the sequence of latent variable values that correspond to the path maximizing $p(\mathbf{Y}, \mathbf{z})$. To do so, we can apply a simple back-tracking procedure. In particular, at each time t, maximization over z_t must be performed for each of the possible values of $z_{t+1} \in \mathcal{S}$. Suppose we keep a record of the values of z_t that correspond to the maxima for each possible value of z_{t+1}, and denote this function $\psi(s_t)$. Once we found the most probable mode for z_T, we can use this function to backtrack along the chain by applying recursively $\hat{s}_t = \psi(\hat{s}_{t+1})$.
Similarly to the HMM case, the algorithm generates two 2-dimensional tables $\mathtt{T}_1, \mathtt{T}_2$ of size $T \times S$. Element $\mathtt{T}_1|_{i\,j}$ of $\mathtt{T}_1$ contains the probability of the most likely path so far $\hat{\mathbf{z}} = (\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_i)$ with $\hat{z}_i = j$ that generates $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_2, \ldots, \mathbf{y}_i)$. Element $\mathtt{T}_2|_{i\,j}$ of $\mathtt{T}_2$ contains the mode $\hat{z}_{i-1}$ of the most likely path so far $\hat{\mathbf{z}} = (\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_{i-1}, \hat{z}_i = j)$. The components of $\mathtt{T}_1, \mathtt{T}_2$ are recursively computed as

$$\mathtt{T}_1|_{i\,j} = \max_k \left(\mathtt{T}_{i-1,k} \Pr(z_{t+1} = j | z_t = k)\, p(\mathbf{y}_t | z_t = s, \mathbf{y}_{t-1}) \right),$$

$$\begin{aligned} \mathtt{T}_2|_{i\,j} &= \arg\max_k \left(\mathtt{T}_{i-1,k} \Pr(z_{t+1} = j | z_t = k)\, p(\mathbf{y}_t | z_t = s, \mathbf{y}_{t-1}) \right) \\ &= \arg\max_k \left(\mathtt{T}_{i-1,k} \Pr(z_{t+1} = j | z_t = k) \right). \end{aligned}$$

We remark that last identity follows from the fact that $\Pr(\mathbf{y}_j | z_j = i)$ is positive and independent of index k. The basis of recursion is

$$\mathtt{T}_1|_{1\,s} = \Pr(z_1 = s) \Pr(\mathbf{y}_1 | z_1 = s, \mathbf{y}_0),$$

$$\mathtt{T}_2|_{1\,s} = 0.$$

The algorithm is summarized in Algorithm 7.7

Algorithm 7.7 Viterbi Algorithm - AR-HMM

Input: Model parameters Θ, observation sequence $\mathbf{Y} = (\mathbf{y}_0, \dots, \mathbf{y}_T)$.

Output: Most likely sequence of hidden modes $\hat{\mathbf{z}} = (\hat{z}_1, \hat{z}_2, \dots, \hat{z}_T)$.

for $s = 1,,2 \dots S$ **do**
  $\mathtt{T}_1|_{1\,s} = \Pr(z_1 = s)\, \Pr(\mathbf{y}_1 | z_1 = s, \mathbf{y}_0)$
  $\mathtt{T}_2|_{1\,s} = 0$
end for
for $t = 2, 3, \dots, T$ **do**
  for do $s = 1, 2, \dots, S$
    $\mathtt{T}_1|_{s\,t} = \max_k \left(\mathtt{T}_1|_{k\,t-1}\, \Pr(z_t = s | z_{t-1} = k, \mathbf{y}_{t-1})\, p(\mathbf{y}_t | z_t = s)\right)$
    $\mathtt{T}_2|_{s\,t} = \arg\max_k \left(\mathtt{T}_1|_{k\,t-1}\, \Pr(z_t = s | z_{t-1} = k, \mathbf{y}_{t-1})\, p(\mathbf{y}_t | z_t = s)\right)$
  end for
end for
$\hat{z}_T = \arg\max_k \mathtt{T}_1|_{k\,T}$
for $t = T, T-1, \dots, 1$ **do**
  $\hat{z}_{t-1} = \mathtt{T}_2|_{\hat{z}_t, t}$
end for

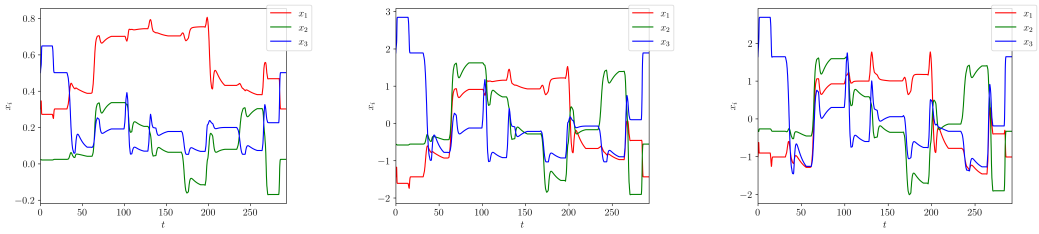
7.2.5. Data Pre-Processing

Before applying the EM and Viterbi algorithm to the dataset, a pre-processing step is necessary. This step allows a more numerically stable EM procedure. The goal of this pre-processing step, called *standardization* [112], is to obtain signals of null mean and unit variance. Let $\mathbf{Y} \in \mathbb{R}^{(T+1)\times d}$ be the original data set, so that the $t-$th row of $\mathbf{Y}$ is $\mathbf{y}(t) = [y_p(t)]_{p\in\{1,2,\dots,d\}} \in \mathbb{R}^d$. At first, we 'move' the dataset to be at zero mean. To do so, we define a new dataset $\widehat{\mathbf{Y}} = [\hat{y}_p(t)]_{t\in\{0,1,\dots,T\}, p\in\{1,2,\dots,d\}}$ with

$$\begin{aligned}\hat{y}_p(t) &\doteq y_p(t) - \mu_p \\ &= y_p(t) - \frac{1}{T+1}\sum_{\tau=0}^{T-1} y_p(\tau).\end{aligned} \tag{7.96}$$

In this way, the average along time of each column of $\widehat{\mathbf{Y}}$ is zero.
Next, we have to modify dataset $\widehat{\mathbf{Y}}$ to have unit variance for each component p. The easiest way is to divide each element $\hat{y}_p(t)$ by the standard deviation along the component, obtaining

$$\begin{aligned}\check{y}_p(t) &\doteq \frac{\hat{y}_p(t)}{\sigma_p} \\ &= \frac{\hat{y}_p(t)}{\sqrt{\frac{1}{T+1}\sum_{\tau=0}^{T}(\hat{y}_p(\tau) - \hat{\mu}_p)^2}}\end{aligned}$$



(a) Un-standardized.

(b) Unit-variance standardization.

(c) Identity-covariance standardization.

Figure 7.10: Comparison between an un-standardized trajectory and two different types of standardization.

$$= \frac{\hat{y}_p(t)}{\sqrt{\frac{1}{T+1}\sum_{\tau=0}^{T}\hat{y}_p^2(\tau)}}, \tag{7.97}$$

where the last identity follows from the fact that dataset $\widehat{\mathbf{Y}}$ has null mean.
With this method, the variance along the time of each component of the dataset is one. This means that the diagonal of the covariance matrix $\breve{\mathbf{\Sigma}}$ of the dataset $\breve{\mathbf{Y}}$ is one in each component. However, in general, the off-diagonal components will be non-null.

Alternatively, we can define the mo

dified dataset $\widetilde{\mathbf{Y}}$ in such a way that the covariance matrix along time of the whole dataset is the identity matrix. To do so, we define

$$\tilde{\mathbf{y}}(t) \doteq \widehat{\mathbf{\Sigma}}^{-\frac{1}{2}}\hat{\mathbf{y}}(t). \tag{7.98}$$

where $\widehat{\mathbf{\Sigma}}^{-\frac{1}{2}}$ denotes the inverse of the square root of the covariance matrix of $\widehat{\mathbf{Y}}$. In this way, the covariance matrix along time of dataset $\widetilde{\mathbf{Y}}$ is the identity matrix.

Figure 7.10 shows the results of applying these transformations to a real 3-dimensional time series. In more details, Figure 7.10a shows the original time series, and Figure 7.10b shows the result of applying (7.96) and (7.97) to obtain three time series with zero mean and unit variance. Figure 7.10c shows the result of applying (7.96) and (7.98) to obtain a 3-dimensional time series with zero mean and a covariance matrix equals to the identity matrix.

7.3. Results

In this Section, we perform tests to validate the AR-HMM model. At first, we test the goodness of our implementation by performing a synthetic test. Next, we apply the EM and Viterbi algorithm on real data-sets to test the effectiveness of the AR-HMM model to segment robotic executions of real tasks.

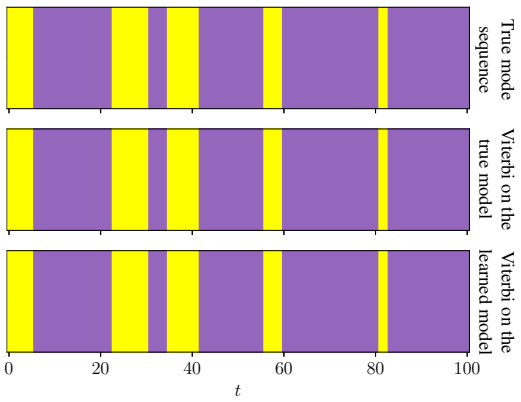


Figure 7.11: Viterbi algorithm applied to the model in Example 7.1. The top row shows the true model segmentation. The middle row shows the segmentation obtained by applying the Viterbi algorithm on the true model. The bottom row shows the segmentation obtained by applying the Viterbi algorithm on the model inferred with the EM algorithm.

7.3.1. Validation Test

In this Section, we perform some tests to validate our implementation of the Expectation-Maximization and Viterbi algorithm. To do so, we use an AR-HMM model to generate a small dataset, and then try to learn the model.

Example 7.1. In this first example, we generate an AR-HMM model with $S = 2$ hidden modes. The true model has initial density

$$\boldsymbol{\varpi} = [0.5 \quad 0.5]^\intercal,$$

and transition matrix

$$\mathbf{T} = \begin{bmatrix} 0.95 & 0.05 \\ 0.1 & 0.9 \end{bmatrix}.$$

The vector field parameters are

$$\mathbf{A}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix},$$
$$\mathbf{A}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}.$$

The covariance matrices are set to

$$\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \sigma^2 \mathbf{Id}_2$$

with $\sigma^2 = 0.1$.

To validate both the Viterbi and the EM algorithm, we perform the following tests. At first, we generate twenty executions with $T = 100$ execution steps each. Then we test the Viterbi algorithm on one of these executions using the correct parameter sets Θ.
Next, we initialize a new AR-HMM model. The parameters are initialized as explained in Section 7.2.3, by clustering on the 'velocity' data-set defined as in (7.94a). Then, we execute the EM algorithm, to learn the set of parameters Θ. After applying the EM algorithm, the 'learned' initial probability is

$$\varpi = [0.55 \quad 0.45]^\intercal,$$

and the transition probability is

$$\mathbf{T} = \begin{bmatrix} 0.9427 & 0.0573 \\ 0.1110 & 0.8890 \end{bmatrix}.$$

After applying the EM algorithm, the inferred vector fields parameters are

$$\mathbf{A}_1 = \begin{bmatrix} 0.7083 & -0.7061 \\ 0.7072 & 0.7071 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} 0.4983 \\ 0.5006 \end{bmatrix},$$

$$\mathbf{A}_2 = \begin{bmatrix} 0.7057 & 0.6985 \\ -0.7047 & 0.7083 \end{bmatrix}, \qquad \mathbf{b}_2 = \begin{bmatrix} -0.5021 \\ 0.4992 \end{bmatrix}.$$

Finally, the learned covariance matrices are

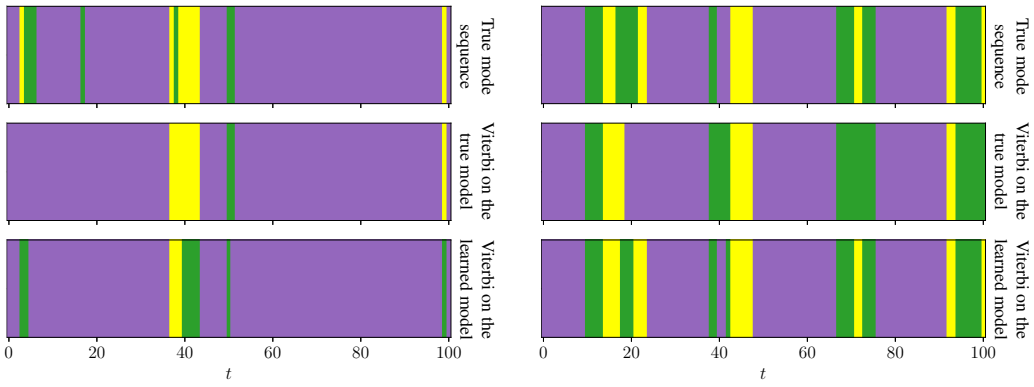
$$\mathbf{\Sigma}_1 = \begin{bmatrix} 0.0098 & 0.0003 \\ 0.0003 & 0.0096 \end{bmatrix}, \qquad \mathbf{\Sigma}_2 = \begin{bmatrix} 0.0101 & -0.0003 \\ -0.0003 & 0.0103 \end{bmatrix}.$$

In Figure 7.11 the result of the segmentation performed with the Viterbi algorithm is shown. The true hidden mode sequence and inferred sequences are obtained from the same sample of the learning set.

This test validates the EM and Viterbi algorithms implementation. The segmentations obtained with the Viterbi algorithm on both the true and inferred model perfectly coincide with the true hidden model sequence. However, this is not usually the case. Indeed, such a 'perfect' result has been obtained thanks to two aspects. First, the underlying model is indeed an AR-HMM. This is not true in general since true data may not have a 'simple' model underneath, and AR-HMM is used as an approximation. Second, the error variance σ^2 in Example 7.1 is quite small.
In the following example, we increase the amount of noise σ^2 to better explain this line of reasoning.

Example 7.2. In this second example, we generate an AR-HMM model with $S = 3$ hidden modes. The true model has initial density

$$\varpi = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}^\intercal,$$



(a) First trial.

(b) Second trial.

Figure 7.12: Viterbi algorithm applied to the model in Example 7.2. The top row shows the true model segmentation. The middle row shows the segmentation obtained by applying the Viterbi algorithm on the true model. The bottom row shows the segmentation obtained by applying the Viterbi algorithm on the model inferred with the EM algorithm.

and transition matrix

$$\mathbf{T} = \begin{bmatrix} 0.90 & 0.05 & 0.05 \\ 0.10 & 0.80 & 0.10 \\ 0.07 & 0.08 & 0.85 \end{bmatrix}.$$

The vector field parameters are

$$\mathbf{A}_1 = \frac{1}{2}\begin{bmatrix} \sqrt{3} & -1 \\ 1 & \sqrt{3} \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\mathbf{A}_2 = \frac{1}{2}\begin{bmatrix} \sqrt{3} & 1 \\ -1 & \sqrt{3} \end{bmatrix}, \qquad \mathbf{b}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\mathbf{A}_3 = \frac{9}{10}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{b}_3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The covariance matrices are set to

$$\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = \sigma^2 \mathbf{I}\mathrm{d}_2$$

with $\sigma^2 = 0.3$.

As for the previous example, we generated twenty executions with $T = 100$ execution steps each. Then, as before, we infer the model with the EM algorithm and apply the Viterbi algorithm both on the true model and on the learned one.

Figure 7.12 shows two different execution of the experiments (including not only the EM and Viterbi algorithm but also the data-set generation).

In the first trial, the learned parameters were

$$\boldsymbol{\varpi} = [0.093 \quad 0.620 \quad 0.287]^\intercal,$$
$$\mathbf{T} = \begin{bmatrix} 0.894 & 0.047 & 0.059 \\ 0.115 & 0.784 & 0.101 \\ 0.099 & 0.167 & 0.734 \end{bmatrix},$$

with dynamics

$$\mathbf{A}_1 = \begin{bmatrix} 0.860 & 0.493 \\ -0.500 & 0.867 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} -0.011 \\ -0.002 \end{bmatrix},$$
$$\mathbf{A}_2 = \begin{bmatrix} 0.845 & -0.479 \\ 0.475 & 0.867 \end{bmatrix}, \qquad \mathbf{b}_2 = \begin{bmatrix} -0.014 \\ 0.000 \end{bmatrix},$$
$$\mathbf{A}_3 = \begin{bmatrix} 0.900 & 0.004 \\ -0.043 & 0.873 \end{bmatrix}, \qquad \mathbf{b}_3 = \begin{bmatrix} 0.005 \\ 0.015 \end{bmatrix},$$

and error matrices

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.212 & 0.001 \\ 0.001 & 0.165 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.112 & -0.003 \\ -0.003 & 0.118 \end{bmatrix}, \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.082 & -0.005 \\ -0.005 & 0.095 \end{bmatrix}.$$

Results of the Viterbi algorithm are shown in Figure 7.12a. Applying the Viterbi algorithm, the percentage of accurately segmented points is 94% when applied on the true model, and 89% when applied on the inferred model.

In the second trial, the learned parameters were

$$\boldsymbol{\varpi} = [0.380 \quad 0.250 \quad 0.370]^\intercal,$$
$$\mathbf{T} = \begin{bmatrix} 0.880 & 0.059 & 0.061 \\ 0.091 & 0.768 & 0.141 \\ 0.137 & 0.108 & 0.755 \end{bmatrix},$$

with dynamics

$$\mathbf{A}_1 = \begin{bmatrix} 0.867 & 0.501 \\ -0.513 & 0.859 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} 0.003 \\ -0.008 \end{bmatrix},$$
$$\mathbf{A}_2 = \begin{bmatrix} 0.875 & -0.483 \\ 0.503 & 0.859 \end{bmatrix}, \qquad \mathbf{b}_2 = \begin{bmatrix} 0.027 \\ -0.015 \end{bmatrix},$$
$$\mathbf{A}_3 = \begin{bmatrix} 0.897 & 0.001 \\ -0.037 & 0.894 \end{bmatrix}, \qquad \mathbf{b}_3 = \begin{bmatrix} -0.031 \\ -0.007 \end{bmatrix},$$

and error matrices

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.087 & -0.002 \\ -0.002 & 0.094 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.114 & 0.001 \\ 0.001 & 0.098 \end{bmatrix}, \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.089 & 0.002 \\ 0.002 & 0.079 \end{bmatrix}.$$

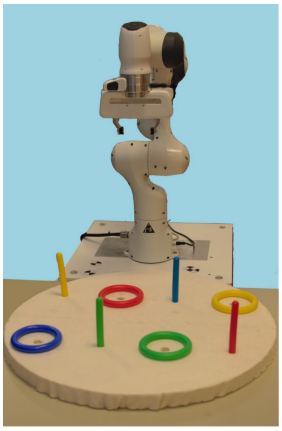


Figure 7.13: Robotic setup for the Peg & Ring task with the Panda industrial manipulator.

Results of the Viterbi algorithm are shown in Figure 7.12b. Applying the Viterbi algorithm, the percentage of accurately segmented points of 87% when applied on the true model, and of 97% on the inferred model.

This second trial is particularly interesting because it remarks on the stochastic nature of the AR-HMM model. Indeed, an 'inexact' model is able to obtain a higher segmentation accuracy than the true model.

7.3.2. Experiments on Real Data

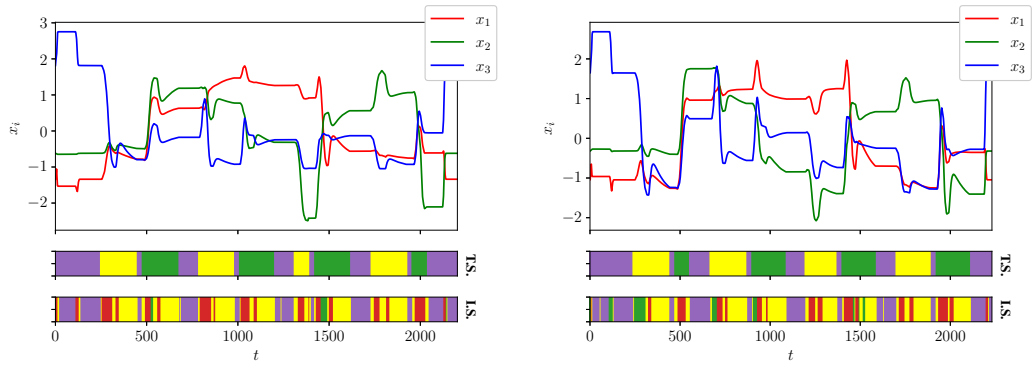
To validate the AR-HMM model as a proper tool to segment real task execution, we test the model on the Peg & Ring task introduced in Chapter 6 on the Panda industrial manipulator (Figure 7.13).

We do not test the 'classical' HMM model because it is not suited for segmenting robot trajectories. Indeed, an HMM assumes that each hidden modes emits an observation which is Gaussian-distributed with given mean and variance. However, when segmenting robotic executions, we assume that at each mode corresponds a particular 'behavior'. Thus, each mode should describe the *evolution* of the system's state, and not the state itself. For this reason, the AR-HMM model is well suited in segmenting trajectories.

On the Panda industrial manipulator, we executed the Peg & Ring task with four colored rings. The setup is shown in Figure 7.13. The dataset consists of fifty executions of the task with the same color order (red, blue, green, and yellow) but changing each time the grasping and releasing position.

We tested the EM and Viterbi algorithm using both the standardization procedures presented in Section 7.2.5 and different amounts of hidden modes.

The task consists of four distinct modes:



(a) Four hidden modes, unit-variance standardization.

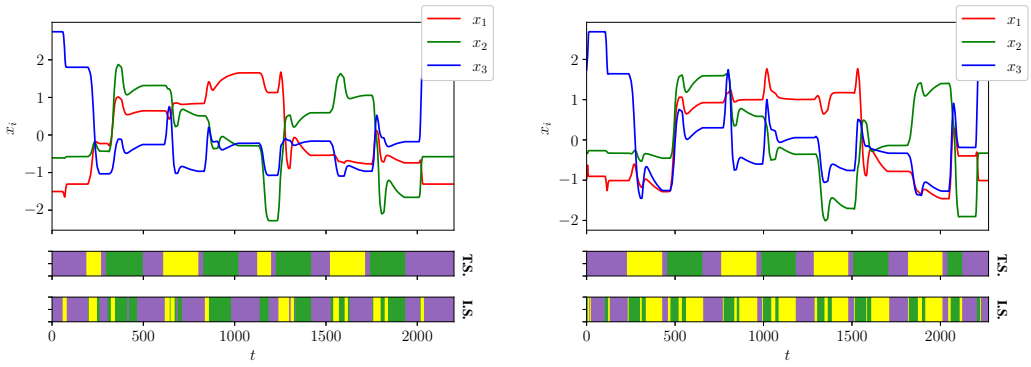
(b) Four hidden modes, identity-covariance standardization.

Figure 7.14: Segmentation using AR-HMM with four modes of trajectories obtained by executing the Peg & Ring task with the Panda industrial manipulator. In both plots, the first (upper) subplot shows the components of the trajectories, the second (middle) plot shows the true segmentation ('T.S.'), and the third (bottom) plot shows the inferred segmentation ('I.S.') obtained by the Viterbi algorithm.

- a *planning phase* in which the system plans the next move, shown in purple in the 'True Segment' (T.S.) plot;
- a *move phase* in which the system is moving the end-effector towards the ring that has to be grasped, shown in yellow in the 'True Segment' (T.S.) plot;
- a *carry phase* in which the system carries the grasped ring towards the same-colored peg, shown in green in the 'True Segment' (T.S.) plot;
- a *ending phase* in which the system returns to the default position once it finishes the task, shown in red in the 'True Segment' (T.S.) plot.

The system alternates the *move* and *carry* phases (with *planning* in between) for all the four rings, and then it ends the task performing the *ending* phase. In some of the samples of the dataset, the recording was stopped before the user system performed the *ending phase*, and thus this phase is not always observed.

At first, we test the AR-HMM model by setting the number of hidden modes equal to four, $S = 4$. Results are shown in Figure 7.14. In these tests, we use a batch of size twenty and test both the standardization procedures from Section 7.2.5. As can be seen from Figure 7.14, the AR-HMM is able to correctly identify the *planning* phases. However, other phases appear to be over-segmented, i.e. a single 'true' segment is split into smaller segments.
In both the test with unit-variance (Figure 7.14a) and identity-covariance (Figure 7.14a) normalization, we see that one of the four modes in the inferred seg-



(a) Three hidden modes, unit-variance standardization.

(b) Three hidden modes, identity-covariance standardization.

Figure 7.15: Segmentation using AR-HMM with three modes of trajectories obtained by executing the Peg & Ring task with the Panda industrial manipulator. In both plots, the first (upper) subplot shows the components of the trajectories, the second (middle) plot shows the true segmentation ('T.S.'), and the third (bottom) plot shows the inferred segmentation ('I.S.') obtained by the Viterbi algorithm.

mentations barely can be found. Thus, we perform more tests with three hidden modes instead of four.

In the following test, we keep the batch-size equal to twenty and test the AR-HMM with both the standardization procedures, using only three hidden modes, $S = 3$. As for the tests with four hidden modes, we see that the AR-HMM is able to correctly find the *planning* phases, but still tends to over-segment both the *move* and *carry* gestures.

The ability of the AR-HMM model to properly segment the planning phase, while over-segmenting both the move and carry gestures lies in the nature of the model. Indeed, the AR-HMM model is build to identify linear dynamics. Thus, the planning phase results properly segmented since in this phase there is barely any movement (and the linear dynamic results to be the null one). On the other hand, the move and carry gestures are too complex to be modeled by simple linear dynamics, and thus the AR-HMM extracts simpler dynamics that, combined, describe the two gestures.

7.4. Conclusions

In this Chapter, we revised the state of the art of the Auto-Regressive Hidden Markov Model. To do so, we first presented the simpler Hidden Markov Model and presented the AR-HMM as a generalization. We presented both the Expectation-

Maximization and the Viterbi algorithm. The former is used to learn the model parameters given a set of observations. The latter is used to extract the most probable sequence of hidden modes that generated an observation given the model parameters. Moreover, we presented a method, based upon the k-means algorithm, to initialize the set of parameters before performing the EM algorithm. We presented also a method to standardize the data-set to improve the numerical stability of the algorithms.

We tested the EM and the Viterbi algorithms on both synthetic and real datasets. Synthetic tests showed that the EM algorithm is able to learn the parameters of the model that generated the datasets and that the Viterbi algorithm is able to infer the sequence of hidden modes that generate the observation.
Real tests showed the main drawback of this type of model, which is being limited to linear dynamics. Indeed, complex dynamics are over-segmented. This means that each gesture is segmented, when using the Viterbi algorithm, into smaller segments. This is due to the fact that each gesture is split into small portions with linear dynamics.

In the next Chapter, we present different modifications to the AR-HMM so as to not limit the model to being limited to linear dynamics.

CHAPTER 8

GENERALIZATION OF AR-HMM

In the previous Chapter, we introduced the 'classical' Auto-Regressive Hidden Markov Model. The main drawback of this model lies in the fact that it can model only linear dynamics. This results in poor segmentation when one aims at using this model to extract complex gestures. Indeed, each gesture is over-segmented, that is, it is split into smaller segments.
In this Chapter, we present two modifications to the AR-HMM to extend the model to more general dynamics. In particular, in Section 8.1 we present the Non-Linear AR-HMM, in which the (linear) Auto-Regressive dynamic of AR-HMM is substituted by a, more general, non-linear dynamic written in terms of basis functions. Then, in Section 8.2 we propose a model able to join the AR-HMM-like latent variable model with a DMP-like dynamic. This last model is still being developed. However, we present analogies with the AR-HMM that justify its further development.

8.1. NON-LINEAR AUTO-REGRESSIVE DYNAMICS

In this Section, we present a Non-Linear AR-HMM. In this model, the linear dynamic of classical AR-HMM is substituted with a more general non-linear dynamic, expressed in terms of basis functions. We will see that classical AR-HMM can be recovered as a particular case of the non-linear one.
The previously introduced AR-HMM adopts a linear dynamic:

$$\mathbf{y}_{t+1}|\mathbf{y}_t, z_{t+1} \sim \mathcal{N}(\mathbf{A}_{z_{t+1}}\mathbf{y}_t + \mathbf{b}_{z_{t+1}}, \mathbf{\Sigma}_{z_{t+1}}).$$

We can generalize the model by substituting the linear dynamic with a non-linear

map $\mathbf{f}_s : \mathbb{R}^d \to \mathbb{R}^d$, thus getting

$$\mathbf{y}_{t+1}|\mathbf{y}_t, z_{t+1} \sim \mathcal{N}(\mathbf{f}_{z_{t+1}}(\mathbf{y}_t), \boldsymbol{\Sigma}_{z_{t+1}}).$$

We remark that the model is still *Auto-Regressive* in the sense that the state at time $t+1$, $\mathbf{y}_{t+1}$, depends on the state at the previous time t, $\mathbf{y}_t$.
In order to be able to learn such a generic non-linear dynamic $\mathbf{f}_s$, $s \in \{1, 2, \dots, S\}$, we need a way to model it using a finite amount of free parameters. To do so, we aim at approximate functions $\mathbf{f}_s$ by expressing it in terms of *basis functions*. Let us define a general set of basis functions $\{\varphi_i : \mathbb{R}^d \to \mathbb{R}\}_{i=0,1,\dots,N, s\in\mathcal{S}}$. We can thus approximate functions $\mathbf{f}_s$ as

$$\mathbf{f}_s(\mathbf{y}) \approx \sum_{i=0}^{N} \boldsymbol{\omega}_i^{(s)} \varphi_i(\mathbf{y}), \tag{8.1}$$

with $\boldsymbol{\omega}_i^{(s)} \in \mathbb{R}^d$ for any $i = 0, 1, \dots, N$, $s \in \mathcal{S}$. Expression (8.1) can be written as follows. Let matrix $\boldsymbol{\Omega}_s \in \mathbb{R}^{d\times(N+1)}$ and vector function $\boldsymbol{\varphi} : \mathbb{R}^d \to \mathbb{R}^{N+1}$ be defined respectively as

$$\boldsymbol{\Omega}_s = \begin{bmatrix} \boldsymbol{\omega}_0^{(s)} & \boldsymbol{\omega}_1^{(s)} & \dots & \boldsymbol{\omega}_N^{(s)} \end{bmatrix}, \tag{8.2a}$$

$$\boldsymbol{\varphi}(\mathbf{y}) = \begin{bmatrix} \varphi_0(\mathbf{y}) & \varphi_1(\mathbf{y}) & \dots & \varphi_N(\mathbf{y}) \end{bmatrix}^\intercal. \tag{8.2b}$$

Then, (8.1) can be written as

$$\mathbf{f}_s(\mathbf{y}) = \boldsymbol{\Omega}_s \boldsymbol{\varphi}(\mathbf{y}). \tag{8.3}$$

We can thus define the *Non-Linear Auto-Regressive Hidden Markov Model*.

Definition 8.1 (Non-Linear AR-HMM)**.** A *Non-Linear Auto-Regressive Hidden Markov Model* (NL-AR-HMM) is a model $\mathcal{H} = \{\mathcal{S}, \mathcal{Y}, \boldsymbol{\varphi}(\cdot), \Theta = \{\boldsymbol{\varpi}, \mathbf{T}, \boldsymbol{\Omega}_s : s \in \mathcal{S}\}\}$ where:

- $\mathcal{S} = \{1, 2, \dots, S\}$ is the set of (hidden) *modes*. The mode at time $t \in \{1, 2, \dots, T\}$ is denoted by z_t.
- Vector $\boldsymbol{\varpi} = [\varpi_1, \varpi_2, \dots, \varpi_S]^\intercal \in \Theta$ defines the *initial probability* of hidden mode:

  $$\Pr(z_1 = i) = \varpi_i, \quad i \in \mathcal{S}.$$

  It satisfies $\varpi_i \in [0, 1]$ and $\sum_{i=1}^{S} \varpi_i = 1$.
- Matrix $\mathbf{T} = [\mathbf{T}|_{ij}]_{i=1,2,\dots,S}^{j=1,2,\dots,S} \in \Theta$ defines the *transition probabilities* between hidden modes:

  $$\Pr(z_{t+1} = j | z_t = i) = \mathbf{T}|_{ij}, \quad i, j \in \mathcal{S}.$$

  It satisfies $\mathbf{T}|_{ij} \in [0, 1]$ and $\sum_{j=1}^{S} \mathbf{T}|_{ij} = 1, \forall i = 1, 2, \dots, S$.

- $\mathcal{Y} = \mathbb{R}^d$ is the set of observations (or emissions). The observation at time $t \in \{0, 1, \ldots, T\}$ is denoted by $\mathbf{y}_t \in \mathbb{R}^d$. The emissions are modeled by the following dynamic:

  $$\mathbf{y}_{t+1}|\mathbf{y}_t, z_{t+1} \sim \mathcal{N}\left(\tilde{\mathbf{f}}_{z_{t+1}}(\mathbf{y}_t), \boldsymbol{\Sigma}_{z_{t+1}}\right),$$

  where $\tilde{\mathbf{f}}_s$ is written in terms of basis functions as (8.3)

  $$\tilde{\mathbf{f}}_s(\mathbf{y}) = \boldsymbol{\Omega}_s \boldsymbol{\varphi}(\mathbf{y}).$$

  Vector $\boldsymbol{\varphi}$ defined as in (8.2b) contains the basis functions φ_i, for $i = 0, 1, \ldots, N$ modeling the non-linear dynamics, and matrix $\boldsymbol{\Omega}_s \in \mathbb{R}^{d \times N+1}$ contains the coefficients $\boldsymbol{\omega}_i^{(s)}, i = 0, 1, \ldots, N, s = 1, 2, \ldots, S$ as defined in (8.2a).

The set Θ contains all the parameters of the model:

$$\Theta = \{\boldsymbol{\varpi}, \mathbf{T}, \boldsymbol{\Omega}_s : s \in \mathcal{S}\}.$$

The graphical model representation is the same as the "classical" AR-HMM (Figure 7.5) since the choice of dynamics does not change the conditional dependence between variables.

8.1.1. Expectation-Maximization Algorithm

In this Section, we will develop the Expectation-Maximization algorithm for the Non-Linear AR-HMM. Since the graphical model representation of the Non-Linear AR-HMM is the same as that of the classical AR-HMM, the complete-data log-likelihood takes the same form (7.54). Therefore, also function $\widetilde{Q}(\Theta, \Theta^{\text{old}})$ to maximize during the learning process takes the same form as (7.55):

$$\begin{aligned}\widetilde{Q}(\Theta, \Theta^{\text{old}}) = &\sum_{z_1 \in \mathcal{S}} \log p(z_1|\Theta) p(z_1|\mathbf{Y}, \Theta^{\text{old}}) + \\ &\sum_{t=0}^{T-1} \sum_{z_{t+1} \in \mathcal{S}} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta) p(z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\ &\sum_{t=1}^{T-1} \sum_{z_t \in \mathcal{S}} \sum_{z_{t+1} \in \mathcal{S}} \log p(z_{t+1}|z_t, \Theta) p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}).\end{aligned} \tag{8.4}$$

The only difference between (8.4) and (7.55) is the expression of the emission probability $p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta)$. To be more precise, both in the classical and Non-Linear AR-HMM, the emission probability is a normal distribution. The only difference is that in the AR-HMM case, the mean is given by the linear dynamic $\mathbf{A}_s \mathbf{y}_t + \mathbf{b}_s$, while in the NL-AR-HMM case it is given by the non-linear dynamic $\boldsymbol{\Omega}_s \boldsymbol{\varphi}(\mathbf{y}_t)$.

We now present in detail the Expectation- and Maximization-step of the EM algorithm. As we will see, the only difference from the EM algorithm of the AR-HMM will be the update rule of the dynamic parameters.

Expectation Step

During the Expectation step, we aim at computing the quantities depending on the old set of parameters Θ^{old} in (8.4).
Since the graphical structure of the Non-Linear AR-HMM is identical to the classical AR-HMM, we have that marginals γ and ξ, as well as forward and backward variables $\hat{\alpha}$ and $\hat{\beta}$ are defined in the same way.
The definition of the marginals γ and ξ is given as in (7.56):

$$\begin{aligned}\gamma(z_t) &\stackrel{\text{def}}{=} p(z_t|\mathbf{Y}, \Theta^{\text{old}}), \quad t = 1, 2, \dots, T;\\ \xi(z_t, z_{t+1}) &\stackrel{\text{def}}{=} p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}}), \quad t = 1, 2, \dots, T-1.\end{aligned}$$

Similarly, the scaled forward and backward variables are defined as in (7.73) and (7.76):

$$\begin{aligned}\hat{\alpha}(z_t) &\stackrel{\text{def}}{=} p(z_t|\mathbf{y}_0, \dots, \mathbf{y}_t, \Theta^{\text{old}}),\\ \hat{\beta}(z_t) &\stackrel{\text{def}}{=} \frac{p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|\mathbf{y}_t, z_t, \Theta^{\text{old}})}{p(\mathbf{y}_{t+1}, \dots, \mathbf{y}_T|\mathbf{y}_0, \dots, \mathbf{y}_t, \Theta^{\text{old}})};\end{aligned}$$

Scaled forward and backward variables can be recursively computed as seen in (7.75) and (7.77)

$$\begin{aligned}\hat{\alpha}(z_t) &= c_t^{-1}\, p(\mathbf{y}_t|z_t, \mathbf{y}_{t-1}, \Theta^{\text{old}}) \sum_{z_{t-1}\in\mathcal{S}} \hat{\alpha}(z_{t-1})\, p(z_t|z_{t-1}, \Theta^{\text{old}}),\\ \hat{\beta}(z_t) &= c_{t+1}^{-1} \sum_{z_{t+1}\in\mathcal{S}} \hat{\beta}(z_{t+1})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\, p(z_{t+1}|z_t, \Theta^{\text{old}});\end{aligned} \tag{8.5}$$

where constant c_t is computed as the normalizing factor in (8.5).
Forward and backward variables are initialized as in (7.80) and (7.81)

$$\begin{aligned}\hat{\alpha}(z_1) &\propto p(\mathbf{y}_1|\mathbf{y}_0, z_1, \Theta^{\text{old}})\, p(z_1|\Theta^{\text{old}}),\\ \hat{\beta}(z_T) &= 1.\end{aligned}$$

Marginal γ and ξ can be written in terms of $\hat{\alpha}$ and $\hat{\beta}$ as, see (7.78) and (7.79),

$$\begin{aligned}\gamma(z_t) &= \hat{\alpha}(z_t)\hat{\beta}(z_t).\\ \xi(z_t, z_{t+1}) &= c_{t+1}^{-1}\, \hat{\alpha}(z_t)\, p(z_{t+1}|z_t, \Theta^{\text{old}})\, p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta^{\text{old}})\hat{\beta}(z_{t+1})\end{aligned}$$

The forward-backward and Expectation Step algorithms for Non-Linear AR-HMM are identical to those for classical AR-HMM, that is Algorithm 7.4 and Algorithm 7.5 respectively. The only difference lies in the expression of the emission probability $p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t)$ in line 9 of Algorithm 7.4 and in line 4 of Algorithm 7.5.

Maximization Step

In the maximization step, we must compute the 'new' set Θ of parameters so to maximize (8.4). In particular, we compute the initial probability vector ϖ, the transition probability matrix $\mathbf{T}$, the matrices of weights $\mathbf{\Omega}_s$, and the covariance matrices $\mathbf{\Sigma}_s$, $s \in \mathcal{S}$ describing the dynamic of each mode. Since only the definition of the dynamic differs from the classical AR-HMM model, we have that the update rules for the initial mode probability density and for the transition matrix are the same as in AR-HMM, (7.83) and (7.85):

$$\varpi_i = Pr(z_1 = i|\mathbf{Y}, \Theta^{\text{old}}); \tag{8.6}$$

$$\mathbf{T}|_{k\ell} = \frac{\sum_{t=1}^{T-1} \Pr(z_t = k, z_{t+1} = \ell|\mathbf{Y}, \Theta^{\text{old}})}{\sum_{t=1}^{T-1} \Pr(z_t = k|\mathbf{Y}, \Theta^{\text{old}})}. \tag{8.7}$$

The only difference in the maximization step lies in the update rule for the matrices of weights $\mathbf{\Omega}_s$, and covariance matrices $\mathbf{\Sigma}_s$, $s \in \mathcal{S}$; even though these update rules are similar in formulation and proof to those of classical AR-HMM.

Proposition 8.1. *Maximization over the weights $\mathbf{\Omega}_s$ describing the AR dynamic and the covariance matrices $\mathbf{\Sigma}_s$, $s \in \mathcal{S}$ is achieved by setting*

$$\mathbf{\Omega}_s = \left(\sum_{t=0}^{T-1} \Pr(z_{t+1} = s|\mathbf{Y}, \Theta^{\text{old}}) \mathbf{y}_{t+1}\, \boldsymbol{\varphi}(\mathbf{y}_t)^\intercal\right)\left(\sum_{t=0}^{T-1} \Pr(z_{t+1} = s|\mathbf{Y}, \Theta^{\text{old}}) \boldsymbol{\varphi}(\mathbf{y}_t)\, \boldsymbol{\varphi}(\mathbf{y}_t)^\intercal\right)^{-1}, \tag{8.8a}$$

$$\mathbf{\Sigma}_s = \frac{\sum_{t=0}^{T-1} \Pr(z_t = s|\mathbf{Y}, \Theta^{\text{old}})\, (\mathbf{y}_{t+1} - \mathbf{\Omega}_s\, \boldsymbol{\varphi}(\mathbf{y}_t))\, (\mathbf{y}_{t+1} - \mathbf{\Omega}_s\, \boldsymbol{\varphi}(\mathbf{y}_t))^\intercal}{\sum_{t=0}^{T-1} \Pr(z_t = s|\mathbf{Y}, \Theta^{\text{old}})}. \tag{8.8b}$$

Proof. The first part of the proof follows the same steps as the proof of Proposition 7.13. The only difference is that we substitute the normal distribution over the AR dynamic (7.88) with the following probability density:

$$p(\mathbf{y}_{t+1}|z_{t+1} = s, \mathbf{y}_t, \Theta) = \mathcal{N}(\mathbf{y}_{t+1}|\mathbf{\Omega}_s\, \boldsymbol{\varphi}(\mathbf{y}_t), \mathbf{\Sigma}_s).$$

Therefore, we arrive at defining the function $\widehat{Q}(\Theta, \Theta^{\text{old}})$ as in (7.90) as (after dropping dependence on s for notation simplicity)

$$\widehat{Q}(\Theta, \Theta^{\text{old}}) = \sum_{t=0}^{T-1} \gamma(t+1)\left(-\frac{1}{2}\log|\mathbf{\Sigma}|\right) - \frac{1}{2}\sum_{t=0}^{T-1} \gamma(t+1)\left((\mathbf{y}_{t+1} - \mathbf{\Omega}\, \boldsymbol{\varphi}(\mathbf{y}_t))^\intercal\, \mathbf{\Sigma}^{-1}\, (\mathbf{y}_{t+1} - \mathbf{\Omega}\, \boldsymbol{\varphi}(\mathbf{y}_t))\right).$$

We now discuss maximization over the weights $\mathbf{\Omega}$. Let us compute

$$\frac{\partial \widehat{Q}(\Theta, \Theta^{\text{old}})}{\partial \mathbf{\Omega}} = -\frac{1}{2}\sum_{t=0}^{T-1} \gamma(t+1)\left(-2\mathbf{\Sigma}^{-1}\, (\mathbf{y}_{t+1} - \mathbf{\Omega}\, \boldsymbol{\varphi}(\mathbf{y}_t))\, \boldsymbol{\varphi}(\mathbf{y}_t)^\intercal\right)$$

$$= \boldsymbol{\Sigma}^{-1}\left(\sum_{t=0}^{T-1}\gamma(t+1)\mathbf{y}_{t+1}\,\boldsymbol{\varphi}(\mathbf{y}_t)^\intercal - \boldsymbol{\Omega}\sum_{t=0}^{T-1}\gamma(t+1)\boldsymbol{\varphi}(\mathbf{y}_t)\,\boldsymbol{\varphi}(\mathbf{y}_t)^\intercal\right).$$

By setting it to zero, we get

$$\boldsymbol{\Omega} = \left(\sum_{t=0}^{T-1}\gamma(t+1)\mathbf{y}_{t+1}\,\boldsymbol{\varphi}(\mathbf{y}_t)^\intercal\right)\left(\sum_{t=0}^{T-1}\gamma(t+1)\boldsymbol{\varphi}(\mathbf{y}_t)\,\boldsymbol{\varphi}(\mathbf{y}_t)^\intercal\right)^{-1}, \tag{8.9}$$

which, by ripristinating the index s, gives (8.8a).

Next, maximization over $\boldsymbol{\Sigma}$ is discussed. Let us compute

$$\frac{\partial\widehat{Q}(\Theta,\Theta^{\text{old}})}{\partial\boldsymbol{\Sigma}^{-1}} = -\frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\frac{\partial}{\partial\boldsymbol{\Sigma}^{-1}}(\log|\boldsymbol{\Sigma}|) - \frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\frac{\partial}{\partial\boldsymbol{\Sigma}^{-1}}\Big((\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))^\intercal\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))\Big). \tag{8.10}$$

The first addend is

$$-\frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\frac{\partial}{\partial\boldsymbol{\Sigma}^{-1}}(\log|\boldsymbol{\Sigma}|) = \frac{1}{2}\boldsymbol{\Sigma}\sum_{t=0}^{T-1}\gamma(t+1),$$

while the second addend is

$$\frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\frac{\partial}{\partial\boldsymbol{\Sigma}^{-1}}\Big((\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))^\intercal\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))\Big) = \frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))^\intercal$$

Combining these results, (8.10) reads

$$\frac{\partial\widehat{Q}(\Theta,\Theta^{\text{old}})}{\partial\boldsymbol{\Sigma}^{-1}} = \frac{1}{2}\boldsymbol{\Sigma}\sum_{t=0}^{T-1}\gamma(t+1) - \frac{1}{2}\sum_{t=0}^{T-1}\gamma(t+1)\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))^\intercal\,.$$

By setting this quantity to zero, we get

$$\boldsymbol{\Sigma} = \frac{\sum_{t=0}^{T-1}\gamma(t+1)\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))\,(\mathbf{y}_{t+1}-\boldsymbol{\Omega}\,\boldsymbol{\varphi}(\mathbf{y}_t))^\intercal}{\sum_{t=0}^{T-1}\gamma(t+1)},$$

which, by ripristinating the index s that we dropped before for notation simplicity, gives (8.8b) □

We remark that, since the values for the marginal γ depend on the old set of parameters Θ^{old}, the weights $\boldsymbol{\Omega}_s$ in (8.8b) are the 'old' values, and not those given from (8.8a). In practice, when implementing the maximization step, it is convenient

Algorithm 8.1 Maximization Step for the Non-Linear AR-HMM.

Input: Marginals $\gamma_i(t)$, $i \in \mathcal{S}$, $t \in \{1,2,T\}$ and $\xi_{ij}(t)$, $i,j \in \mathcal{S}$, $t \in 1,2,\ldots,T-1$; observed data $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T)$, $\mathbf{y}_t \in \mathbb{R}^d$, $t = 0,1,\ldots,T$.

Output: New set of parameter $\Theta = \{\varpi, \mathbf{T}, \mathbf{A}_s, \mathbf{b}_s, s \in \mathcal{S}\}$.

▹ Maximization over the initial mode probability density

1: Set

$$\varpi|_i = \gamma_i(1).$$

▹ Maximization over the transition probability matrix

2: Set

$$\mathbf{T}|_{k\ell} = \frac{\sum_{t=1}^{T-1} \xi_{k\ell}(t)}{\sum_{t=1}^{T-1} \gamma_k(t)}.$$

▹ Maximization over the set of weights $\mathbf{\Omega}_s$, $s \in \mathcal{S}$

3: **for** $s \in \mathcal{S} = \{1,2,\ldots,S\}$ **do**

4: Set

$$\mathbf{\Sigma}_s = \frac{\sum_{t=0}^{T-1} \gamma_s(t+1)\,(\mathbf{y}_{t+1} - \mathbf{\Omega}_s\,\boldsymbol{\varphi}(\mathbf{y}_t))\,(\mathbf{y}_{t+1} - \mathbf{\Omega}_s\,\boldsymbol{\varphi}(\mathbf{y}_t))^\mathsf{T}}{\sum_{t=0}^{T-1} \gamma_s(t+1)}$$

$$\mathbf{\Omega}_s = \left(\sum_{t=0}^{T-1} \gamma_s(t+1)\mathbf{y}_{t+1}\,\boldsymbol{\varphi}(\mathbf{y}_t)^\mathsf{T}\right)\left(\sum_{t=0}^{T-1} \gamma_s(t+1)\boldsymbol{\varphi}(\mathbf{y}_t)\,\boldsymbol{\varphi}(\mathbf{y}_t)^\mathsf{T}\right)^{-1}$$

5: **end for**

to update the covariance matrices first, and then update the set of weights describing the non-linear dynamic.

The M-step is summarized in Algorithm 8.1.

As for linear AR-HMM, we may desire to apply the EM algorithm using a set of observations $\mathbf{\Upsilon} = \{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(K)}\}$, $\mathbf{Y}^{(k)} = (\mathbf{y}_0^{(k)}, \mathbf{y}_1^{(k)}, \ldots, \mathbf{y}_{T^{(k)}}^{(k)})$. The procedure follows as explained in Section 7.2.2. Maximization over the initial mode probability and the transition probability is achieved as in (7.93a) and (7.93b). The only difference arises when updating the dynamics parameters. The matrix of weights $\mathbf{\Omega}_s$ and the covariance matrix $\mathbf{\Sigma}_s$ are updated as follows:

$$\mathbf{\Omega}_s = \left(\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1} \Pr(z_{t+1} = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}})\mathbf{y}_{t+1}^{(k)}\,\boldsymbol{\varphi}(\mathbf{y}_t^{(k)})^\mathsf{T}\right)\left(\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1} \Pr(z_{t+1} = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}})\boldsymbol{\varphi}(\mathbf{y}_t^{(k)})\,\boldsymbol{\varphi}(\mathbf{y}_t^{(k}\right.$$

$$\boldsymbol{\Sigma}_s = \frac{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1} \Pr(z_t = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}})\left(\mathbf{y}_{t+1}^{(k)} - \boldsymbol{\Omega}_s\,\boldsymbol{\varphi}(\mathbf{y}_t^{(k)})\right)\left(\mathbf{y}_{t+1}^{(k)} - \boldsymbol{\Omega}_s\,\boldsymbol{\varphi}(\mathbf{y}_t^{(k)})\right)^{\mathsf{T}}}{\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1} \Pr(z_t = s|\mathbf{Y}^{(k)}, \Theta^{\text{old}})}.$$

8.1.2. Families of Basis Functions

The Non-Linear AR-HMM can model any type of non-linear dynamics, given that the set of basis functions $\{\varphi_i\}_{i=0,1,\dots,N}$ is descriptive enough. In this Section, we propose some family of basis functions.

At first, we show how to recover the classical AR-HMM as a particular case of Non-Linear AR-HMM via a particular choice of basis functions. To do so, we define the basis function $\boldsymbol{\varphi} : \mathbb{R}^d \to \mathbb{R}^{d+1}$ as

$$\boldsymbol{\varphi}(\mathbf{y}) = \boldsymbol{\varphi}\left(\begin{bmatrix} y_1 & y_2 & \cdots & y_d \end{bmatrix}^{\mathsf{T}}\right) \doteq \begin{bmatrix} 1 & y_1 & y_2 & \cdots & y_d \end{bmatrix}^{\mathsf{T}}.$$

In this way, the matrix of coefficients $\boldsymbol{\Omega}$ contains both the linear and bias terms of the linear dynamic. In particular, given a linear dynamic $\mathbf{y}_{t+1} = \mathbf{A}\mathbf{y}_t + \mathbf{b}$, matrix $\boldsymbol{\Omega}$ takes the form $\boldsymbol{\Omega} = [\mathbf{b} \quad \mathbf{A}]$.

A second, well-known family of basis functions is the family of *Gaussian Radial Basis* (GRB) functions. Given a set of mean values $\{\boldsymbol{\nu}_i \in \mathbb{R}^d\}_{i=1,2,\dots,N}$ and covariance matrices $\{\boldsymbol{\Sigma}_i \in \mathbb{R}^{d\times d} : \boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_i^{\mathsf{T}}, \boldsymbol{\Sigma}_i \text{ is positive definite}\}_{i=1,2,\dots,N}$. Then, we can define the basis functions as

$$\varphi_0(\mathbf{y}) = 1, \tag{8.11a}$$

$$\varphi_i(\mathbf{y}) = \exp\left(-(\mathbf{y} - \boldsymbol{\nu}_i)^{\mathsf{T}}\,\boldsymbol{\Sigma}_i^{-1}(\mathbf{y} - \boldsymbol{\nu}_i)\right), \qquad i = 1, 2, \dots, N. \tag{8.11b}$$

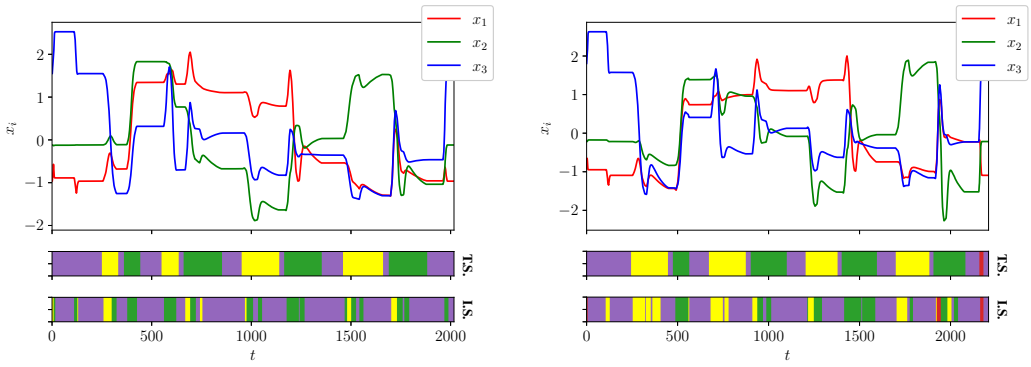
We remark that function in (8.11b) does not coincide with the probability density function of a normal multivariate distribution (for instance, the integral on $\mathbb{R}^d$ is not one). However, this does not matter since we are defining a set of basis functions and not a probability distribution. Usually, the matrices $\boldsymbol{\Sigma}_i$ are defined as multiple of the identity matrix: $\boldsymbol{\Sigma}_i = \varsigma_i \mathbf{I}\mathrm{d}_d$. In this way, definition (8.11b) reads $\varphi_i(\mathbf{y}) = \exp(-\varsigma_i^{-1}\|\mathbf{y} - \boldsymbol{\nu}_i\|^2)$.

A third family of basis functions is the set of polynomial functions up to a degree k, that is the set

$$\boldsymbol{\varphi}(\mathbf{y}) = \left\{\prod_{i=1}^{d} y_i^{c_i} : c_i \in \mathbb{N}, \sum_{i=1}^{d} c_i \le k\right\}. \tag{8.12}$$

8.1.3. (Preliminary) Results

In this Section, we present the segmentation obtained with the NL-ARHMM. For all the tests, data are standardized to have null mean and identity-matrix covariance.



(a) Three hidden modes.

(b) Four hidden modes.

Figure 8.1: Results of the Viterbi algorithm on the Peg & Ring task on the Panda robot using the cubic NL-ARHMM. In both plots, the first (upper) subplot shows the components of the trajectories, the second (middle) plot shows the true segmentation ('T.S.'), and the third (bottom) plot shows the inferred segmentation ('I.S.') obtained by the Viterbi algorithm.

Tests are performed on the same dataset as in Section 7.3.2.
We decided to test the NL-ARHMM using cubic polynomial basis functions, that is basis functions as in (8.12) with $k = 3$. This means that the family of basis functions is the set

$$\left\{\prod_{i=1}^{3} y_i^{c_i} : c_i \in \{0,1,2,3\}, \sum_{i=1}^{3} c_i \leq 3\right\} = \left\{1, y_1, y_2, y_3, y_1y_2, y_1y_3, y_2y_3, y_1^2, y_2^2, y_3^2, y_1y_2y_3, y_1^2y_2, y_1^2y_3, y_2^2y_3, y_1^3, y_2^3, y_3^3\right\}.$$

We tested the model with three, $S = 3$, and four, $S = 4$, hidden modes. Figure 8.1 shows the results of the Viterbi algorithm applied to the NL-ARHMM learned via the EM algorithm. Similar to the classical, linear, AR-HMM, the model correctly identifies the planning phases while it tends to over-segment other gestures. However, at a preliminary observation, over-segmentation seems to be slighted reduced using cubic NL-ARHMM w.r.t. classical AR-HMM (see Figures 7.15 and 7.14), in particular in the case of four hidden modes, $S = 4$.

8.2. DMP-HMM

In the previous Section, we presented a modification of the AR-HMM model to generalize it to non-linear dynamics. In this Section, we introduce a model able to merge the AR-HMM model to the Dynamic Movement Primitives framework. In particular, we will combine the DMP-based dynamical system evolution within the

AR-HMM framework. As we will explain later in detail, this can be interpreted as a natural extension of the AR-HMM model when we will substitute the hidden variable z_t with a set of hidden variables. The development of the inference algorithm is still incomplete. However, we will present the difficulties that may arise during the development, presenting some ways that they can be addressed. As future work, we aim at completing and implementing the EM and Viterbi algorithms.

The idea behind the DMP-HMM model is to use, instead of the linear Auto-Regressive dynamic in the AR-HMM, a DMP-based model, derived from the discretization of the DMP dynamical system. To do so, let us assume that the DMP (2.10) and canonical system (2.2) parameters $\mathbf{K}, \mathbf{D}, \tau$ and α, and the time-step size δ_t of the discretization are fixed. Thus, we can think about DMP evolution as a locally linear Auto-Regressive model where the offset is not constant but depends on the canonical system value. To be more precise, let $\mathbf{y}_t$ and $\mathbf{y}_{t+1}$ denote the DMP system's state at time-step t and $t+1$ respectively. Moreover, let s_t denotes the canonical system value at time step t. We recall, from (2.17), that the DMP evolution can be modeled as

$$\tau \dot{\mathbf{y}} = \boldsymbol{\Xi}\mathbf{y} + \boldsymbol{\beta}(s),$$

where vector $\mathbf{y}$ contains both the DMP system position and velocity, matrix $\boldsymbol{\Xi}$ is defined in (2.18), and vector $\boldsymbol{\beta}(s)$ is defined in (2.19). Since we do not known in advance the goal position at time t and the set of weights $\boldsymbol{\Omega} = \{\boldsymbol{\omega}_i\}$, we consider the more general formulation

$$\tau \dot{\mathbf{y}} = \boldsymbol{\Xi}\mathbf{y} + \boldsymbol{\beta}(s, \mathbf{g}, \boldsymbol{\Omega}). \tag{8.13}$$

By discretizing (8.13) with a time-step size equals to δ_t, and by applying the Exponential Euler numerical integrator, we get

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \delta_t \boldsymbol{\Phi}_1(\delta_t \boldsymbol{\Xi})\left(\boldsymbol{\Xi}\mathbf{y}_t + \boldsymbol{\beta}(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega})\right). \tag{8.14}$$

By calling $\mathbf{P}$ the matrix $\delta_t \boldsymbol{\Phi}_1(\delta_t \boldsymbol{\Xi})$, (8.14) reads

$$\mathbf{y}_{t+1} = (\mathbf{Id} + \delta_t \mathbf{P}\boldsymbol{\Xi})\,\mathbf{y}_t + \delta_t \mathbf{P}\boldsymbol{\beta}(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega}). \tag{8.15}$$

Next, by calling $\widetilde{\mathbf{A}} = \mathbf{Id} + \delta_t \mathbf{P}\boldsymbol{\Xi}$ and $\tilde{\mathbf{b}}(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega}) = \delta_t \mathbf{P}\boldsymbol{\beta}(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega})$, (8.15) can be written as

$$\mathbf{y}_{t+1} = \widetilde{\mathbf{A}}\,\mathbf{y}_t + \tilde{\mathbf{b}}(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega}). \tag{8.16}$$

Thus, we can express the next-state probability distribution as a normal distribution centered at the value given by the r.h.s. of (8.16):

$$\mathbf{y}_{t+1} \sim \mathcal{N}\left(\tilde{\mathbf{A}}\,\mathbf{y}_t + \tilde{\mathbf{b}}\,(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega}), \boldsymbol{\Sigma}\right). \tag{8.17}$$

We remark that other numerical integrator methods (such as forward Euler) would result in a formulation similar to (8.17). Thus, the DMP-HMM model will not be limited by the particular choice of the integrator.
We recall that the Canonical System can be exactly integrated by setting

$$s_{t+1} = \exp\left(-\frac{\alpha}{\tau}\delta_t\right) s_t. \tag{8.18}$$

In summary, the next state probability depends on the canonical system value s_{t+1}, the goal position $\mathbf{g}_{t+1}$, the set of weights $\mathbf{\Omega}$, and the covariance matrix $\mathbf{\Sigma}$ in (8.17). When executing a sequence of DMPs, three events may happen at each time step: the current DMP keeps evolving, the current DMP is 'reset', or a different DMP starts being executed. To model this behavior, we introduce, at each time step, a *reset variable* ε_t with Bernoulli distribution $\Pr(\varepsilon_t = 0) = q_0, \Pr(\varepsilon_t = 1) = q_1 \doteq 1 - q_0$. If the reset variable is zero, there is no reset, and the DMP and Canonical System evolve accordingly (8.17) and (8.18) respectively. On the other hand, if the reset variable takes value one, the Canonical System is reset to its initial value $s = 1$, and the DMP that has to be executed next is sampled using a transition probability, similarly to the transition probability between hidden modes in the classical AR-HMM.

We can thus formally define the DMP-HMM model.

Definition 8.2 (DMP-HMM)**.** A *DMP-HMM* is a model $\mathcal{H} = \left\{\mathcal{S}, \mathcal{Y}, \Theta = \{\boldsymbol{\varpi}, \mathbf{T}, q_1, \{\boldsymbol{\mu}_\mathbf{g}, \mathbf{\Sigma}_\mathbf{g}\}, \{\mathbf{\Omega}_\varsigma, \mathbf{\Sigma}_\varsigma : \varsigma \in \mathcal{S}\}\}\right\}$ where

- $\mathcal{S} = \{1, 2, \ldots, S\}$ is the set of hidden *modes*. The mode at time $t \in \{1, 2, \ldots, T\}$ is denoted by m_t.
- $\mathcal{Y} = \mathbb{R}^{2d}$ is the state space. We recall that, since the DMP system is a second order ODE in normal form, if the state is $d-$dimensional, the DMP will model both its position and its velocity, thus doubling the state-space's number of dimensions.
- Vector $\boldsymbol{\varpi} = [\varpi_i]_{i=1,2,\ldots,S} \in \Theta$ defines the *initial mode probability*:

  $$\Pr(m_1 = i) = \varpi_i, \qquad i \in \mathcal{S}.$$

  It satisfies $\varpi_i \in [0, 1]$ and $\sum_{i=1}^{S} \varpi_i = 1$.
- Matrix $\mathbf{T} = [\mathbf{T}|_{ij}]_{i=1,2,\ldots,S}^{j=1,2,\ldots,S} \in \Theta$ defines the *transition probabilities* between hidden modes:

  $$\Pr(m_{t+1} | m_t = i, \varepsilon_{t+1} = 1) = \mathbf{T}|_{ij}, \qquad i, j \in \mathcal{S}. \tag{8.19}$$

  It satisfies $\mathbf{T}|_{ij} \in [0, 1]$ and $\sum_{j=1}^{S} \mathbf{T}|_{ij} = 1, \forall i = 1, 2, \ldots, S$.

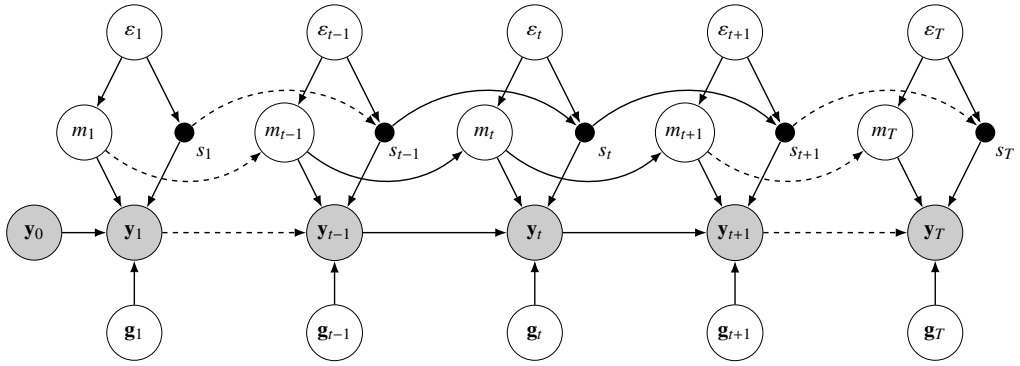


Figure 8.2: Graphical model representation of the DMP-HMM model.

- Scalar $q_1 \in [0,1]$ is the *reset probability* $\varepsilon_t \sim \mathcal{B}(q_1)$, that is

$$\Pr(\varepsilon_t = 1) = q_1, \qquad \Pr(\varepsilon_t = 0) = q_0 \doteq 1 - q_1.$$

  In the case $\varepsilon_t = 1$ a transition is occurring in the latent mode. Such transition is governed as in (8.19). Otherwise, if $\varepsilon_t = 0$, then $m_t = m_{t-1}$. The value of variable ε_t determines the next value of the canonical system:

$$\varepsilon_t = 1 \Rightarrow s_t = 1, \qquad \varepsilon_t = 0 \Rightarrow s_t = \exp\left(-\frac{\alpha}{\tau}\delta_t\right) s_{t-1}.$$

- Parameters $\boldsymbol{\mu}_{\mathbf{g}}$ and $\boldsymbol{\Sigma}_{\mathbf{g}}$ models the normal distribution of the goal position

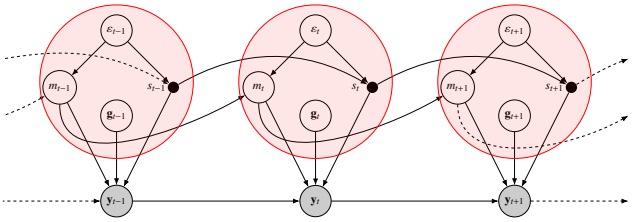
$$\mathbf{g}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{\mathbf{g}}, \boldsymbol{\Sigma}_{\mathbf{g}}\right).$$

- Set $\{\boldsymbol{\Omega}_\varsigma, \boldsymbol{\Sigma}_\varsigma : \varsigma \in \mathcal{S}\}$ models the DMP evolution accordingly to

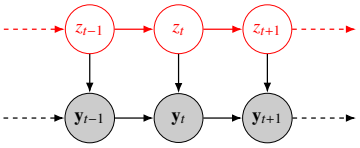
$$\mathbf{y}_{t+1} | m_{t+1} = \varsigma \quad \sim \quad \mathcal{N}\left(\tilde{\mathbf{A}}\,\mathbf{y}_t + \tilde{\mathbf{b}}\,(s_{t+1}, \mathbf{g}_{t+1}, \boldsymbol{\Omega}_\varsigma), \boldsymbol{\Sigma}_\varsigma\right).$$

Figure 8.2 shows the graphical model representation of the DMP-HMM model. We remark that s_t is denoted by a full black dot in the graphical model because it is a deterministic function of the previous value s_{t-1} and the present value of the reset variable ε_t.

Before presenting the development of the Expectation-Maximization algorithm, we remark that the DMP-HMM model can be interpreted as a generalization of the AR-HMM model. Indeed, by considering the latent state z_t as $z_t = (\varepsilon_t, m_t, s_t, \mathbf{g}_t)$, we have that the graphical model structure and, thus, the conditional independent properties are identical to the AR-HMM model. Figure 8.3
This analogy will make some steps in the development of the EM algorithm simple extensions of the AR-HMM case.



(a) DMP-HMM showing the latent space definition z_t.



(b) AR-HMM-like representation of the DMP-HMM.

Figure 8.3: Depiction of the DMP-HMM as a generalization of the AR-HMM.

8.2.1. Expectation-Maximization

In this Section, we introduce the EM algorithm for the DMP-HMM. In particular, we will compute the function $Q(\Theta, \Theta^{\text{old}})$ that has to be maximized at each step, showing how it can be decomposed along different sets of latent variables. In the future, we aim at completing the development of the EM algorithm.

As pointed out before, we will be able to develop it as a generalization of the EM algorithm for AR-HMM. Indeed, we can interpret the DMP-HMM model as an AR-HMM model where the hidden mode z_t is defined as $z_t = (\varepsilon_t, m_t, s_t, \mathbf{g}_t) \in \mathcal{Z} = \{0,1\} \times \mathcal{S} \times \mathbb{R} \times \mathbb{R}^d$. For this reason, we aim at maximize

$$\begin{aligned}\widetilde{Q}(\Theta, \Theta^{\text{old}}) = &\int_{\mathcal{Z}} \log p(z_1|\Theta) p(z_1|\mathbf{Y}, \Theta^{\text{old}})\, dz_1 + \\ &\sum_{t=0}^{T-1} \int_{\mathcal{Z}} \log p(\mathbf{y}_{t+1}|z_{t+1}, \mathbf{y}_t, \Theta) p(z_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, dz_{t+1} + \\ &\sum_{t=1}^{T-1} \iint_{\mathcal{Z}^2} \log p(z_{t+1}|z_t, \Theta) p(z_t, z_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, dz_t\, dz_{t+1}.\end{aligned} \tag{8.20}$$

The main difference between (8.20) and (7.55) lies in the space in which the latent state is defined. Indeed, since in the AR-HMM case the hidden mode lives in a discrete set $z_t \in \mathcal{S} = \{1, 2, \ldots, S\}$, only summation occurres. On the other hand, in the DMP-HMM case, the latent space is a cartesian product of both discrete and continuous sets, thus we use integrals instead of sums.

At first, let us simplify the function (8.20). To begin, let us decompose the integrals over the latent space $\mathcal{Z}$, obtaining

$$\widetilde{Q}(\Theta, \Theta^{\text{old}}) = \sum_{\varepsilon_1} \sum_{m_1 \in \mathcal{S}} \int_{\mathcal{G}} \int_0^1 \log p(\varepsilon_1, m_1, s_1, \mathbf{g}_1|\Theta) p(\varepsilon_1, m_1, s_1, \mathbf{g}_1|\mathbf{Y}, \Theta^{\text{old}})\, ds_1\, d\mathbf{g}_1 + \tag{8.21a}$$

$$\sum_{t=0}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \sum_{m_{t+1}\in\mathcal{S}} \int_{\mathcal{G}} \int_0^1 \log p(\mathbf{y}_{t+1}|\varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}, \mathbf{y}_t, \Theta) \\ p(\varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, d\mathbf{g}_{t+1}\, ds_{t+1} + \tag{8.21b}$$

$$\sum_{t=1}^{T-1} \sum_{\varepsilon_t,\varepsilon_{t+1}\in\{0,1\}} \sum_{m_t,m_{t+1}\in\mathcal{S}} \iint_{\mathcal{G}^2} \int_0^1 \int_0^1 \log p(\varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|\varepsilon_t, m_t, s_t, \mathbf{g}_t, \Theta) \\ p(\varepsilon_t, m_t, s_t, \mathbf{g}_t, \varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, ds_t\, ds_{t+1}\, d\mathbf{g}_t\, d\mathbf{g}_{t+1}. \tag{8.21c}$$

We now discuss separately the three terms in (8.21) to simplify them.
Let us start with (8.21a). Since we fix ε_1 and s_1 to take values 0 and 1 respectively, maximization over those terms is not required. Moreover, we remark that m_1 and $\mathbf{g}_1$ are independent given Θ. Thus, (8.21a) can be written as

$$\sum_{m_1\in\mathcal{S}} \int_{\mathcal{G}} \left(\log p(m_1|\Theta) + \log p(\mathbf{g}_1|\Theta)\right) p(m_1, \mathbf{g}_1|\mathbf{Y}, \Theta)\, d\mathbf{g}_1 = \\ \sum_{m_1\in\mathcal{S}} \int_{\mathcal{G}} \log p(m_1|\Theta) p(m_1, \mathbf{g}_1|\mathbf{Y}, \Theta)\, d\mathbf{g}_1 + \sum_{m_1\in\mathcal{S}} \int_{\mathcal{G}} \log p(\mathbf{g}_1|\Theta) p(m_1, \mathbf{g}_1|\mathbf{Y}, \Theta)\, d\mathbf{g}_1.$$

Finally, using the marginalization rule (7.5a) we get that (8.21a) can be written as

$$\sum_{m_1\in\mathcal{S}} \log p(m_1|\Theta) p(m_1|\mathbf{Y}, \Theta^{\text{old}}) + \int_{\mathcal{G}} \log p(\mathbf{g}_1|\Theta) p(\mathbf{g}_1|\mathbf{Y}, \Theta^{\text{old}})\, d\mathbf{g}_1. \tag{8.22a}$$

Next, we discuss term (8.21b). By conditional independence property

$$\mathbf{y}_{t+1} \perp\!\!\!\perp \varepsilon_{t+1}|m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}, \mathbf{y}_t, \Theta$$

we can marginalize over ε_{t+1} obtaining that (8.21b) is equal to

$$\sum_{t=0}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \sum_{m_{t+1}\in\mathcal{S}} \int_{\mathcal{G}} \int_0^1 \log p(\mathbf{y}_{t+1}|m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}, \mathbf{y}_t, \Theta) p(m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, d\mathbf{g}_{t+1}\, ds_{t+1}. \tag{8.22b}$$

Finally, let us discuss term (8.21c). By the conditional independe property

$$\varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1} \perp\!\!\!\perp \varepsilon_t, \mathbf{g}_t|m_t, s_t, \Theta,$$

and marginalizing over ε_t and $\mathbf{g}_t$, term (8.21c) can be written as

$$\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \sum_{m_t,m_{t+1}\in\mathcal{S}} \int_{\mathcal{G}} \int_0^1 \int_0^1 \log p(\varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|m_t, s_t, \Theta)$$

$$p(m_t, s_t, \varepsilon_{t+1}, m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, ds_t\, ds_{t+1}\, d\mathbf{g}_{t+1}.$$

Next, we can decompose the logarithm and marginalize when possible, getting that the expression above now reads

$$\begin{aligned}
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \log p(\varepsilon_{t+1}|\Theta) p(\varepsilon_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \sum_{m_t, m_{t+1}\in\mathcal{S}} \log p(m_{t+1}|\varepsilon_{t+1}, m_t, \Theta) p(m_t, \varepsilon_{t+1}, m_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \int_0^1 \int_0^1 \log p(s_{t+1}|\varepsilon_{t+1}, s_t, \Theta) p(s_t, \varepsilon_{t+1}, s_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, ds_t\, ds_{t+1} + \\
&\sum_{t=1}^{T-1} \int_{\mathcal{G}} \log p(\mathbf{g}_{t+1}|\Theta) p(\mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, d\mathbf{g}_{t+1}.
\end{aligned}$$

Since the value of s_{t+1} given s_t and ε_{t+1} is deterministic, no maximization over the third term is needed. Thus, maximization over (8.21c) is equivalent to maximize

$$\begin{aligned}
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \log p(\varepsilon_{t+1}|\Theta) p(\varepsilon_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \sum_{m_t, m_{t+1}\in\mathcal{S}} \log p(m_{t+1}|\varepsilon_{t+1}, m_t, \Theta) p(m_t, \varepsilon_{t+1}, m_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \quad (8.22c) \\
&\sum_{t=1}^{T-1} \int_{\mathcal{G}} \log p(\mathbf{g}_{t+1}|\Theta) p(\mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, d\mathbf{g}_{t+1}.
\end{aligned}$$

Thus, by combining (8.22a), (8.22b) and (8.22c), we have that maximization over (8.21) is equivalent to maximization over

$$\begin{aligned}
\widetilde{\widetilde{Q}}(\Theta, \Theta^{\text{old}}) = &\sum_{m_1\in\mathcal{S}} \log p(m_1|\Theta) p(m_1|\mathbf{Y}, \Theta^{\text{old}}) + \\
&\sum_{t=0}^{T-1} \sum_{m_{t+1}\in\mathcal{S}} \int_{\mathcal{G}} \int_0^1 \log p(\mathbf{y}_{t+1}|m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}, \Theta) p(m_{t+1}, s_{t+1}, \mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, ds_{t+1}\, d\mathbf{g}_{t+1} + \\
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \log p(\varepsilon_{t+1}|\Theta) p(\varepsilon_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\
&\sum_{t=1}^{T-1} \sum_{\varepsilon_{t+1}\in\{0,1\}} \sum_{m_t, m_{t+1}\in\mathcal{S}} \log p(m_{t+1}|\varepsilon_{t+1}, m_t, \Theta) p(m_t, \varepsilon_{t+1}, m_{t+1}|\mathbf{Y}, \Theta^{\text{old}}) + \\
&\sum_{t=0}^{T-1} \int_{\mathcal{G}} \log p(\mathbf{g}_{t+1}|\Theta) p(\mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})\, d\mathbf{g}_{t+1}.
\end{aligned} \quad (8.23)$$

8.2.2. Conclusions and Future Development

In this Chapter, we introduced two extensions to the AR-HMM model presented in Section 7.2.
In particular, in Section 8.1 we proposed a *Non-Linear AR-HMM* that allows generalizing the (affine) Auto-Regressive dynamics of the model to general non-linear dynamics. Preliminary results show that this new model is able to reduce over-segmentation of robotic trajectories since it is able to describe more complex dynamics without having to split them into smaller linear parts.
Then, in Section 8.2 we defined a new model, namely the DMP-HMM, that combines DMP-like dynamics into the AR-HMM latent variable model. The development of the Expectation-Maximization algorithm for this model is still incomplete.

As future work, we aim at completing the EM algorithm for the DMP-HMM. In particular, it should be possible to decouple the computation of the forward and backward variables. Indeed, we showed that the target function $\widetilde{\widetilde{Q}}(\Theta, \Theta^{\text{old}})$ that has to be maximized can be split in five distinct components (see (8.23)): initial mode probability, emission probability, reset probability, mode-transition probability, and goal probability. For this reason, future work will focus on the decoupling of the forward and backward variables, together with their update rules and initialization, into the different components. We remark that, if it wasn't possible to perform such a decoupling, one can compute the variable as a function of the whole hidden mode $z_t = (\varepsilon_t, m_t, s_t, \mathbf{g}_t)$ and marginalize it. This is theoretically possible since the function that has to be maximized can be written in the same way as for classical AR-HMM over an extended latent space, see (8.20). However, taking advantage of the decomposition given in (8.23) would result in a computationally faster EM algorithm.
A difficulty that we may face during the development of the EM-algorithm lies in the computation of the probability $p(\mathbf{g}_{t+1}|\mathbf{Y}, \Theta^{\text{old}})$, which may require approximate inference of this function. This would result in a non-exact maximization step in the EM algorithm.

If it happens to not be possible to learn the parameters with the EM algorithm, we could rely on the Gibbs sampling method, recalled in Section E.4.

IV
CONCLUSIONS

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

In this thesis, we addressed the automation of surgical gestures in the context of Robotic Minimally Invasive Surgery. In particular, we investigated two major topics in this context: the learning of a dynamical system describing each movement a surgeon must perform to accomplish a task and the automatic unsupervised segmentation of real trials into such movements.

The gesture learning is performed using the Dynamic Movement Primitives (DMPs) framework. DMPs are able to learn desired behaviors both in Cartesian and Unit-Quaternion space, allowing to model both the evolution of the position and the orientation of the robotic end-effector. The major advantage of DMPs is that the learned behavior can be generalized both spatially and temporally to new goal positions and speed of execution while maintaining a behavior similar to the learned one. We highlighted various desirable properties of DMP that make them well suited for surgical gestures execution. First, a DMP can be learned from a small number of observed behaviors (actually, even a single observation is enough). This is particularly important in surgical scenarios where it is hard to obtain data from real surgical procedures. Second, obstacle avoidance can be implemented in the DMP framework via potential functions. This is of fundamental importance in surgical gesture automation since undesired collisions of the end-effector with patient's tissues may result in, somewhen deadly, damages. Finally, DMPs can be modified to automatically adapt to the robot joints' limit. This allows us to possibly generalize the learned gestures to different robotic setups.

The first contribution of this thesis lies in different improvements of the DMP model.

As the first improvement, we proposed different sets of basis functions to model the

desired behavior. Our proposed sets of basis functions have multiple desirable properties that make them well suited for robotic applications. In particular, proposed basis functions are compactly supported. This property guarantees faster convergence of the DMP system to the goal position, as well as a numerically more stable learning phase.
As the second improvement, we proposed a modification of the DMP framework that improves the scalability of the learned behavior to different starting and goal positions. In this way, the shape of the trajectory is kept the same independently of the start or goal positions. The main advantage of this improvement lies in the fact that a behavior can be straightforwardly adapted to scenarios different from the learned one. Moreover, it is possible to transfer the learned behavior to different robotic setups without the need for any pre-processing phase.
As the third, and final, improvement, we proposed a novel algorithm that allows learning a unique DMP behavior from a set of multiple observations. In real applications, this allows the removal of undesired oscillations from noisy datasets.

The second contribution of this thesis lies in the definition of new methods to deal with obstacle avoidance within the DMP framework. At the state of the art, obstacle avoidance for DMPs was implemented only for point obstacles. In this thesis, we proposed two different methods to deal with volumetric obstacles, both of which require the definition of an isopotential function, that is, a function in which the zero-level set is the obstacle surface. Both our proposed methods define a potential which negative gradient gives a repulsive term that keeps the DMP system away from the obstacle.
The first proposed potential is a 'static' one, that is, it depends only on the position of the system relative to the obstacle. The main advantage of this method is that a 'real' obstacle can be modeled with a unique potential. On the other hand, modeling an obstacle with point obstacles require the definition of a mesh of points covering the obstacle surface. Thus, our method results to be faster to compute.
The second proposed potential is a 'dynamic' one, that is, it depends also on the velocity of the system relative to the obstacle. This improves the obstacle avoidance behavior by resulting in a smoother trajectory that deviates less from the learned behavior.

The third contribution of the thesis lies in the implementation of two frameworks to automate surgical-related tasks. Both frameworks have three components: perception modules, high-level task reasoners, and low-level controllers. In the first framework, the high-level task reasoner is implemented using ontologies. On the other hand, the second framework uses Answer Set Programming as a task reasoner. Both frameworks use Dynamic Movement Primitives as the low-level controllers.
The frameworks are tested on a Peg & Ring task. We choose this particular task

because it mimics some of the challenges of real surgery, and it is used to train and assess surgeons' skills. Results show the ability of DMPs in learning real gestures and imitating the dexterity of expert users.

As future work, we aim at extending the proposed task-automation frameworks to more complex tasks and scenarios.

The second part of the thesis introduces the Auto-Regressive Hidden Markov Model (AR-HMM) to perform unsupervised segmentation of surgical tasks into gestures. AR-HMMs are a type of *latent-variable models*, that is, models in which the observed data depends on hidden, or unseen, variables (called modes). In particular, in the AR-HMM the latent variables determine the auto-regressive model governing the evolution of the observed variable. Once the model parameters have been inferred, it is possible to extract the most probable sequence of hidden modes that generated the observed sequence, thus obtaining a segmentation.
Preliminary results show that the classical AR-HMM tends to over segment executions of the Peg & Ring task. Our theory is that since AR-HMM is limited to linear dynamics, it is not able to capture complex movements, and thus it tends to split them into smaller segments with simpler dynamics. For this reason, we presented a generalization of AR-HMM to handle non-linear dynamics by writing the dynamic as a linear combination of non-linear basis functions.

As a final contribution of the thesis, we proposed a new model that joins the AR-HMM-like latent variable model with the DMP dynamic, by proposing the DMP-HMM. While the development of the inference algorithm for this model is still incomplete, we defined the target function that must be maximized to accomplish the inference and presented a decomposition that will allow a computationally efficient learning phase. Moreover, we presented some analogies between the DMP-HMM and the AR-HMM that will help in the development of the algorithm. We also discussed which difficulties may arise during such development, together with some methods that can be used to solve them

Future work will focus on finishing the development and implementing the DMP-HMM.

V

APPENDICES

APPENDIX A

NUMERICAL INTEGRATION

In this Appendix, we present the numerical integration schemes used in our implementation of Dynamic Movement Primitives (DMPs). In particular, we present two schemes: an exponential method and an adaptive Runge-Kutta (RK) method.
In our implementation of DMPs, when no extra perturbation term is used (i.e. no obstacle avoidance is needed) the exponential method is used, otherwise, the adaptive RK scheme is used.

In full generality, we aim at solving the autonomous Ordinary Differential Equation

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t)) \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases}, \tag{A.1}$$

where $\mathbf{y} \in \mathbb{R}^d$ and $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$.
We remark that we can limit ourselves to autonomous (i.e. not directly dependent on time t) ODE since the DMP system is an autonomous system. Indeed, let us consider formulation (2.17): $\tau\mathbf{z} = \boldsymbol{\Xi}\mathbf{z} + \boldsymbol{\beta}(s)$. We can extend the state vector as $\mathbf{y} \doteq [\mathbf{z}^\intercal, s]^\intercal$, so that (2.17) now reads

$$\tau \begin{bmatrix} \dot{\mathbf{z}} \\ \dot{s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Xi} & \mathbf{0} \\ \mathbf{0}^\intercal & -\alpha \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ s \end{bmatrix} + \begin{bmatrix} \boldsymbol{\beta}(s) \\ 0 \end{bmatrix}.$$

Thus, the DMP system (2.17) with the canonical system (2.2) can be written as a single autonomous system

$$\tau\dot{\mathbf{y}} = \widehat{\mathbf{A}}\,\mathbf{y} + \hat{\mathbf{b}}(\mathbf{y}), \tag{A.2}$$

Algorithm A.1 Exponential Euler.

Input: Vector field parameters $\widehat{\mathbf{A}}, \hat{\mathbf{b}}(\cdot)$, and τ; initial conditions $\mathbf{y}_0$, time step δ_t, total number of time steps N.

Output: Solutions $\mathbf{y}(t)$ for $t = \delta_t, 2\delta_t, \ldots, N\delta_t$

1: Compute matrix $\mathbf{P} = \boldsymbol{\Phi}_1\left(\frac{\delta_t}{\tau}\widehat{\mathbf{A}}\right)$

2: **for** $k = 1, 2, \ldots, N$ **do**

3: Compute the next solution

$$\mathbf{y}(k\delta_t) = \mathbf{y}\left((k-1)\delta_t\right) + \frac{\delta_t}{\tau}\mathbf{P}\left(\widehat{\mathbf{A}}\mathbf{y}((k-1)\delta_t) + \hat{\mathbf{b}}\left(\mathbf{y}((k-1)\delta_t)\right)\right).$$

4: **end for**

with

$$\widehat{\mathbf{A}} = \begin{bmatrix} \boldsymbol{\Xi} & \mathbf{0} \\ \mathbf{0}^\intercal & -\alpha \end{bmatrix}, \tag{A.3a}$$

$$\hat{\mathbf{b}}(\mathbf{y}) = \begin{bmatrix} \beta(s) \\ 0 \end{bmatrix}. \tag{A.3b}$$

A.1. Exponential Euler method

The *exponential Euler method* [53] approximates the solution at time $t + \delta_t$ given the solution at time t as

$$\mathbf{y}(t+\delta_t) \approx \mathbf{y}(t) + \frac{\delta_t}{\tau}\boldsymbol{\Phi}_1\left(\frac{\delta_t}{\tau}\widehat{\mathbf{A}}\right)\left(\widehat{\mathbf{A}}\mathbf{y}(t) + \hat{\mathbf{b}}(\mathbf{y}(t))\right), \tag{A.4}$$

where function $\boldsymbol{\Phi}_1$ for a generic square matrix $\mathbf{A}$ is defined as

$$\boldsymbol{\Phi}_1(\mathbf{A}) \doteq \sum_{j=0}^{\infty} \frac{\mathbf{A}^j}{(j+1)!}, \tag{A.5}$$

and can be efficiently estimated using Padé approximation. This method has the advantage that is numerically more stable than classical Euler scheme. Its main drawback is that matrix $\boldsymbol{\Phi}_1$ is expensive to compute. However, when using DMPs without implementing obstacle avoidance, there is no need for an adaptive time step δ_t. Thus, matrix $\boldsymbol{\Phi}_1(\delta_t/\tau\widehat{\mathbf{A}})$ can be computed once for all.

The numerical integrator is summarized in Algorithm A.1.

A.2. Runge-Kutta scheme

When the DMP is used with the perturbation term $\mathbf{p}(\mathbf{x}, \mathbf{v})$ in (3.1), the non-linearity introduced with the additional term may result in a numerically non stable method

if the time step δ_t is not small enough. To solve this problem, we use an *adaptive* numerical method.

In particular, we adopt an *adaptive Runge-Kutta* method [107]. This family of integrators generates two solutions, one with higher accuracy order than the other, at each integration step. The solution is accepted if the difference between the two estimates is smaller than a given tolerance. Otherwise, the solution is discarded and the time step is reduced. The main advantage of using adaptive RK methods is that the computation of the two estimates has the same intermediate steps, reducing the overall computational cost.

In particular, we adopt the *Bogacki-Shampine* method [10], whose Butcher's tableau is

0	0			
1/2	1/2			
3/4	0	3/4		
1	2/9	1/3	4/9	
	2/9	1/3	4/9	
	7/24	1/4	1/3	1/8

The Bogacki-Shampine method for autonomous systems works as follows. At each time t, let δ_t denote the time-step's size, and let $\mathbf{f}(\mathbf{y}) = \frac{1}{\tau}\left(\widehat{\mathbf{A}}\mathbf{y} + \hat{\mathbf{b}}(\mathbf{y}) + \mathbf{p}(\mathbf{y})\right)$ (term $\mathbf{p}(\cdot)$ encodes the obstacle avoidance perturbation term). At first, the coefficients $\boldsymbol{\xi}_i$ are computed as follows:

$$
\begin{aligned}
\boldsymbol{\xi}_1 &= \mathbf{f}(\mathbf{y}(t)),\\
\boldsymbol{\xi}_2 &= \mathbf{f}\left(\mathbf{y}(t) + \delta_t \frac{1}{2}\boldsymbol{\xi}_1\right),\\
\boldsymbol{\xi}_3 &= \mathbf{f}\left(\mathbf{y}(t) + \delta_t \frac{3}{4}\boldsymbol{\xi}_2\right),\\
\boldsymbol{\xi}_4 &= \mathbf{f}\left(\mathbf{y}(t) + \delta_t\left(\frac{2}{9}\boldsymbol{\xi}_1 + \frac{1}{3}\boldsymbol{\xi}_2 + \frac{4}{9}\boldsymbol{\xi}_3\right)\right).
\end{aligned}
$$

At this point, two solutions $\mathbf{y}^{(1)}(t+\delta_t)$ and $\mathbf{y}^{(2)}(t+\delta_t)$ can be computed:

$$
\begin{aligned}
\mathbf{y}^{(1)}(t+\delta_t) &= \mathbf{y}(t) + \delta_t\left(\frac{7}{24}\boldsymbol{\xi}_1 + \frac{1}{4}\boldsymbol{\xi}_2 + \frac{1}{3}\boldsymbol{\xi}_3 + \frac{1}{8}\boldsymbol{\xi}_4\right),\\
\mathbf{y}^{(2)}(t+\delta_t) &= \mathbf{y}(t) + \delta_t\left(\frac{2}{9}\boldsymbol{\xi}_1 + \frac{1}{3}\boldsymbol{\xi}_2 + \frac{4}{9}\boldsymbol{\xi}_3\right).
\end{aligned}
$$

The estimate $\mathbf{y}^{(1)}$ is a second-order estimation, while $\mathbf{y}^{(2)}$ is a third-order estimation. At each time step, after computing the approximate solutions $\mathbf{y}^{(1)}(t+\delta_t)$ and $\mathbf{y}^{(2)}(t+\delta_t)$, the solution is accepted if

$$
\left\|\mathbf{y}^{(2)}(t+\delta_t) - \mathbf{y}^{(1)}(t+\delta_t)\right\| \leq \texttt{tol}. \tag{A.6}
$$

Algorithm A.2 Bogacki-Shampine method - step.

Input: Vector field $\mathbf{f}(\cdot)$, present solution $\mathbf{z}(t)$, time-step size δ_t, tolerance `tol`, and decrising rate $h \in (0, 1)$.

Output: New time-step size $\tilde{\delta}_t$, solution at time $t + \tilde{\delta}_t$, $\mathbf{z}(t + \tilde{\delta}_t)$

Set `flag_conv` = `False`
Set $\tilde{\delta}_t = \delta_t$
while `not flag_conv` **do**
▷ Compute coefficients $\boldsymbol{\xi}_i$
$\boldsymbol{\xi}_1 = \mathbf{f}(\mathbf{z}(t))$
$\boldsymbol{\xi}_2 = \mathbf{f}\left(\mathbf{z}(t) + \tilde{\delta}_t \frac{1}{2}\boldsymbol{\xi}_1\right)$
$\boldsymbol{\xi}_3 = \mathbf{f}\left(\mathbf{z}(t) + \tilde{\delta}_t \frac{3}{4}\boldsymbol{\xi}_2\right)$
$\boldsymbol{\xi}_4 = \mathbf{f}\left(\mathbf{z}(t) + \tilde{\delta}_t \left(\frac{2}{9}\boldsymbol{\xi}_1 + \frac{1}{3}\boldsymbol{\xi}_2 + \frac{4}{9}\boldsymbol{\xi}_3\right)\right)$
▷ Compute the two approximate solutions
$\mathbf{z}^{(1)}(t + \tilde{\delta}_t) = \mathbf{z}(t) + \tilde{\delta}_t \left(\frac{7}{24}\boldsymbol{\xi}_1 + \frac{1}{4}\boldsymbol{\xi}_2 + \frac{1}{3}\boldsymbol{\xi}_3 + \frac{1}{8}\boldsymbol{\xi}_4\right)$
$\mathbf{z}^{(2)}(t + \tilde{\delta}_t) = \mathbf{z}(t) + \tilde{\delta}_t \left(\frac{2}{9}\boldsymbol{\xi}_1 + \frac{1}{3}\boldsymbol{\xi}_2 + \frac{4}{9}\boldsymbol{\xi}_3\right)$
▷ Check if the time step is small enough
if $\left\|\mathbf{z}^{(2)}(t + \delta_t) - \mathbf{z}^{(1)}(t + \delta_t)\right\| \leq$ `tol` **then**
Set `flag_conv` = `True`
return $\tilde{\delta}_t$, $\mathbf{z}^{(2)}(t + \tilde{\delta}_t)$
else
Decrease the time step: $\tilde{\delta}_t = h\tilde{\delta}_t$
end if
end while

In such case, solution $\mathbf{y}^{(2)}$ is used, since it is the one computed with the more accurate method. Otherwise, if the error above is larger than a given tolerance, the time-step's size δ_t is reduced by a pre-determined percentage and the approximate solutions $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$ re-computed until the error condition (A.6) is satisfied.

The method for a single time step is summarized in Algorithm A.2. When solving the DMP system, the method for a single step has to be run until a condition is satisfied. This condition may be a final time of evaluation of the system or a condition on the convergence to the goal. When implementing the integration on the whole time domain, only three function evaluations are required per each step. Indeed, since the third-order solution $\mathbf{y}^{(2)}$ at each time-step is the argument of $\boldsymbol{\xi}_4$, we have that coefficient $\boldsymbol{\xi}_1$ at the next time-step coincides with it.

APPENDIX B

CONDITION NUMBER THEORY

Here we present a quick recall on the theory of condition number of matrices.

Consider the linear problem $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a non singular matrix, $\mathbf{b} \in \mathbb{R}^N$ is a vector, and $\mathbf{x} \in \mathbb{R}^N$ is the unknown vector. The *condition number* of a non-singular matrix is defined as

$$\text{cond}(\mathbf{A}) \stackrel{\text{def}}{=} ||\mathbf{A}||\,||\mathbf{A}^{-1}||,$$

where $||\mathbf{A}||$ denotes the 2-norm of the matrix $\mathbf{A}$.
When the linear system is numerically solved, we obtain a solution $\tilde{\mathbf{x}}$. Let us define the *residual* $\mathbf{r}$ as

$$\mathbf{r} \stackrel{\text{def}}{=} \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}.$$

From $\mathbf{A}\tilde{\mathbf{x}} = \mathbf{b} - \mathbf{r}$ and $\mathbf{A}\mathbf{x} = \mathbf{b}$, it follows $\mathbf{A}(\mathbf{x} - \tilde{\mathbf{x}}) = \mathbf{r}$, from which

$$||\tilde{\mathbf{x}} - \mathbf{x}|| \leq ||\mathbf{A}^{-1}||\,||\mathbf{r}||\,.$$

Using the inequality $||\mathbf{b}|| \leq ||\mathbf{A}||\,||\mathbf{x}||$, we obtain the following result on the relative error of the solution $\mathbf{x}$:

$$\frac{||\tilde{\mathbf{x}} - \mathbf{x}||}{||\mathbf{x}||} \leq ||\mathbf{A}||\,||\mathbf{A}^{-1}||\frac{||\mathbf{r}||}{||\mathbf{b}||} = \text{cond}(\mathbf{A})\frac{||\mathbf{r}||}{||\mathbf{b}||}.$$

The above inequality shows that the relative error made when solving a linear system is amplified by the condition number of the matrix $\mathbf{A}$.

APPENDIX C

ROTATION IN $\mathbb{R}^d$

We now present the algorithm, presented in [141], that we use in Section 2.6 to generate a matrix $\mathbf{R}$ that rotates a unit vector $\mathbf{x}_0 \in \mathbb{R}^d$, $\|\mathbf{x}_0\| = 1$, into a unit vector $\mathbf{x}_1 \in \mathbb{R}^d$, $\|\mathbf{x}_1\| = 1$: $\mathbf{x}_1 = \mathbf{R}\,\mathbf{x}_0$. The idea behind the algorithm is to create two rotation matrices $\mathbf{R}_0$ and $\mathbf{R}_1$ such that

$$\mathbf{R}_0\,\mathbf{x}_0 = [1,0,0,\ldots,0]^\intercal,$$
$$\mathbf{R}_1\,\mathbf{x}_1 = [1,0,0,\ldots,0]^\intercal.$$

Then, vector $\mathbf{x}_0$ is firstly mapped to vector $[1,0,\ldots,0]^\intercal$ using $\mathbf{R}_0$, and then mapped to vector $\mathbf{x}_1$ using $\mathbf{R}_1$. Indeed, since $\mathbf{R}_0\,\mathbf{x}_0 = \mathbf{R}_1\,\mathbf{x}_1 = \mathbf{e}_1$, and recalling that rotation matrices are orthogonal ($\mathbf{R}^{-1} = \mathbf{R}^\intercal$), we can write

$$\mathbf{x}_1 = \mathbf{R}_1^{-1}\,\mathbf{e}_1 = \mathbf{R}_1^\intercal\,\mathbf{e}_1 = \mathbf{R}_1^\intercal\,\mathbf{R}_0\,\mathbf{x}_0.$$

Algorithm C.1 takes as input a unit vector $\mathbf{x}$ and returns matrix $\mathbf{R}$ such that $\mathbf{R}\,\mathbf{x} = \mathbf{e}_1$. Algorithm C.2 takes as input two unit vectors $\mathbf{x}_0$, $\mathbf{x}_1$, and returns the rotation matrix $\mathbf{R}$ such that $\mathbf{x}_1 = \mathbf{R}\mathbf{x}_0$.

Algorithm C.1 Accelerated Rotation

Input: Vector $\mathbf{x} \in \mathbb{R}^d$ with unit norm $\|\mathbf{x}\| = 1$.

Output: Rotation matrix $\mathbf{R} \in \mathbb{R}^{d \times d}$ that rotates vector $\mathbf{x}$ to the direction of x_1.

Initialize $\mathbf{R} = \mathbf{Id}_d$
Initialize $\texttt{step} = 1$
while $\texttt{step} < d$ **do**
    Set $\mathbf{A} = \mathbf{Id}_d$
    Set $\texttt{n} = 1$
    while $\texttt{n} \leq d - \texttt{step}$ **do**
        Set $r_2 = \mathbf{x}|_{\texttt{n}}{}^2 + \mathbf{x}|_{\texttt{n+step}}{}^2$
        if $r_2 > 0$ **then**
            Set $r = \sqrt{r_2}$
            Set $\texttt{pcos} = \mathbf{x}|_{\texttt{n}}/r$
            Set $\texttt{psin} = -\mathbf{x}|_{\texttt{n+step}}/r$
    ▷ Base 2-dimensional rotation
            $\mathbf{A}|_{\texttt{n,n}} = \texttt{pcos}$
            $\mathbf{A}|_{\texttt{n,n+step}} = -\texttt{psin}$
            $\mathbf{A}|_{\texttt{n+step,n}} = \texttt{psin}$
            $\mathbf{A}|_{\texttt{n+step,n+step}} = \texttt{pcos}$
        end if
    ▷ Move to the next base operation
        Update $\texttt{n} = \texttt{n} + 2\texttt{step}$
    end while
    Update $\texttt{step} = 2 * \texttt{step}$
    Update $\mathbf{x} = \mathbf{A}\,\mathbf{x}$
    Update $\mathbf{R} = \mathbf{A}\,\mathbf{R}$
end while

Algorithm C.2 Rotation Matrix

Input: Vectors $\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^d$ with unit norm $\|\mathbf{x}_0\| = \|\mathbf{x}_1\| = 1$

Output: Rotation matrix $\mathbf{R}$ s.t. $\mathbf{R}\,\mathbf{x}_0 = \mathbf{x}_1$

Compute $\mathbf{R}_0 = \texttt{fnAR}(\mathbf{x}_0)$
Compute $\mathbf{R}_1 = \texttt{fnAR}(\mathbf{x}_1)$
Compute $\mathbf{R} = \mathbf{R}_1^\intercal\,\mathbf{R}_0$

APPENDIX D

QUATERNIONS

In this Chapter, we introduce the concept of *quaternion*, together with some important properties and how they relate to the problem of describing the orientation of a solid body.

D.1. INTRODUCTION TO QUATERNION

Quaternions were introduced for the first time by William Rowan Hamilton in 1843 [52]. They are an extension of the field of complex numbers defined as follows.

Definition D.1 (Quaternion)**.** A *quaternion* $\mathfrak{q} \in \mathbb{H}$ is defined as

$$\mathfrak{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}, \tag{D.1}$$

where a, b, c and d are real numbers, and $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ are symbols satisfying the following property:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1. \tag{D.2}$$

The quantity a is often referred to as *scalar part* of the quaternion, while the quantity $b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ is often referred to as *vector part*. In fact, in Chapter 4 we will use the notations

$$\mathfrak{q} = v + \mathbf{u}, \qquad \mathfrak{q} = [v, \mathbf{u}], \tag{D.3}$$

in which $v = a$ and $\mathbf{u} = [b, c, d]^\intercal$.

Proposition D.1. *Equation* (D.2) *gives the following identities*

$$\mathbf{ij} = \mathbf{k} = -\mathbf{ji}, \tag{D.4a}$$

$$\mathbf{jk} = \mathbf{i} = -\mathbf{kj}, \tag{D.4b}$$

$$\mathbf{ki} = \mathbf{j} = -\mathbf{ik}. \tag{D.4c}$$

Proof. We will prove only (D.4a). Identities (D.4b) and (D.4c) can be proved similarly.
Recalling that $\mathbf{ijk} = -1$, we can right-multiply both hand sides by $\mathbf{k}$ obtaining $\mathbf{ijk}^2 = -\mathbf{k}$, and, by using the fact that $\mathbf{k}^2 = -1$ we get

$$-\mathbf{ij} = -\mathbf{k},$$

from which the first identity of (D.4a).
To prove the second identity, we actually prove that $\mathbf{ij} = -\mathbf{ji}$. To do so, let us just consider the quantity $\mathbf{ij}$. If we perform both a left- and a right-multiplication by $\mathbf{i}$ we obtain $\mathbf{i}^2\mathbf{ji}$ which gives, by recalling $\mathbf{i}^2 = -1$, $-\mathbf{ji}$. □

A multiplicative group structure, named *Hamilton product*, can be defined on the quaternions as follows:

- The real quaternion 1 is the identity element;
- The real quaternions $\mathfrak{q} = a$, $a \in \mathbb{R}$ commute with all other quaternions;
- The product is first given for the basis elements using properties (D.2) (and thus (D.4)), and then extended by using the distributive property.
- Every nonzero quaternion has an inverse w.r.t. the Hamilton product:

$$(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k})^{-1} = \frac{1}{a^2 + b^2 + c^2 + d^2}(a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}).$$

Remark D.1. The Hamilton product is not commutative, but it is associative. Thus the quaternions form an associative algebra over the real numbers.

D.1.1. Operations

We now introduce and investigate some of the operations between quaternions.

Definition D.2 (Sum of quaternions)**.** The *sum* between quaternions is straightforward to define. Let $\mathfrak{q}_1 = a_1 + b_1\mathbf{i} + c_1\mathbf{j} + d_1\mathbf{k}$ and $\mathfrak{q}_2 = a_2 + b_2\mathbf{i} + c_2\mathbf{j} + d_2\mathbf{k}$ be two quaternions, their sum is defined as

$$+ : \mathbb{H} \times \mathbb{H} \to \mathbb{H}$$
$$(\mathfrak{q}_1, \mathfrak{q}_2) \mapsto \mathfrak{q}_1 + \mathfrak{q}_2 = (a_1 + a_2) + (b_1 + b_2)\mathbf{i} + (c_1 + c_2)\mathbf{j} + (d_1 + d_2)\mathbf{k}.$$

In scalar-vector notation we have

$$(v_1 + \mathbf{u}_1) + (v_2 + \mathbf{u}_2) = (v_1 + v_2) + (\mathbf{u}_1 + \mathbf{u}_2).$$

Definition D.3 (Product of quaternions)**.** The *product* between quaternions is given by the Hamilton product defined above. Let $\mathfrak{q}_1 = a_1 + b_1\mathbf{i} + c_1\mathbf{j} + d_1\mathbf{k}$ and $\mathfrak{q}_2 = a_2 + b_2\mathbf{i} + c_2\mathbf{j} + d_2\mathbf{k}$ be two quaternions, their sum is defined as

$$\begin{aligned} * : \mathbb{H} \times \mathbb{H} &\to \mathbb{H} \\ (\mathfrak{q}_1, \mathfrak{q}_2) &\mapsto \mathfrak{q}_1 * \mathfrak{q}_2. \end{aligned}$$

We now show how to perform the computation of the product:

$$\begin{aligned} \mathfrak{q}_1 * \mathfrak{q}_2 &= (a_1 + b_1\mathbf{i} + c_1\mathbf{j} + d_1\mathbf{k}) * (a_2 + b_2\mathbf{i} + c_2\mathbf{j} + d_2\mathbf{k}) \\ &= (a_1a_2) + (a_1b_2 + b_1a_2)\mathbf{i} + (a_1c_2 + c_1a_2)\mathbf{j} + (a_1d_2 + d_1a_2)\mathbf{k} + b_1b_2\mathbf{i}^2 + b_1c_2\mathbf{ij} + b_1d_2\mathbf{ik} + \\ &\qquad c_1b_2\mathbf{ji} + c_1c_2\mathbf{j}^2 + c_1d_2\mathbf{jk} + d_1b_2\mathbf{ki} + d_1c_2\mathbf{kj} + d_1d_2\mathbf{k}^2 \\ &= (a_1a_2) + (a_1b_2 + b_1a_2)\mathbf{i} + (a_1c_2 + c_1a_2)\mathbf{j} + (a_1d_2 + d_1a_2)\mathbf{k} - b_1b_2 + \\ &\qquad b_1c_2\mathbf{k} - b_1d_2\mathbf{j} - c_1b_2\mathbf{k} - c_1c_2 + c_1d_2\mathbf{i} + d_1b_2\mathbf{j} - d_1c_2\mathbf{i} - d_1d_2 \\ &= (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\mathbf{i} + (a_1c_2 + c_1a_2 - b_1d_2 + d_1b_2 \\ &\qquad (a_1d_2 + d_1a_2 + b_1c_2 - c \end{aligned}$$

In scalar-vector form it can be written as

$$(v_1 + \mathbf{u}_1) * (v_2 + \mathbf{u}_2) = (v_1v_2 - \mathbf{u}_1 \cdot \mathbf{u}_2) + (v_1\mathbf{u}_2 + v_2\mathbf{u}_1 + \mathbf{u}_1 \times \mathbf{u}_2), \tag{D.5}$$

where $\cdot$ denotes the standard scalar product in $\mathbb{R}^3$, and $\times$ the standard vector (cross) product in $\mathbb{R}^3$.

We remark that the product between quaternions is not commutative.

Definition D.4 (Conjugate of quaternion)**.** Next operator that we introduce is the *conjugate* of a quaternion. Given a quaternion $\mathfrak{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, we define its conjugate $\bar{\mathfrak{q}}$ as

$$\bar{\mathfrak{q}} \doteq a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}. \tag{D.6}$$

In scalar-vector notation we have

$$\overline{v + \mathbf{u}} = v - \mathbf{u}.$$

We now introduce the concept of *norm* of a quaternion.

Definition D.5 (Norm of quaternion)**.** Given a quaternion $\mathfrak{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, we define its norm as

$$\begin{aligned} \|\cdot\| : \mathbb{H} &\to \mathbb{R} \\ \mathfrak{q} &\mapsto \|\mathfrak{q}\| \doteq \sqrt{a^2 + b^2 + c^2 + d^2}. \end{aligned} \tag{D.7}$$

Proposition D.2. *The operator* (D.7) *defined in Definition D.5 is a norm.*

Proof. For a given quaternion $\mathfrak{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, let us consider the vector $\mathbf{w} = [a, b, c, d]^\intercal$. The quaternion norm $\|\cdot\|$ is equivalent to the 2-norm of the vector $\mathbf{w}$, $\|\mathbf{w}\|_2$. The latter trivially satisfies all the properties of a norm, hence the thesis. □

Proposition D.3. *The norm of a quaternion can be computed as*

$$\|\mathfrak{q}\|^2 = \overline{\mathfrak{q}} * \mathfrak{q}. \tag{D.8}$$

Proof. Writing $\mathfrak{q}$ as $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ and computing the r.h.s. of (D.8)

$$\begin{aligned}
\overline{\mathfrak{q}} * \mathfrak{q} &= (a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k})(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}) \\
&= a^2 + ab\mathbf{i} + ac\mathbf{j} + ad\mathbf{k} - ab\mathbf{i} - b^2\mathbf{i}^2 - bc\mathbf{ij} - bd\mathbf{ik} - ac\mathbf{j} - bc\mathbf{ji} - c^2\mathbf{j}^2 - cd\mathbf{jk} - ad\mathbf{k} - bd\mathbf{ki} - cd\mathbf{kj} - \\
&= (a^2 + b^2 + c^2 + d^2) + (ab - ab - cd + cd)\mathbf{i} + (ac + bd - ac - bd)\mathbf{j} + (ad - bc + bc - ad)\mathbf{k} \\
&= (a^2 + b^2 + c^2 + d^2) \\
&= \|\mathfrak{q}\|^2
\end{aligned}$$

□

Definition D.6 (Quaternion Inverse)**.** The *inverse of a quaternion* $\mathfrak{q} \neq 0$, denoted by $\mathfrak{q}^{-1}$, is the quaternion satisfying

$$\mathfrak{q} * \mathfrak{q}^{-1} = \mathfrak{q}^{-1} * \mathfrak{q} = 1.$$

Proposition D.4. *The inverse of a quaternion can be computed as*

$$\mathfrak{q}^{-1} = \frac{\overline{\mathfrak{q}}}{\|\mathfrak{q}\|^2}.$$

Proof. By left-multiplying the identity $\mathfrak{q} * \mathfrak{q}^{-1} = 1$ by $\overline{\mathfrak{q}}$ we get $\overline{\mathfrak{q}} * \mathfrak{q} * \mathfrak{q}^{-1} = \overline{\mathfrak{q}}$, from which $\|\mathfrak{q}\|^2 \mathfrak{q}^{-1} = \overline{\mathfrak{q}}$. Hence the thesis. □

The set of all quaternions, together with the sum and product operations defined above forms a *non-commutative division ring* $(\mathbb{H}, +, *)$. This means that any non-zero quaternion admits inverse element and that the product $*$ is not commutative.

We now introduce some other operations on quaternion, all of them useful in the DMP formulation in unit quaternion space (4.1).

The first operation we introduce is the *exponential of a quaternion* $\exp : \mathbb{H} \to \mathbb{H}$.

Proposition D.5 (Exponential of a Quaternion)**.** *The exponential of a quaternion* $\mathfrak{q} = v + \mathbf{u}$ *is given by*

$$\exp(\mathfrak{q}) = \mathrm{e}^v \left(\cos\|\mathbf{u}\| + \mathbf{u} \frac{\sin\|\mathbf{u}\|}{\|\mathbf{u}\|} \right). \tag{D.9}$$

Proof. We will prove the result for the quaternion $0+\mathbf{u}$. The general result (D.9) follows from the fact that, if $\mathfrak{q}_1$ and $\mathfrak{q}_2$ commute, then $e^{\mathfrak{q}_1+\mathfrak{q}_2} = e^{\mathfrak{q}_1} * e^{\mathfrak{q}_2} = e^{\mathfrak{q}_2} * e^{\mathfrak{q}_1}$. In general, two quaternions do not commute. However, they always do if one of them is a quaternion with zero vector part. We can thus write $\mathfrak{q} = (v+\mathbf{0}) + (0+\mathbf{u})$, obtaining $\exp(\mathfrak{q}) = \exp(v+\mathbf{0}) * \exp(0+\mathbf{u}) = e^{v}\exp(0+\mathbf{u})$.
To prove the result for quaternion $0+\mathbf{u}$, we start by recalling that the exponential is given by the absolutely convergent series

$$\exp(z) = \sum_{k=0}^{\infty} \frac{z^k}{k!}.$$

We note that $(0+\mathbf{u})^2 = (0+\mathbf{u})*(0+\mathbf{u}) = (b\mathbf{i}+c\mathbf{j}+d\mathbf{k})*(b\mathbf{i}+c\mathbf{j}+d\mathbf{k}) = -b^2-c^2-d^2 = -\|\mathbf{u}\|^2$. This identity implies $\mathbf{u}^3 = -\mathbf{u}\|\mathbf{u}\|^2, \mathbf{u}^4 = \|\mathbf{u}\|^4, \mathbf{u}^5 = \mathbf{u}\|\mathbf{u}\|^4, \ldots$.
We now compute the convergent series

$$\begin{aligned}
\exp(0+\mathbf{u}) &= \sum_{k=0}^{\infty} \frac{(0+\mathbf{u})^k}{k!} \\
&= 1 + \frac{\mathbf{u}}{1!} + \frac{\mathbf{u}^2}{2!} + \frac{\mathbf{u}^3}{3!} + \frac{\mathbf{u}^4}{4!} + \frac{\mathbf{u}^5}{5!} + \frac{\mathbf{u}^6}{6!} + \ldots \\
&= 1 + \frac{\|\mathbf{u}\|\,\mathbf{u}}{1!\,\|\mathbf{u}\|} - \frac{\|\mathbf{u}\|^2}{2!} - \frac{\|\mathbf{u}\|^3\,\mathbf{u}}{3!\,\|\mathbf{u}\|} + \frac{\|\mathbf{u}\|^4}{4!} + \frac{\|\mathbf{u}\|^5\,\mathbf{u}}{5!\,\|\mathbf{u}\|} - \frac{\|\mathbf{u}\|^6}{6!} + \ldots \\
&= \left(1 - \frac{\|\mathbf{u}\|^2}{2!} + \frac{\|\mathbf{u}\|^4}{4!} - \frac{\|\mathbf{u}\|^6}{6!} + \ldots\right) + \frac{\mathbf{u}}{\|\mathbf{u}\|}\left(\frac{\|\mathbf{u}\|}{1!} - \frac{\|\mathbf{u}\|^3}{3!} + \frac{\|\mathbf{u}\|^5}{5!} + \ldots\right) \\
&= \cos\|\mathbf{u}\| + \frac{\mathbf{u}}{\|\mathbf{u}\|}\sin\|\mathbf{u}\|
\end{aligned}$$

Hence the thesis. □

Having defined the exponential, it is natural to define the logarithm.

Proposition D.6 (Logarithm of quaternion)**.** *The logarithm of a quaternion* $\mathfrak{q} = v+\mathbf{u} \neq 0$ *is given by*

$$\log(\mathfrak{q}) = \log(\|\mathfrak{q}\|) + \frac{\mathbf{u}}{\|\mathbf{u}\|}\arccos\left(\frac{v}{\|\mathfrak{q}\|}\right). \tag{D.10}$$

The logarithm of a quaternion is naturally extended to the classical logarithm when $\mathbf{u} = \mathbf{0}$.

We remark that the usual identities involving exponential and logarithm functions are no longer true in general due to the non-commutativity of the quaternion product. In general $\exp(\mathfrak{q}_1 + \mathfrak{q}_2)$ and $\exp(\mathfrak{q}_1) * \exp(\mathfrak{q}_2)$ are not equal. Similarly $\log(\mathfrak{q}_1 * \mathfrak{q}_2)$ and $\log(\mathfrak{q}_1) + \log(\mathfrak{q}_2)$ are not necessarily equal. Moreover, we remark that $\exp(\log(\mathfrak{q})) = \mathfrak{q}$, while it is not true in general that $\log(\exp(\mathfrak{q})) = \mathfrak{q}$.

The logarithm simplifies, for unit quaternions $\mathfrak{p} \in \mathbb{S}^3 \doteq \{\mathfrak{q} \in \mathbb{H} : \|\mathfrak{q}\| = 1\}$ to

$$\log(\mathfrak{p}) = \log(v + \mathbf{u}) = \begin{cases} \arccos(v)\frac{\mathbf{u}}{\|\mathbf{u}\|} & \text{if } \mathfrak{p} = -1 \\ \mathbf{0} & \text{otherwise} \end{cases}.$$

We remark that if the argument is a unit quaternion, the logarithm has image in $\mathbb{R}^3$: $\log : \mathbb{S}^3 \to \mathbb{R}^3$.
The logarithm map can be used to specify a distance metric on $\mathbb{S}^3$ as follows:

$$\text{dist}(\mathfrak{q}_1, \mathfrak{q}_2) = \begin{cases} 2\pi & \text{if } \mathfrak{q}_1 * \overline{\mathfrak{q}_2} = -1 \\ 2\|\log(\mathfrak{q}_1 * \overline{\mathfrak{q}_2})\| & \text{otherwise} \end{cases}.$$

D.2. Relation Between Unit Quaternions and Rotations

We start by reviewing some basic results on the topic of *three-dimensional rotations*. In geometry, different approaches have been developed to model orientations and rotations of objects in three dimensions. We will review some of these approaches (namely *Euler angles* and *rotation matrices*). Then, we will describe in a more detailed way how unit quaternions can be used for this problem.

Remark D.2. In the following, we may use the term *rotation* instead of *orientation*, since orientations are rotations relative to a reference coordinate system.

D.2.1. Euler Angles and Rotation Matrices

We start reviewing some more intuitive ways to describe rotations in $\mathbb{R}^3$, to become more familiar with the topic before presenting the less intuitive unit quaternion representation.
In the following, we will denote by (O, X, Y, Z) and (o, x, y, z) two reference frames in the three-dimensional Euclidean space. We remark that, when talking about rotations (and not translations), we have $O = o$.
We start our discussion talking about *rotation matrices* [29]. A rotation matrix is a matrix whose multiplication with a vector rotates the vector while preserving its length.

Definition D.7. We define the *special orthogonal group* as the set of all 3×3 rotation matrices, equipped with the standard multiplication between matrices. We denote the set as $SO(3)$.

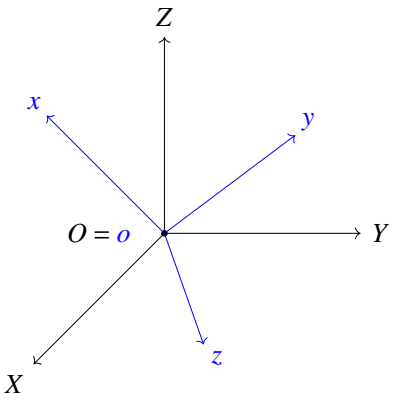


Figure D.1: Two reference frames with different orientations.

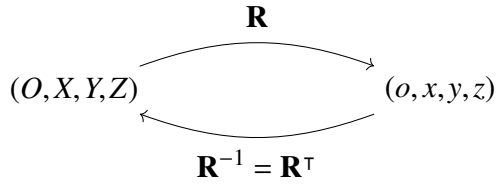


Figure D.2: Relation between two reference frames.

Remark D.3. If $\mathbf{R} \in SO(3)$, then $\det(\mathbf{R}) = \pm 1$ and $\mathbf{R}^\intercal = \mathbf{R}^{-1}$.

Rotation matrices whose determinant is $+1$ are called *proper* rotation matrices, while those for which $\det(R) = -1$ are referred to as *improper*. We will restrict our analysis to proper rotations since improper ones (also called *rotoinversions*) are not rigid-body transformations.
In the following we will refer to the elements of a rotation matrix as follows:

$$R = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}.$$

Remark D.4. There are two conventions for defining the rotation matrix that encodes the attitude of a rigid body. Some authors prefer to write the matrix whose action maps from the reference frame (o, x, y, z) of the rigid body to the "world" frame (O, X, Y, Z), while others prefer to consider the matrix whose action maps the world coordinate frame (O, X, Y, Z) to the one of the rigid body (o, x, y, z). However, we remark that one map is the inverse of the other, thus the matrices performing these two actions are one the transpose of the other.

Given a vector $\mathbf{z} \in \mathbb{R}^3$ whose coordinate are given w.r.t. the reference frame (O, X, Y, Z), let $\mathbf{z}' \in \mathbb{R}^3$ being the same vector written w.r.t. (o, x, y, z), we have the following relations when $o = O$:

$$\mathbf{z}' = \mathbf{R}\mathbf{z}, \qquad \mathbf{z} = \mathbf{R}^{-1}\mathbf{z}' = \mathbf{R}^\intercal \mathbf{z}'.$$

The easier rotations we can think about are the rotations around an axis of the reference frame. We call these rotations *coordinate rotations*.

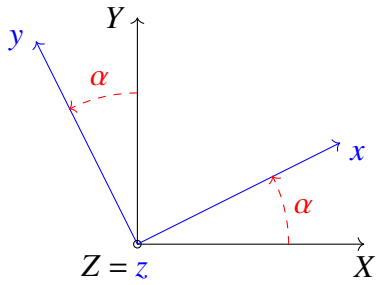


Figure D.3: Scheme of a rotation around the z axis.

Let us denote by $\mathbf{R}_i(\alpha)$ the rotation matrix associated to a rotation of α around the axis i, where $i = 1,2,3$ corresponds, respectively, to the axis x, y, and z. We can write them as

$$\mathbf{R}_1(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix},$$

$$\mathbf{R}_2(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix},$$

$$\mathbf{R}_3(\alpha) = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

A rotation matrix may also be referred to as a *direction cosine matrix* because the elements of the matrix are the cosines of the unsigned angles between the two reference frames. Let us denote by $\theta_{i,j}$, $i \in \{x,y,z\}$, $j \in \{X,Y,Z\}$ the unsigned angle between the axis i of the reference frame (o,x,y,z) and the axis j of the reference frame (O,X,Y,Z). In terms of these angles, the rotation matrix can be written as

$$\begin{aligned} \mathbf{R} &= \left[\cos(\theta_{i,j})\right]_{i,j} \\ &= \begin{bmatrix} \cos(\theta_{x,X}) & \cos(\theta_{x,Y}) & \cos(\theta_{x,Z}) \\ \cos(\theta_{y,X}) & \cos(\theta_{y,Y}) & \cos(\theta_{y,Z}) \\ \cos(\theta_{z,X}) & \cos(\theta_{z,Y}) & \cos(\theta_{z,Z}) \end{bmatrix}. \end{aligned}$$

Example D.1. Let us consider the case in which the reference frame (o,x,y,z) is obtained by rotating the reference frame (O,X,Y,Z) about an angle α around the $Z = z$ axis. Here, $\theta_{x,X} = \theta_{y,Y} = \alpha$, $\theta_{x,Y} = \frac{\pi}{2} - \alpha$, $\theta_{y,X} = \frac{\pi}{2} + \alpha$, $\theta_{x,Z} = \theta_{y,Z} = \theta_{z,X} = \theta_{z,Y} = \frac{\pi}{2}$, and $\theta_{z,Z} = 0$, obtaining

$$\mathbf{R} = \begin{bmatrix} \cos(\theta_{x,X}) & \cos(\theta_{x,Y}) & \cos(\theta_{x,Z}) \\ \cos(\theta_{y,X}) & \cos(\theta_{y,Y}) & \cos(\theta_{y,Z}) \\ \cos(\theta_{z,X}) & \cos(\theta_{z,Y}) & \cos(\theta_{z,Z}) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\alpha) & \cos(\alpha - \frac{\pi}{2}) & \cos(\frac{\pi}{2}) \\ \cos(\alpha + \frac{\pi}{2}) & \cos(\alpha) & \cos(\frac{\pi}{2}) \\ \cos(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) & \cos(0) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

An intuitive representation of orientation is given by *Euler angles* [29, 136]. Three coordinate rotations in sequence can describe any rotation. Let us consider triple rotations in which the first rotation is an angle ϑ_k about the k-axis, the second rotation is an angle ϑ_j around the j-axis, and the third, and last, rotation is an angle ϑ_i around the i-axis. For notation convenience, let us arrange these angles in the vector $\boldsymbol{\vartheta} \doteq [\vartheta_i, \vartheta_j, \vartheta_k]^\intercal$, called *Euler angle vector*. The function that maps the Euler angle vector to its corresponding rotation matrix, denoted by $R_{ijk} : \mathbb{R}^3 \to SO(3)$, is

$$R_{ijk}(\vartheta_i, \vartheta_j, \vartheta_k) = \mathbf{R}_i(\vartheta_i)\mathbf{R}_j(\vartheta_j)\mathbf{R}_k(\vartheta_k).$$

We remark that, of all possible sequences of three integers $\{1,2,3\}$, only twelve of them satisfy the condition that no two consecutive numbers are equal. Of these twelve, the most used in the literature are the sequences $(1,2,3)$, $(3,1,3)$, and $(3,2,3)$.

D.2.2. Unit Quaternion Space

In the literature, the preferred method to describe orientation is to use the space of unit quaternions $\mathbb{S}^3 \doteq \{\mathfrak{q} \in \mathbb{H} : \|\mathfrak{q}\| = 1\}$. Indeed, Euler angles may have some configurations that are *singular*, and rotation matrices, while being singularity-free, require nine components that are not independent one from the others. On the other hand, unit quaternions provide a singularity-free, non-minimal representation of orientation (as rotation matrices do), but with only four parameters instead of nine. The non-minimality trivially follows from the unit-norm condition. Indeed, unit quaternions require four parameters but have only three degrees of freedom.

To understand how unit quaternions can be used to describe orientations, we start by presenting some remarks on the operation of multiplication between quaternions.

We start by representing a quaternion $\mathfrak{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \in \mathbb{H}$ as a vector $\mathbf{q} = [a,b,c,d]^\intercal \in \mathbb{R}^4$. We will denote $q_0 = a$ and $\mathbf{q}_{1:3} = [b,c,d]^\intercal$. With this notation, the quaternion product in Definiton D.3 can be written as

$$\mathbf{q} * \mathbf{p} = \begin{bmatrix} q_0 p_0 - \mathbf{q}_{1:3}^\intercal \mathbf{p}_{1:3} \\ q_0 \mathbf{p}_{1:3} + p_0 \mathbf{q}_{1:3} - \mathbf{q}_{1:3} \times \mathbf{p}_{1:3} \end{bmatrix} \tag{D.11}$$

$$= \begin{bmatrix} q_0 & -\mathbf{q}_{1:3}^\intercal \\ \mathbf{q}_{1:3} & q_0 I_3 - C(\mathbf{q}_{1:3}) \end{bmatrix} \begin{bmatrix} p_0 \\ \mathbf{p}_{1:3} \end{bmatrix}$$
$$= \begin{bmatrix} p_0 & -\mathbf{p}_{1:3}^\intercal \\ \mathbf{p}_{1:3} & q_0 I_3 + C(\mathbf{q}_{1:3}) \end{bmatrix} \begin{bmatrix} q_0 \\ \mathbf{q}_{1:3} \end{bmatrix}$$

where the matrix function $C : \mathbb{R}^3 \to \mathbb{R}^{3\times 3}$ is defined as

$$C(\mathbf{x}) = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}.$$

We remark that the product on the left hand side in (D.11) has to be intended as the product between the quaternions described by the vectors $\mathbf{p}$ and $\mathbf{q}$.
More compactly, quaternion multiplication can be written as

$$\mathbf{q} * \mathbf{p} = Q(\mathbf{q})\mathbf{p} = \widetilde{Q}(\mathbf{p})\mathbf{q}$$
$$\mathbf{p} * \mathbf{q} = Q(\mathbf{p})\mathbf{q} = \widetilde{Q}(\mathbf{q})\mathbf{p}$$

where the *quaternion matrix* function $Q : \mathbb{H} \to \mathbb{R}^{4\times 4}$ is defined as

$$Q(\mathbf{q}) = \begin{bmatrix} q_0 & -\mathbf{q}_{1:3}^\intercal \\ \mathbf{q}_{1:3} & q_0 I_3 + C(\mathbf{q}_{1:3}) \end{bmatrix} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix},$$

and the closely related *conjugate quaternion matrix* function $\widetilde{Q} : \mathbb{H} \to \mathbb{R}^{4\times 4}$ is defined as

$$\widetilde{Q}(\mathbf{q}) = \begin{bmatrix} q_0 & -\mathbf{q}_{1:3}^\intercal \\ \mathbf{q}_{1:3} & q_0 I_3 - C(\mathbf{q}_{1:3}) \end{bmatrix} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix}.$$

The following relations hold:

$$Q(\overline{\mathbf{q}}) = Q(\mathbf{q})^\intercal,$$
$$\widetilde{Q}(\overline{\mathbf{q}}) = \widetilde{Q}(\mathbf{q})^\intercal.$$

Let us now consider a vector $\mathbf{z} \in \mathbb{R}^3$ in the global coordinates, and let $\mathbf{z}' \in \mathbb{R}^3$ be the same vector in the body-fixed coordinates. The following relations hold (where the operations of inverse and conjugation have to be intended in quaternion sense):

$$\begin{bmatrix} 0 \\ \mathbf{z}' \end{bmatrix} = \mathbf{q} \begin{bmatrix} 0 \\ \mathbf{z} \end{bmatrix} \mathbf{q}^{-1} = \mathbf{q} \begin{bmatrix} 0 \\ \mathbf{z} \end{bmatrix} \overline{\mathbf{q}} = \widetilde{Q}(\mathbf{q})^\intercal Q(\mathbf{q}) \begin{bmatrix} 0 \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^\intercal \\ \mathbf{0} & R_q(\mathbf{q}) \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{z} \end{bmatrix},$$

with

$$R_q(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}.$$

That is

$$\begin{aligned} \mathbf{z}' &= R_q(\mathbf{q})\mathbf{z}, \\ \mathbf{z} &= R_q(\mathbf{q})^\intercal \mathbf{z}'. \end{aligned}$$

With this result, we have proven that each unit quaternion corresponds to a unique rotation matrix. Thus, any unit quaternion corresponds to a rotation.

The time derivative of a quaternion $\dot{\mathfrak{q}}$ is related to the angular velocity ω by the following relation:

$$\dot{\mathfrak{q}} = \frac{1}{2}\omega * \mathfrak{q}, \tag{D.12}$$

where ω should be intended as a quaternion with null real part. That is, if the angular velocity is $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^\intercal$, then quaternion ω is defined as $\omega = \mathbf{i}\omega_1 + \mathbf{j}\omega_2 + \mathbf{k}\omega_3$. From relation (D.12) follows the following formula, called *quaternion propagation*.

Proposition D.7 (Quaternion propagation)**.** *Let $\mathfrak{q} \in \mathbb{S}^3$ be a unit quaternion with scalar part v and vector part $\mathbf{u}$. Moreover, let $\boldsymbol{\omega} \in \mathbb{R}^3$ be the angular velocity. Then, the time derivative of $\mathfrak{q} \in \mathbb{H}$ can be computed as*

$$\begin{aligned} \dot{v} &= -\tfrac{1}{2}\langle \mathbf{u}, \boldsymbol{\omega} \rangle \\ \dot{\mathbf{u}} &= \tfrac{1}{2}(v\mathbf{I}\mathrm{d}_3 - \mathbf{S}(\mathbf{u}))\boldsymbol{\omega} \end{aligned} \tag{D.13}$$

where $\mathbf{S}([u_1, u_2, u_3]^\intercal)$ is the skew-symmetric matrix

$$\mathbf{S}([u_1, u_2, u_3]^\intercal) = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \tag{D.14}$$

Proof. Let us compute the right hand side of (D.12) in scalar-vector notation. Using (D.5) we obtain

$$\begin{aligned} \dot{\mathfrak{q}} &= \frac{1}{2}((0 + \boldsymbol{\omega}) * (v + \mathbf{u})) \\ &= \frac{1}{2}(v\boldsymbol{\omega} + \boldsymbol{\omega} \times \mathbf{u}) \\ &= \frac{1}{2}(v\boldsymbol{\omega} - \mathbf{u} \times \boldsymbol{\omega}) \\ &= \frac{1}{2}(v\mathbf{I}\mathrm{d}_3 - \mathbf{S}(\mathbf{u}))\boldsymbol{\omega} \end{aligned}$$

where we used the fact that

$$\mathbf{u} \times \boldsymbol{\omega} = \mathbf{S}(\mathbf{u})\boldsymbol{\omega}.$$

□

APPENDIX E

PROBABILITY THEORY AND GRAPHICAL MODELS

In this Chapter, we will recall the concepts of Probability Theory focusing, in particular, on the topic of *Graphical Models*. In [9] a complete treatment of probabilistic graphical model can be found. Here, we present only the concepts needed for the understanding of Part III of this thesis.

E.1. PROBABILITY THEORY

In this Section, we recall the most important concepts of Probability Theory. We assume that the concepts of *probability of an event*, *random variable*, and *probability densities* (both discrete and continuous) are known. We will recall some important results such as the *sum rule*, the *product rule*, and the *Bayes' Theorem*; as well as concepts such as the *expectation* of a random variable.

At first, let us recall some definitions.

Definition E.1 (Joint probability)**.** We define the *joint probability of two random variables x and y* as the probability of both events x and y, and we denote it as $p(x,y)$.

Definition E.2 (Conditional probability)**.** We define the *conditional probability of x given y* as the probability of event x given the event y, and we denote it as $p(x|y)$.

Given these two definitions, we can give some important results of probability theory.

Proposition E.1 (Sum rule)**.** *The probability of an event x can be written as the*

sum, over the set all possible events for y, of the joint probability $p(x, y)$:

$$p(x) = \sum_{y \in Y} p(x, y). \tag{E.1a}$$

In the case in which y is a continuous random variable, (E.1a) *is written as*

$$p(x) = \int_Y p(x, y)\, dy. \tag{E.1b}$$

Proposition E.2 (Product Rule)**.** *The probability of the joint probability of random variables x and y is the product between the probability of x conditioned to y and the probability of y:*

$$p(x, y) = p(x|y)p(y). \tag{E.2}$$

Theorem E.1 (Bayes)**.** *Given two random variables x and y, and assuming that $p(x)$ never vanishes, the following relation holds true:*

$$p(y|x) = \frac{p(x|y)\, p(y)}{p(x)}. \tag{E.3}$$

Proof. From the product rule (E.2) and the fact that joint probabilities are symmetric $p(x, y) = p(y, x)$, we have

$$p(y|x)p(x) = p(y, x) = p(x, y) = p(x|y)p(y).$$

Since $p(x)$ never vanishes, we can divide by $p(x)$ the first and last terms, obtaining the thesis. □

We remark that Bayes' theorem can be relaxed by not assuming that $p(x)$ never vanishes. In such case, (E.3) reads

$$p(y|x)p(x) = p(x|y)p(y).$$

We now recall the concept of expected value.

Definition E.3 (Expectation of a random variable)**.** Given a discrete random variable x which can take values in X, and has probability density $p(x)$, we define the *expected value of x* as

$$\mathbb{E}[x] = \sum_{x \in X} x\, p(x). \tag{E.4a}$$

Equation (E.4a) reads, in the case of continuous random variables,

$$\mathbb{E}[x] = \int_Y x\, p(x)\, dx. \tag{E.4b}$$

We can generalize the concept of expectation as follows.

Definition E.4 (Expectation of a function of a random variable)**.** Given a discrete random variable x which can assume values in X and has probability density $p(x)$, and given a function f of x, we define the *expected value of* $f(x)$ as

$$\mathbb{E}_x[f] = \sum_{x \in X} f(x)\, p(x). \tag{E.5a}$$

Equation (E.5a) reads, in the case of continuous random variables,

$$\mathbb{E}_x[f] = \int_Y f(x)\, p(x)\, dx. \tag{E.5b}$$

We remark that the concept of expectation of a random variable can be interpreted as a particular case of expectation of a function of a random variable in which the function f is the identity: $f(x) = x$.

E.2. Probabilistic Graphical Models

The idea of using graphical models in probabilities gives many advantages. Most importantly, they provide a simple way to visualize the structure of a model, give insights into the properties of the model itself, and permit the expression of complex probability-related computations in terms of graphical manipulations.
In full generality, let us consider N random variables $x_1, x_2, \ldots, x_N$. The joint probability is $p(x_1, x_2, \ldots, x_N)$. By applying the product rule of probability, we can write it as

$$p(x_1, x_2, \ldots, x_N) = p(x_N | x_1, x_2, \ldots, x_{N-1})\, p(x_1, x_2, \ldots, x_{N-1}).$$

By keep iterating the product rule, we obtain the following identity

$$p(x_1, x_2, \ldots, x_N) = p(x_N | x_1, x_2, \ldots, x_{N-1}) p(x_{N-1} | x_1, x_2, \ldots, x_{N-2}) \cdots p(x_2 | x_1)\, p(x_1). \tag{E.6}$$

The graphical model for the right hand side of (E.6) is obtained by following the two steps:

1. For each random variable, a node is created;

2. For each conditional distribution $p(x_n | x_1, x_2, \ldots, x_{n-1})$, a directed edge is created from the node representing x_i, $i = 1, 2, \ldots, n-1$ to the node representing x_n.

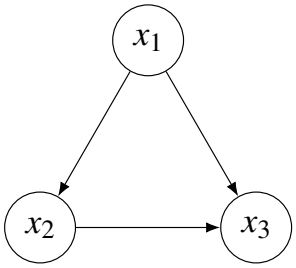


Figure E.1: Graphical model representation of the right hand side of (E.7).

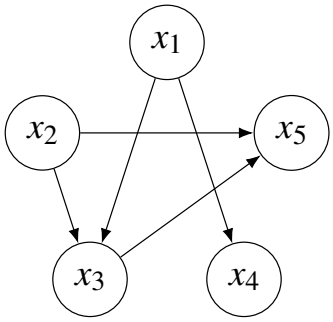


Figure E.2: Graphical model representation of the right hand side of (E.8).

For instance, if $N = 3$, (E.6) reads

$$p(x_1, x_2, x_3) = p(x_3|x_1, x_2)\, p(x_2|x_1)\, p(x_1). \tag{E.7}$$

We remark that the decomposition on the right hand side of (E.6) (and, consequently, (E.7)) is not symmetrical, while the left hand side is. This implies that the joint distribution has not a unique graphical representation. Moreover, we also remark that for a generic joint distribution, the resulting graph is *fully connected*, that is, between each couple of nodes there exists an edge connecting those nodes.
In general, a graphical model is not a fully connected graph. For instance, let us consider a joint distribution between five random variables $x_1, x_2, \ldots, x_5$ satisfying

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_5|x_3, x_2)\, p(x_4|x_1)\, p(x_3|x_1, x_2) p(x_2) p(x_1). \tag{E.8}$$

The graphical model, depicted in Figure E.2, is not a fully connected graph (for instance, the edge between x_1 and x_5 is missing).
As general statement we have that the joint distribution of a set of random variables $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$ in a graphical model is given by the product over all the nodes of the graph of the conditional distribution of node x_n conditioned to all the parents of x_n in the graph:

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(x_n|\mathrm{pa}_n),$$

where pa_n denotes the set of nodes who are parents of x_n.
We remark that the graphs that we consider are *Directly Acyclic Graph*. That is, there are no paths in the graph that start on the same node on which they end.

When treating models with both latent and observed variables, it is useful to have a way to distinguish these two sets. To this end, in a graphical model, we will

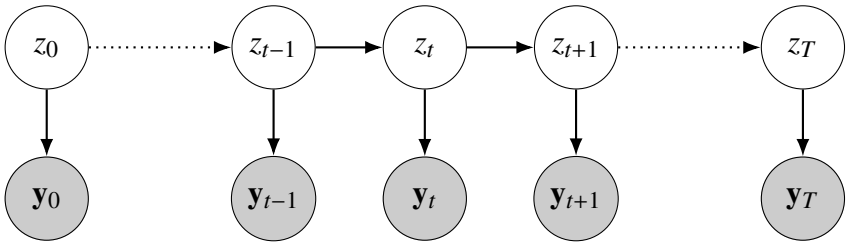
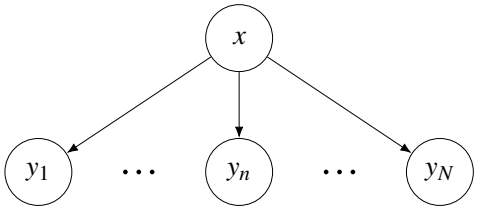
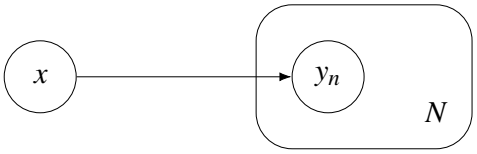


Figure E.3: Graphical model representation of an Hidden Markov Model.



(a) Graphical representation of (E.9) without the compression.



(b) Graphical representation of (E.9) with the compression.

Figure E.4: Different representations of the right hand side of (E.9).

use white nodes to denote latent (or unseen) random variables, and colored nodes to denote observed variables. For instance, Figure E.3 shows an *Hidden Markov Model*, in which variables z_n are the latent variables, while variables y_n are the observed ones.

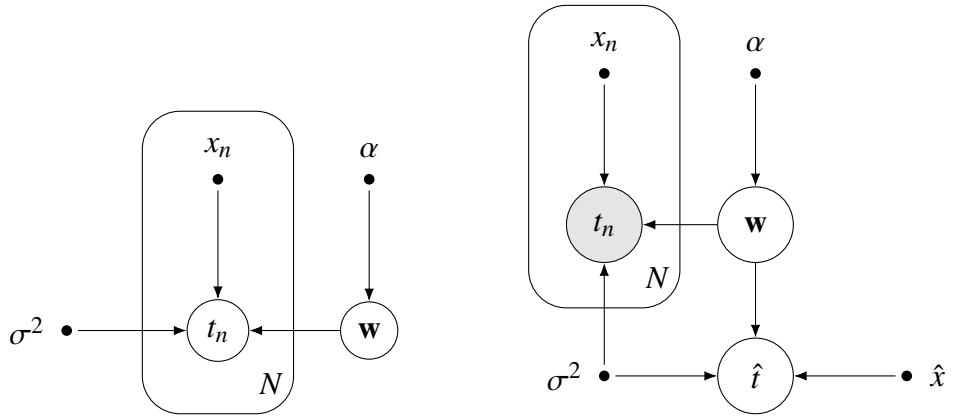
Another useful feature that we desire to have in a graphical model representation is a way to "compress" a sequence of variables. For instance, assume that we have a set of variables $\mathbf{y} = \{y_n\}_{n\in\{1,2,\dots,N\}}$ that depends on the same variable x so that the joint probability is

$$p(\mathbf{y}, x) = \prod_{n=1}^{N} p(y_n|x). \tag{E.9}$$

Using the concepts introduced so far, the graphical representation of the right hand side of (E.9) would be the one depicted in Figure E.4a. We can instead draw a single node y_n surrounded by a box denoting the number N of variables y_n, obtaining the graphical representation depicted inf Figure E.4b.

Finally, the last feature we aim to insert in the graphical model representation is the presence of deterministic parameters. These are denoted with small black dots.

Example E.1. Let us consider a polynomial regression problem. The polynomial coefficients are given by the vector $\mathbf{w}$, and we denote the observed data $\mathbf{t} = \{t_1, t_2, \dots, t_N\}$. Moreover, we denote the input variables by $\mathbf{X} = \{x_1, x_2, \dots, x_N\}$. Furthermore, we assume that there is a (known) noise with variance σ^2, as well a hyper-parameter α representing the precision of the Gaussian prior over $\mathbf{w}$. The



(a) Graphical Model representation of the polynomial regression (E.10).

(b) Graphical Model representation of the polynomial regression (E.11).

Figure E.5: Graphical Model representation of Bayesian Polynomial Regression. On the left, the outputs t_n are not observed. On the right, the outputs t_n are observed, and the prediction $\hat{t}$ is made.

joint probability, thus, reads

$$p(\mathbf{t}, \mathbf{w}|\mathbf{X}, \alpha, \sigma^2) = p(\mathbf{w}|\alpha) \prod_{n=1}^{N} p(t_n|\mathbf{w}, x_n, \sigma^2). \tag{E.10}$$

The graphical model of this representation is given in Figure E.5a.
Suppose now that we observed the outputs t_n, and that a new input value $\hat{x}$ is observed, and we are interested in the corresponding probability of the output $\hat{t}$. The corresponding joint distribution is given by

$$p(\hat{t}, \mathbf{t}, \mathbf{w}|\hat{x}, \mathbf{X}, \alpha, \sigma^2) = \left(\prod_{n=1}^{N} p(t_n|x_n \mathbf{w}, \sigma^2) \right) p(\mathbf{w}|\alpha) p(\hat{t}|\hat{X}, \mathbf{w}, \sigma^2), \tag{E.11}$$

whose graphical representation is given in Figure E.5b.

E.2.1. Conditional Independence

An important concept for probability distributions over multiple variables is that of *conditional independence*.

Definition E.5 (Conditional Independence)**.** Consider three random variables x_1, x_2, x_3. If the conditional distribution of x_1 given x_2 and x_3 is such that it does not depend on x_2, that is

$$p(x_1|x_2, x_3) = p(x_1|x_3), \tag{E.12}$$

we say that x_1 is *conditional independent* of x_2 given x_3.

This can be expressed in a different way. Consider the joint distribution of x_1 and x_2 conditioned to x_3. By using the product rule of probability and (E.12) we can write it as

$$p(x_1, x_2|x_3) = p(x_1|x_2, x_3)\, p(x_2|x_3) = p(x_1|x_3)\, p(x_2|x_3).$$

This says that variables x_1 and x_2 are statistically independent given x_3. This is denoted as

$$x_1 \perp\!\!\!\perp x_2 \,|\, x_3.$$

In general, given an expression for the joint distribution over a set of variables in terms of a product of conditional distributions, we could test whether any potential conditional independence property holds by repeated applications of the rules of probability. However, such an approach may result in long and tedious computations. An important feature of graphical models is that conditional independence properties of the joint distribution can be read directly from the graph, using the concept of *d-separation*.
Consider a general directed graph in which A, B, and C are three non-intersecting sets of nodes. Furthermore, consider all paths from any node in A to any node of B. Any such path is said to be *blocked by C* if it includes a node such that either:

1. The arrows on the path meet either head-to-tail or tail-to-tail at a node in C, or

2. The arrows meet head-to-head at a node such that neither it nor any of its descendants are in C.

If all paths from A to B are blocked, then A is d-separated from B by C, and the joint distributions over all the variables in the graph satisfy $A \perp\!\!\!\perp B|C$.

For instance, consider the graph in Figure E.6. In such graph, the path from x_1 to x_4 is not blocked by x_5. Indeed, the arrows of the path meet head-to-head in x_3, but the descendant of x_3 is x_5; and they meet tail-to-tail in x_2. Thus, conditional independence $x_1 \perp\!\!\!\perp x_4 \,|\, x_5$ does not follow from the graph. On the other hand, the path from x_1 to x_4 is blocked by x_2 since the arrow meets tail-to-tail. Thus, conditional independence $x_1 \perp\!\!\!\perp x_4 \,|\, x_2$ follows.

E.3. Expectation Maximization

The *Expectation Maximization* (EM) algorithm is a general strategy to find maximum likelihood solutions for probabilistic models having latent and observed variables. Let us denote by $\mathbf{Y}$ and $\mathbf{Z}$, respectively, the observed and latent variables;

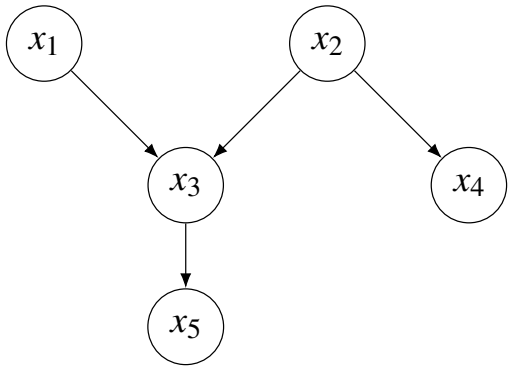


Figure E.6: Example of the concept of d-separation: the path from x_1 to x_4 is blocked by x_2, but not from x_5 (see the text for further details).

and by Θ the set of parameters of the model, so that we can define the *likelihood function*

$$p(\mathbf{Y}|\Theta) = \int_{\mathcal{Z}} p(\mathbf{Y}, \mathbf{Z}|\Theta)\, d\mathbf{Z}. \tag{E.13}$$

We remark that, in the case of discrete variables, the integral is replaced by the summation. We will always use the integral since it is more general.
The goal of the EM algorithm is to compute parameters Θ in order to maximize (E.13). We assume that the maximization of $p(\mathbf{Y}|\Theta)$ is untractable, while maximization of the complete data likelihood $p(\mathbf{Y}, \mathbf{Z}|\Theta)$ is significantly easier. We now introduce the following decomposition.

Proposition E.3. *By introducing a distribution $q(\mathbf{Z})$ defined over the latent variables, the following decomposition holds*

$$\log p(\mathbf{Y}|\Theta) = \mathcal{L}(q, \Theta) + \nabla_{KL}(q\|p), \tag{E.14}$$

where we defined the lower bound *and the* Kullback-Leibler *(KL) divergence as, respectively,*

$$\mathcal{L}(q, \mathbf{\Theta}) = \int_{\mathcal{Z}} q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Y}, \mathbf{Z}|\Theta)}{q(\mathbf{Z})}\right) d\mathbf{Z}, \tag{E.15a}$$

$$\nabla_{KL}(q\|p) = -\int_{\mathcal{Z}} q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Z}|\mathbf{Y}, \Theta)}{q(\mathbf{Z})}\right) d\mathbf{Z}. \tag{E.15b}$$

Proof. To prove the statement, let us compute the sum in (E.14):

$$\begin{aligned}
\log p(\mathbf{Y}|\Theta) &= \mathcal{L}(q, \Theta) + \nabla_{\mathrm{KL}}(q\|p) \\
&= \int_{\mathcal{Z}} q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Y}, \mathbf{Z}|\Theta)}{q(\mathbf{Z})}\right) d\mathbf{Z} - \int_{\mathcal{Z}} q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Z}|\mathbf{Y}, \Theta)}{q(\mathbf{Z})}\right) d\mathbf{Z} \\
&= \int_{\mathcal{Z}} \left(q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Y}, \mathbf{Z}|\Theta)}{q(\mathbf{Z})}\right) - q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Z}|\mathbf{Y}, \Theta)}{q(\mathbf{Z})}\right) \right) d\mathbf{Z}
\end{aligned}$$

$$= \int_{\mathcal{Z}} q(\mathbf{Z}) \left(\log\left(\frac{p(\mathbf{Y}, \mathbf{Z}|\Theta)}{q(\mathbf{Z})} \right) - \log\left(\frac{p(\mathbf{Z}|\mathbf{Y}, \Theta)}{q(\mathbf{Z})} \right) \right) d\mathbf{Z} \tag{E.16}$$

By the law of logarithm $\log(a/b) = \log(a) - \log(b)$, we can write (E.16) as

$$\int_{\mathcal{Z}} q(\mathbf{Z}) \left(\log p(\mathbf{Y}, \mathbf{Z}|\Theta) \right) - \log(q(\mathbf{Z})) - \log(p(\mathbf{Z}|\mathbf{Y}, \Theta)) + \log(q(\mathbf{Z})) \right) d\mathbf{Z},$$

which can be simplified in (by using again the law of logarithm)

$$\int_{\mathcal{Z}} q(\mathbf{Z}) \log\left(\frac{p(\mathbf{Y}, \mathbf{Z}|\boldsymbol{\Theta})}{p(\mathbf{Z}|\mathbf{Y}, \boldsymbol{\Theta})} \right) d\mathbf{Z}. \tag{E.17}$$

By the product rule of probability (E.2), the argument of the logarithm in (E.17) can be written as

$$\begin{aligned} \frac{p(\mathbf{Y}, \mathbf{Z}|\boldsymbol{\Theta})}{p(\mathbf{Z}|\mathbf{Y}, \boldsymbol{\Theta})} &= \frac{p(\mathbf{Z}|\mathbf{Y}, \boldsymbol{\Theta}) p(\mathbf{Y}|\boldsymbol{\Theta})}{p(\mathbf{Z}|\mathbf{Y}, \boldsymbol{\Theta})} \\ &= p(\mathbf{Y}|\boldsymbol{\Theta}). \end{aligned}$$

Thus, since this last term does not depend on $\mathbf{Z}$, the integral (E.17) can be written as

$$\begin{aligned} \int_{\mathcal{Z}} q(\mathbf{Z}) \log(p(\mathbf{Y}|\boldsymbol{\Theta})) \, d\mathbf{Z} &= \log(p(\mathbf{Y}|\boldsymbol{\Theta})) \int_{\mathcal{Z}} q(\mathbf{Z}) \, d\mathbf{Z} \\ &= \log(p(\mathbf{Y}|\boldsymbol{\Theta})), \end{aligned}$$

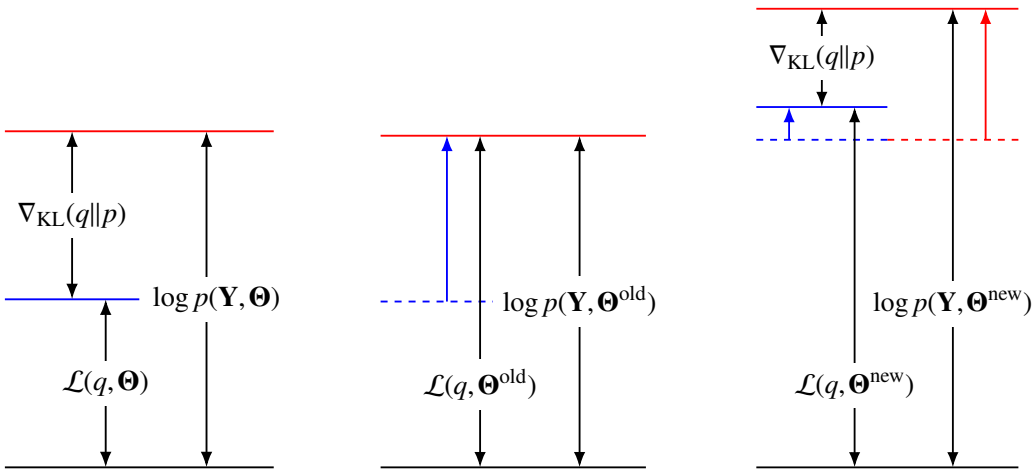
where last identity follows from the fact that q is a probability distribution, thus $\int_{\mathcal{Z}} q(\mathbf{Z}) \, d\mathbf{Z} = 1$. □

The Kullback-Leibler divergence (E.15b) satisfies $\nabla_{\mathrm{KL}}(q||p) \geq 0$, and the identity holds if and only if $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{Y}, \Theta)$. Therefore, $\mathcal{L}(q, \Theta) \leq p(\mathbf{Z}|\mathbf{Y}, \Theta)$, from which the name "lower bound". Identity (E.14) is depicted in Figure E.7a.

The EM algorithm is a two-stage iterative optimization technique for finding maximum likelihood solutions. Suppose that the current value for the set of parameters is given by Θ^{old}.

During the *Expectation step* (E-step), the lower bound $\mathcal{L}(q, \Theta^{\mathrm{old}})$ is maximized w.r.t. $q(\mathbf{Z})$, while maintaining Θ^{old} fixed. We remark that, since the right hand side of (E.14) does not depend on $q(\mathbf{Z})$, the lower bound is maximized when the KL-divergence vanishes, i.e. when $q(\mathbf{Z})$ equals $p(\mathbf{Z}|\mathbf{Y}, \Theta^{\mathrm{old}})$. In Figure E.7b, the E-step is schematized.

During the subsequent *Maximization step* (M-step), the distribution $q(\mathbf{Z})$ is kept fixed, and the lower bound is maximized w.r.t. Θ, giving a value Θ^{new}. This will cause the lower bound to increase (if not already at maximum), which will cause the log-likelihood to increase as well. Because the distribution q is determined using old parameters and is kept fixed during the M-step, $q(\mathbf{Z}) \neq p(\mathbf{Z}|\mathbf{Y}, \Theta^{\mathrm{new}})$, and



(a) Identity (E.14). (b) Illustration of the E-step. (c) Illustration of the M-step.

Figure E.7: Illustration of the EM algorithm. In Figure E.7a the decomposition (E.14) is shown. In Figure E.7b the E-step is illustrated: the distribution q is chosen in such a way to make the KL divergence vanish, while maintaining the same set of parameters Θ^{old}. In Figure E.7c, q is fixed, and a new set of parameters Θ^{new} is chosen so to increase the lower bound, which result in an increase of the distribution $p(\mathbf{Y},\Theta^{\mathrm{new}})$, as well as a new, non-zero, KL divergence.

hence there will be a non-zero KL divergence $\nabla_{\mathrm{KL}}(q\|p) \neq 0$. The increase in the log-likelihood is, therefore, *bigger* than the increase in lower bound. In Figure E.7c, the M-step is schematized.
Afterwards, the old set of parameters Θ^{old} is substituted with the new set Θ^{new} and both steps are re-iterated until convergence.

We now recover a different formulation of the EM algorithm. By substituting $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{Y},\Theta^{\mathrm{old}})$ in (E.15a), we see that, after the E-step, the lower bound takes the form

$$\begin{aligned}\mathcal{L}(q,\Theta) &= \int_{\mathcal{Z}} p(\mathbf{Z}|\mathbf{Y},\Theta^{\mathrm{old}}) \log p(\mathbf{Y},\mathbf{Z}|\Theta) - \int_{\mathcal{Z}} p(\mathbf{Z}|\mathbf{Y},\Theta^{\mathrm{old}}) \log p(\mathbf{Y},\mathbf{Z}|\Theta^{\mathrm{old}}) \\ &= Q(\Theta,\Theta^{\mathrm{old}}) + \text{const},\end{aligned}$$

where 'const' collects all the terms that does not depend on Θ.
Thus, in general, the EM algorithm aim at maximizing the function $Q(\Theta,\Theta^{\mathrm{old}})$, which is the expectation of the logarithm of the complete data likelihood, as function of parameters Θ, w.r.t. the distribution over the latent variables, as function of the old set of parameters Θ^{old}:

$$Q(\Theta,\Theta^{\mathrm{old}}) = \mathbb{E}_{p(\mathbf{Z}|\mathbf{Y},\Theta^{\mathrm{old}})}[\log p(\mathbf{Y},\mathbf{Z}|\Theta)]. \tag{E.18}$$

Algorithm E.1 summarizes the Expectation Maximization algorithm.

Algorithm E.1 EM Algorithm

Input: Initial set of parameters Θ^{old}, observations $\mathbf{Y}$, tolerance `tol`.

Output: New set of parameters Θ^{new}

Set `convergence` to `False`
while `not convergence` **do**
    Compute the likelihood $p(\mathbf{Y}|\Theta^{\text{old}})$
▷ Expectation step
    Compute the latent variable likelihood w.r.t Θ^{old}:

$$p(\mathbf{Z}|\mathbf{Y}, \Theta^{\text{old}})$$

▷ Maximization step
    Compute Θ^{new} as

$$\Theta^{\text{new}} = \arg\max_{\Theta} \mathbb{E}_{p(\mathbf{Z}|\mathbf{Y},\Theta^{\text{old}})}[\log p(\mathbf{Y}, \mathbf{Z}|\Theta)]$$

    Compute the likelihood $p(\mathbf{Y}|\Theta^{\text{new}})$
    if $|p(\mathbf{Y}|\Theta^{\text{old}}) - p(\mathbf{Y}|\Theta^{\text{new}})| <$ `tol` **then**
        Set `convergence` to `True`
    end if
    Set $\Theta^{\text{old}} = \Theta^{\text{new}}$
end while

E.4. Gibbs Sampling

In this Section, we introduce the *Gibbs Sampling* [115, 58, 113] algorithm which allows to easily sample variables $\mathbf{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_m\}$. The goal of the Gibbs sampling is to draw samples that are distributed like a general joint distribution

$$p(\mathbf{\Phi}) = p(\phi_1, \phi_2, \ldots, \phi_m). \tag{E.19}$$

A set of T samples is denoted by $\{\hat{\mathbf{\Phi}}^{(t)} = \{\hat{\phi}_i^{(t)}\}_{i=1}^m\}_{t=1}^T$. The assumption behind Gibbs sampling is that sampling directly from the joint distribution (E.19) is intractable, while it is easy to sample from the conditional distributions

$$p(\phi_j|\{\phi_i\}_{i\neq j}), \tag{E.20}$$

for $j = 1, 2, \ldots, m$. The idea of Gibbs sampling is to keep sampling from the conditional distributions (E.20) changing the index j. The values in set $\{\phi_i\}_{i\neq j}$ are updated with the last sampled value.

Gibbs Sampling, summarized in Algorithm E.2 is a *Markov Chain Monte Carlo* (MCMC) method [115]. This means that sample $\hat{\mathbf{\Phi}}^{(t)}$ is function only of the previ-

Algorithm E.2 Gibbs Sampler

Input: Initial samples $\hat{\mathbf{\Phi}}^{(0)} = \{\hat{\phi}_1^{(0)}, \hat{\phi}_2^{(0)}, \ldots, \hat{\phi}_m^{(0)}\}$, number of samples T.

Output: Set of samples $\{\hat{\mathbf{\Phi}}^{(t)} = \{\hat{\phi}_i^{(t)}\}_{i=1}^m\}_{t=1}^T$.

1: **for** $t = 0, 1, \ldots, T-1$ **do**
2: $\quad \hat{\phi}_1^{(t+1)} \sim p(\phi_1 | \hat{\phi}_2^{(t)}, \hat{\phi}_3^{(t)}, \ldots, \hat{\phi}_m^{(t)})$
3: $\quad \hat{\phi}_2^{(t+1)} \sim p(\phi_2 | \hat{\phi}_1^{(t+1)}, \hat{\phi}_3^{(t)}, \ldots, \hat{\phi}_m^{(t)})$
4: $\quad \vdots$
5: $\quad \hat{\phi}_i^{(t+1)} \sim p(\phi_i | \hat{\phi}_1^{(t+1)}, \ldots, \hat{\phi}_{i-1}^{(t+1)}, \hat{\phi}_{i+1}^{(t)} \ldots, \hat{\phi}_m^{(t)})$
6: $\quad \vdots$
7: $\quad \hat{\phi}_m^{(t+1)} \sim p(\phi_m | \hat{\phi}_1^{(t+1)}, \hat{\phi}_2^{(t+1)}, \ldots, \hat{\phi}_{m-1}^{(t+1)})$
8: **end for**

Algorithm E.3 Gibbs Sampler for Latent State Models

Input: Initial parameters estimate $\widehat{\Theta}^{(0)}$, number of iterations T, observed sequence $\mathbf{Y}$.

Output: Final estimate of parameters $\widehat{\Theta}^{(T)}$ and latent mode sequence $\mathbf{Z}^{(T)}$.

1: **for** $t = 0, 1, \ldots, T-1$ **do**
2: $\quad \widehat{\mathbf{Z}}^{(t+1)} \sim p(Z | \widehat{\Theta}^{(t)}, Y)$
3: $\quad \widehat{\Theta}^{(t+1)} \sim p(\Theta | \widehat{Z}^{(t+1)}, Y)$
4: **end for**

ous sample$\hat{\mathbf{\Phi}}^{(t-1)}$. Convergence in MCMC methods means that the Markov chain $\hat{\mathbf{\Phi}}^{(t)}$ has a unique stationary distribution, which is equal to the joint distribution (E.19). This means that the average $\frac{1}{T}\sum_{t=1}^T f(\mathbf{\Phi}^{(t)})$ converges to the expectation $\int f(\mathbf{\Phi}) p(\mathbf{\Phi})$ almost surely as $T \to \infty$ for every bounded function f. Moreover, the Markov chain can be shown to converge to the stationary distribution from any point in the domain of the distribution, meaning that Algorithm E.2 can be initialized with any reasonable starting point $\hat{\mathbf{\Phi}}^{(0)}$.

A two-stage Gibbs sampler is a particular case in which the set $\mathbf{\Phi}$ is composed of two elements. It is of particular interest in the case of hidden variable models, such as Hidden Markov Models. Indeed, let us consider a latent variable model defined by the set of parameters Θ. We aim at maximizing the joint distribution of the set of parameters Θ and the latent mode sequence $\mathbf{Z}$ given the observed sequence $\mathbf{Y}$, $p(\Theta, \mathbf{Z}|\mathbf{Y})$. The two-stage Gibbs sampler summarized in Algorithm E.3. The main advantage of using Gibbs sampler in the case of latent variable models is that it can be initialized both with a set of parameters, as presented in Algorithm E.3, or with an initial guess of the latent mode sequence $\mathbf{Z}^{(0)}$. In this second case, steps 2 and 3 of Algorithm E.3 are run in inverse order.

APPENDIX F

K-MEANS ALGORITHM

In this Appendix, we present a recall on the *k-means clustering algorithm*.
k-means is one of the most popular clustering algorithms. Clustering is a technique that allows finding groups of similar objects, that is to subdivide a set of objects into groups so that elements in the group are more related to other elements of the same group than objects of other groups [112].
k-means is a *prototype-based* clustering algorithm. That means that each cluster (i.e. group) is represented by a prototype, which, in the *k*-means case, is the *centroid* of the group.

The algorithm starts by randomly picking k centroids. Then, it keeps updating the clusters and the centroids until convergence. The clusters are given by assigning each sample to the nearest centroid. The centroids are updated as the average of the data in the same cluster. The algorithm is stopped when there is no change in the cluster assignment or a user-defined tolerance or a maximum number of iteration is reached.
The procedure is summarized in Algorithm F.1.

We said that the *k*-means clustering algorithm aims at grouping objects so that elements in the same group are similar. We define the *similarity* between two elements $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ as the opposite of the distance $\|\mathbf{x} - \mathbf{y}\|$. We can describe the *k*-means clustering algorithm as an iterative approach for minimizing the *within-cluster sum of squared errors* (SSE), also called *cluster inertia*

$$\iota(\mathbf{X}, \boldsymbol{\mu}) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \sum_{j=}^{k} \omega_i^j \left\|\mathbf{x}_i - \boldsymbol{\mu}_j\right\|^2,$$

where $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,2,\ldots,N}$ is the dataset on which we aim to perform the clustering,

Algorithm F.1 k-means.

Input: Data set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,2,...,N}$, number of clusters k.

Output: Centroids $\{\boldsymbol{\mu}_j\}_{j=1,...,k}$ and clusters $\{\mathbf{C}_j\}_{j=1,...,k}$

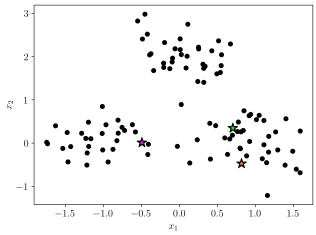
1: Randomly select initial centroids $\{\boldsymbol{\mu}_j\}$ from the data-set $\mathbf{X}$.
2: **repeat**
3: Define cluster $\mathbf{C}_j$ by assigning each element $\mathbf{x}_i$ to the nearest centroid $\boldsymbol{\mu}_j$.
4: Update the centroid $\boldsymbol{\mu}_j$ as the average of the point in cluster $\mathbf{C}_j$.
5: **until** convergence

$\boldsymbol{\mu} = \{\boldsymbol{\mu}_j\}_{j=1,2,...,k}$ is the set of cluster centroids, and ω_i^j is one if sample $\mathbf{x}_i$ is in cluster j, and it is zero otherwise.

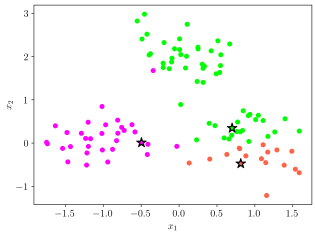
Figure F.1 shows the main steps of the k-means algorithm for $k = 3$ clusters for data in $\mathbb{R}^2$. In Figure F.1a the dataset and the first randomly selected centroids are shown. In Figure F.1b the clusters resulting from the initial selection of the centroids are shown. Figure F.1c shows how the given clusters result in an update of the centroids. Figure F.1d shows the clusters updated with the new centroids definition. Finally, Figure F.1e shows the final set of centroids and clusters.

So far, we discussed the classic k-means algorithm, in which the initial centroids are randomly placed. This may result in a bad final clustering or slow convergence. For this reason, we employ the *k-means* ++ variation [4] which leads to better and more consistent results than the classic k-means. The idea behind k-means++ is to place the initial centroids far away from each other. The first centroid is randomly selected from the dataset. Then, each other centroid is selected from the dataset with a probability proportional to the minimum distance from any other centroid. In this way, points far from the already selected centroids are more likely to be selected as the next centroids.

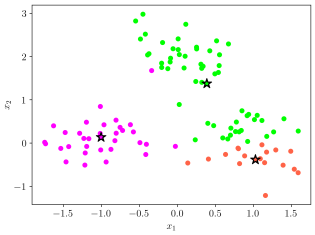
The k-means++ procedure is summarized in Algorithm F.2.



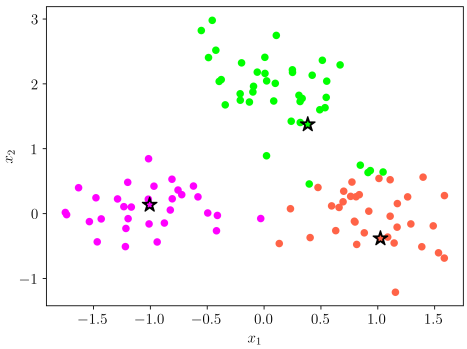
(a) Initial centroids.



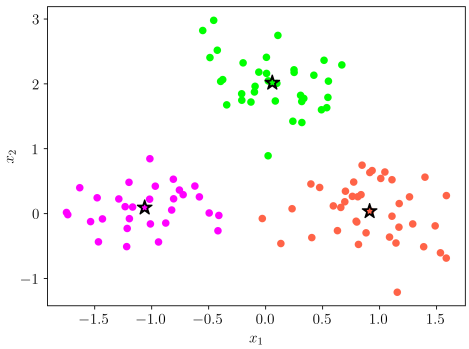
(b) First clustering.



(c) First update of the centroids.



(d) Re-computed cluster.



(e) Final clustering and centroids.

Figure F.1: Main steps of the *k*-means clustering algorithm. Dots represent the data points, and stars denote the centroids.

Algorithm F.2 k-means++.

Input: Data set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,2,\ldots,N}$, number of clusters k.

Output: Centroids $\{\boldsymbol{\mu}_j\}_{j=1,\ldots,k}$ and clusters $\{\mathbf{C}_j\}_{j=1,\ldots,k}$

▹ Initialize the centroids set

Initialize the centroid set $M = \emptyset$

Randomly select the first centroid $\boldsymbol{\mu}_1$ from $\mathbf{X}$ and add it to M.

for $p = 2, 3, \ldots, k$ **do**

    For each element in $\mathbf{X}$, compute the minimum squared distance $d(\mathbf{x}_j, M)^2$ to any of the centroids in M.

    Select the next centroid $\boldsymbol{\mu}_p$ by sampling from the discrete probability distribution defined as

$$\Pr(\boldsymbol{\mu}_p = \mathbf{x}_j) = \frac{d(\mathbf{x}_j, M)^2}{\sum_{i=1}^{N} d(\mathbf{x}_i, M)^2}.$$

end for

▹ Proceed as the classical k-menas

repeat

    Define cluster $\mathbf{C}_j$ by assigning each element $\mathbf{x}_i$ to the nearest centroid $\boldsymbol{\mu}_j$.

    Update the centroid $\boldsymbol{\mu}_j$ as the average of the point in cluster $\mathbf{C}_j$.

until convergence

BIBLIOGRAPHY

[1] J. Abonyi, R. Babuska, and F. Szeifert. Modified gath-geva fuzzy clustering for identification of takagi-sugeno fuzzy models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(5):612–621, 2002.

[2] F. J. Abu-Dakka and V. Kyrki. Geometry-aware dynamic movement primitives. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4421–4426. IEEE, 2020.

[3] M. Alvarez, J. R. Peters, N. D. Lawrence, and B. Schölkopf. Switched latent force models for movement segmentation. In *Advances in neural information processing systems*, pages 55–63, 2010.

[4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[5] P. Baldi, S. Brunak, and F. Bach. *Bioinformatics: the machine learning approach*. MIT press, 2001.

[6] M. Balduccini, M. Gelfond, R. Watson, and M. Nogueira. The usa-advisor: A case study in answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 439–442. Springer, 2001.

[7] P. Beeson and B. Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935. IEEE, 2015.

[8] J. A. Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10] P. Bogacki and L. F. Shampine. A 3 (2) pair of runge-kutta formulas. *Applied Mathematics Letters*, 2(4):321–325, 1989.

[11] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *CoRR*, abs/1709.07857, 2017.

[12] G. Brewka, I. Niemela, and M. Truszczynski. Preferences and nonmonotonic reasoning. *AI magazine*, 29(4):69–69, 2008.

[13] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21(12):1031–1052, 2002.

[14] J. C. Butcher and G. Wanner. Runge-kutta methods: some historical notes. *Applied Numerical Mathematics*, 22(1-3):113–151, 1996.

[15] M. Caliari and A. Ostermann. Implementation of exponential rosenbrock-type integrators. *Applied Numerical Mathematics*, 59(3-4):568–581, 2009.

[16] S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent service robotics*, 9(1):1–29, 2016.

[17] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard. Learning and reproduction of gestures by imitation. *IEEE Robotics Automation Magazine*, 17(2):44–54, June 2010.

[18] D. B. Camarillo, T. M. Krummel, and J. K. Salisbury Jr. Robotic technology in surgery: past, present, and future. *The American Journal of Surgery*, 188(4):2–15, 2004.

[19] Y.-C. Chen. Solving robot trajectory planning problems with uniform cubic b-splines. *Optimal Control Applications and Methods*, 12(4):247–262, 1991.

[20] J. Chiang, Z. J. Wang, and M. J. McKeown. A hidden markov, multivariate autoregressive (hmm-mar) network framework for analysis of surface emg (semg) data. *IEEE Transactions on Signal Processing*, 56(8):4069–4081, 2008.

[21] S. Chiappa and J. R. Peters. Movement extraction by detecting dynamics switches and repetitions. In *Advances in neural information processing systems*, pages 388–396, 2010.

[22] I. R. Cole. *Modelling CPV.* PhD thesis, Loughborough University, 2015.

[23] A. Colmerauer. An introduction to prolog iii. In *Computational Logic*, pages 37–79. Springer, 1990.

[24] F. Corcione, C. Esposito, D. Cuccurullo, A. Settembre, N. Miranda, F. Amato, F. Pirozzi, and P. Caiazzo. Advantages and limits of robot-assisted laparoscopic surgery: preliminary experience. *Surgical Endoscopy and Other Interventional Techniques*, 19(1):117–119, 2005.

[25] A. Darzi and Y. Munz. The impact of minimally invasive surgical techniques. *Annu. Rev. Med.*, 55:223–237, 2004.

[26] G. De Giacomo and M. Vardi. Synthesis for ltl and ldl on finite traces. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[27] A. De Luca, L. Lanari, and G. Oriolo. A sensitivity approach to optimal spline robot trajectories. *Automatica*, 27(3):535–539, 1991.

[28] F. Despinoy, D. Bouget, G. Forestier, C. Penet, N. Zemiti, P. Poignet, and P. Jannin. Unsupervised trajectory segmentation for surgical gesture recognition in robotic training. *IEEE Transactions on Biomedical Engineering*, 63(6):1280–1291, 2016.

[29] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[30] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *European Conference on Planning*, pages 169–181. Springer, 1997.

[31] Z. Dogmus, G. Gezici, V. Patoglu, and E. Erdem. Developing and maintaining an ontology for rehabilitation robotics. In *KEOD*, pages 389–395, 2012.

[32] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[33] Y. Ephraim and W. J. Roberts. Revisiting autoregressive hidden markov modeling of speech signals. *IEEE Signal processing letters*, 12(2):166–169, 2005.

[34] B. R. Fajen and W. H. Warren. Behavioral dynamics of steering, obstable avoidance, and route selection. *Journal of Experimental Psychology: Human Perception and Performance*, 29(2):343, 2003.

[35] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. C. Teppan. Industrial applications of answer set programming. *KI-KUnstliche Intelligenz*, 32(2-3):165–176, 2018.

[36] M. Fard, S. Ameri, R. Chinnam, and R. Ellis. Soft boundary approach for unsupervised gesture segmentation in robotic-assisted surgery. *IEEE Robotics and Automation Letters*, 2(1):171–178, 2016.

[37] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[38] A. Gams, B. Nemec, A. J. Ijspeert, and A. Ude. Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics*, 30(4):816–830, Aug 2014.

[39] Y. Gao, S. S. Vedula, C. E. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Béjar, D. D. Yuh, et al. Jhu-isi gesture and skill assessment working set (JIGSAWS): A surgical activity dataset for human motion modeling. In *MICCAI Workshop: M2CAI*, volume 3, page 3, 2014.

[40] A. Gasparetto and V. Zanotto. A new method for smooth trajectory planning of robot manipulators. *Mechanism and machine theory*, 42(4):455–471, 2007.

[41] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to gringo, clasp, clingo, and iclingo. 2008.

[42] M. Gebser, P. Obermeier, T. Schaub, M. Ratsch-Heitmann, and M. Runge. Routing driverless transport vehicles in car assembly with answer set programming. *Theory and Practice of Logic Programming*, 18(3-4):520–534, 2018.

[43] M. Ginesi, D. Meli, A. Calanca, D. Dall'Alba, N. Sansonetto, and P. Fiorini. Dynamic movement primitives: Volumetric obstacle avoidance. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 234–239, Dec 2019.

[44] M. Ginesi, D. Meli, H. C. Nakawala, A. Roberti, and P. Fiorini. A knowledge-based framework for task automation in surgery. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 37–42, Dec 2019.

[45] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, and P. Fiorini. Autonomous task planning and situation awareness in robotic surgery. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3144–3150, 2020.

[46] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, and P. Fiorini. Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions. *Journal of Intelligent & Robotic Systems*, 101(4):79, Mar 2021.

[47] M. Ginesi, N. Sansonetto, and P. Fiorini. Overcoming some drawbacks of dynamic movement primitives. *Robotics and Autonomous Systems*, 144:103844, 2021.

[48] X. Guan, R. Raich, and W.-K. Wong. Efficient multi-instance learning for activity recognition from time series data using an auto-regressive hidden markov model. In *International Conference on Machine Learning*, pages 2330–2339, 2016.

[49] T. Guber. A translational approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–229, 1993.

[50] G. Guthart and J. Salisburg. The intuitive telesurgery system: overview and application. w: Proceedings of 2000 ieee international conference; 2000, on robots and automation. san francisco, 22-28 april 2000, 2000.

[51] T. Haidegger, P. Kazanzides, I. Rudas, B. Benyó, and Z. Benyó. The importance of accuracy measurement standards for computer-integrated interventional systems. In *EURON GEM Sig Workshop on The Role of Experiments in Robotics Research at IEEE ICRA*, pages 1–6, 2010.

[52] W. R. Hamilton. *Elements of quaternions*. Longmans, Green, & Company, 1866.

[53] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numer.*, 19(May):209–286, 2010.

[54] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal. Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2587–2592. IEEE, 2009.

[55] M. Hong and J. W. Rozenblit. Modeling of a transfer task in computer assisted surgical training. In *Proceedings of the Modeling and Simulation in Medicine Symposium*, pages 1–6, 2016.

[56] R. Huang, H. Cheng, H. Guo, Q. Chen, and X. Lin. Hierarchical interactive learning for a human-powered augmentation lower exoskeleton. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 257–263. IEEE, 2016.

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

[58] N. H. Hudson. *Inference in Hybrid Systems with Applications in Neural Prosthetics*. PhD thesis, California Institute of Technology, Pasadena, California, 2009.

[59] Y. K. Hwang, N. Ahuja, et al. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

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

[61] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1398–1403. IEEE, 2002.

[62] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554, 2003.

[63] A. J. Ijspeert, J. Nakanishi, T. Shibata, and S. Schaal. Nonlinear dynamical systems for imitation with humanoid robots. In *Proceedings of the IEEE/RAS International Conference on Humanoids Robots (Humanoids2001)*, number CONF, pages 219–226, 2001.

[64] E. Jang, S. Vijayanarasimhan, P. Pastor, J. Ibarz, and S. Levine. End-to-end learning of semantic grasping. *CoRR*, abs/1707.01932, 2017.

[65] F. Jelinek. *Statistical methods for speech recognition*. MIT press, 1997.

[66] R. P. Joshi, N. Koganti, and T. Shibata. Robotic cloth manipulation for clothing assistance task using dynamic movement primitives. In *Proceedings of the Advances in Robotics*, pages 1–6. 2017.

[67] B.-H. Juang and L. Rabiner. Mixture autoregressive hidden markov models for speech signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(6):1404–1413, 1985.

[68] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505. IEEE, 1985.

[69] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International Journal of Robotics Research*, 20(10):1–15, 2016.

[70] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International Journal of Robotics Research*, 36(13-14):1595–1618, 2017.

[71] O. Kroemer, H. Van Hoof, G. Neumann, and J. Peters. Learning to predict phases of manipulation tasks as hidden states. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4009–4014. IEEE, 2014.

[72] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden markov models in computational biology. applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531, 1994.

[73] W. Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

[74] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. Temporal convolutional networks for action segmentation and detection. *CoRR*, abs/1611.05267, 2016.

[75] C. Lea, G. D. Hager, and R. Vidal. An improved model for segmentation and recognition of fine-grained activities with application to surgical training tasks. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 1123–1129. IEEE, 2015.

[76] C. Lea, A. Reiter, R. Vidal, and G. D. Hager. Efficient segmental inference for spatiotemporal modeling of fine-grained actions. *CoRR*, abs/1602.02995, 2016.

[77] B. G. Leroux and M. L. Puterman. Maximum-penalized-likelihood estimation for independent and markov-dependent mixture models. *Biometrics*, pages 545–558, 1992.

[78] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.

[79] X. Li, M. Parizeau, and R. Plamondon. Training hidden markov models with multiple observations-a combinatorial method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4):371–377, 2000.

[80] V. Lifschitz. Answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 373–374. Springer, 1999.

[81] C. Lin, P. Chang, and J. Luh. Formulation and optimization of cubic polynomial joint trajectories for industrial robots. *IEEE Transactions on automatic control*, 28(12):1066–1074, 1983.

[82] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters. Probabilistic segmentation applied to an assembly task. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 533–540. IEEE, 2015.

[83] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters. Learning movement primitive libraries through probabilistic segmentation. *The International Journal of Robotics Research*, 36(8):879–894, 2017.

[84] M. J. Mack. Minimally invasive and robotic surgery. *Jama*, 285(5):568–572, 2001.

[85] L. MacKenzie, J. Ibbotson, C. Cao, and A. Lomax. Hierarchical decomposition of laparoscopic surgery: a human factors approach to investigating the operating room environment. *Minimally Invasive Therapy & Allied Technologies*, 10(3):121–127, 2001.

[86] E. Magid, D. Keren, E. Rivlin, and I. Yavneh. Spline-based robot navigation. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2296–2301. IEEE, 2006.

[87] C. Manning and H. Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[88] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning to sequence movement primitives from demonstrations. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4414–4421. IEEE, 2014.

[89] L. Markus. Asyptotically autonomous differential systems, contributions to the theory of nonlinear oscillations, vol. iii, s. lefschetz, ed, 1956.

[90] T. Matsubara, S.-H. Hyon, and J. Morimoto. Learning stylistic dynamic movement primitives from multiple demonstrations. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1277–1283. Citeseer, 2010.

[91] T. Matsubara, S.-H. Hyon, and J. Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural networks*, 24(5):493–500, 2011.

[92] H. Mayer, I. Nagy, D. Burschka, A. Knoll, E. U. Braun, R. Lange, and R. Bauernschmitt. Automation of manual tasks for minimally invasive surgery. In *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*, pages 260–265. IEEE, 2008.

[93] H. Mayer, I. Nagy, A. Knoll, E. U. Braun, R. Bauernschmitt, and R. Lange. Haptic feedback in a telepresence system for endoscopic heart surgery. *Presence: Teleoperators and Virtual Environments*, 16(5):459–470, 2007.

[94] F. Meier, E. Theodorou, F. Stulp, and S. Schaal. Movement segmentation using a primitive library. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3407–3412. IEEE, 2011.

[95] V. S. Mellarkod and M. Gelfond. Enhancing asp systems for planning with temporal constraints. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 309–314. Springer, 2007.

[96] G. P. Moustris, S. C. Hiridis, K. M. Deliparaschos, and K. M. Konstantinidis. Evolution of autonomous and semi-autonomous robotic surgical systems: a

review of the literature. *The international journal of medical robotics and computer assisted surgery*, 7(4):375–392, 2011.

[97] A. Murali, A. Garg, S. Krishnan, F. Pokorny, P. Abbeel, T. Darrell, and K. Goldberg. TSC-DL:Unsupervised trajectory segmentation of multi-modal surgical demonstrations with deep learning. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[98] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Goldberg. Learning by observation for surgical subtasks: Multilateral cutting of 3d viscoelastic and 2d orthotropic tissue phantoms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1202–1209. IEEE, 2015.

[99] R. Nag, K. Wong, and F. Fallside. Script recognition using hidden markov models. In *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 2071–2074. IEEE, 1986.

[100] T. D. Nagy and T. Haidegger. An open-source framework for surgical subtask automation. 2018.

[101] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

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

[103] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 91–98. IEEE, 2008.

[104] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.

[105] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015.

[106] S. Puls, J. Graf, H. Wörn, and I. Maurtua. Cognitive robotics in industrial environments. In *Human Machine Interaction-Getting Closer*, pages 213–234. InTech, 2012.

[107] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[108] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[109] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807. IEEE, 1993.

[110] A. Rai, F. Meier, A. Ijspeert, and S. Schaal. Learning coupling terms for obstacle avoidance. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 512–518. IEEE, 2014.

[111] A. Rai, G. Sutanto, S. Schaal, and F. Meier. Learning feedback terms for reactive planning and control. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2184–2191. IEEE, 2017.

[112] S. Raschka. *Python Machine Learning*. Packt Publishing, 2015.

[113] P. Resnik and E. Hardisty. Gibbs sampling for the uninitiated. Technical report, Maryland Univ College Park Inst for Advanced Computer Studies, 2010.

[114] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, Oct 1992.

[115] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2005.

[116] A. Roberti, N. Piccinelli, D. Meli, R. Muradore, and P. Fiorini. Improving rigid 3-d calibration for robotic surgery. *IEEE Transactions on Medical Robotics and Bionics*, 2(4):569–573, 2020.

[117] E. Rohmer, S. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework.

[118] M. Saveriano, F. Franzel, and D. Lee. Merging position and orientation motion primitives. In *International Conference on Robotics and Automation (ICRA), 2019*, 2019.

[119] S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[120] S. Schaal, P. Mohajerian, and A. Ijspeert. Dynamics systems vs. optimal control — a unifying view. In P. Cisek, T. Drew, and J. F. Kalaska, editors, *Computational Neuroscience: Theoretical Insights into Brain Function*, volume 165 of *Progress in Brain Research*, pages 425 – 445. Elsevier, 2007.

[121] R. Schaback. A practical guide to radial basis functions. *Electronic Resource*, 11, 2007.

[122] A. Sidiropoulos and Z. Doulgeri. A reversible dynamic movement primitive formulation. *arXiv preprint arXiv:2010.07708*, 2020.

[123] A. Soares Júnior, B. N. Moreno, V. C. Times, S. Matwin, and L. d. A. F. Cabral. Grasp-uts: an algorithm for unsupervised trajectory segmentation. *International Journal of Geographical Information Science*, 29(1):46–68, 2015.

[124] N. J. Soper and G. M. Fried. The fundamentals of laparoscopic surgery: its time has come. *Bull Am Coll Surg*, 93(9):30–32, 2008.

[125] G. Sutanto, Z. Su, S. Schaal, and F. Meier. Learning sensor feedback models from demonstrations via phase-modulated neural networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1142–1149. IEEE, 2018.

[126] C. J. Taylor and D. J. Kriegman. Minimization on the lie group so (3) and related manifolds. *Yale University*, 16(155):6, 1994.

[127] A. Ude, B. Nemec, T. Petrić, and J. Morimoto. Orientation in cartesian space dynamic movement primitives. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2997–3004. IEEE, 2014.

[128] A. Vakanski, I. Mantegh, A. Irish, and F. Janabi-Sharifi. Trajectory learning for robot programming by demonstration using hidden markov model and dynamic time warping. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(4):1039–1052, 2012.

[129] S. S. Vedula, A. O. Malpani, L. Tao, G. Chen, Y. Gao, P. Poddar, N. Ahmidi, C. Paxton, R. Vidal, S. Khudanpur, et al. Analysis of the structure of surgical activity for a suturing and knot-tying task. *PloS one*, 11(3):e0149174, 2016.

[130] T. J. Vidovszky, W. Smith, J. Ghosh, and M. R. Ali. Robotic cholecystectomy: learning curve, advantages, and limitations. *Journal of Surgical Research*, 136(2):172–178, 2006.

[131] I. Visser. Seven things to remember about hidden markov models: A tutorial on markovian models for time series. *Journal of Mathematical Psychology*, 55(6):403–415, 2011.

[132] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[133] R. Volpe. *Real and artificial forces in the control of manipulators: theory and experiments*. PhD thesis, PhD thesis, Carnegie Mellon University, Department of Physics, 1990.

[134] R. Wang, Y. Wu, W. L. Chan, and K. P. Tee. Dynamic movement primitives plus: For enhanced reproduction quality and efficient trajectory modification using truncated kernels and local biases. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3765–3771. IEEE, 2016.

[135] Y. Wang and G. S. Chirikjian. A new potential field method for robot path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 977–982. IEEE, 2000.

[136] E. W. Weisstein. Euler angles. 2009.

[137] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics*, 4(1):389–396, 1995.

[138] T. Xuan. *Autoregressive hidden Markov model with application in an El Nino study*. PhD thesis, Master Thesis, University of Saskatchewan, 2004.

[139] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–86. IEEE, 2017.

[140] G.-Z. Yang, J. Cambias, K. Cleary, E. Daimler, J. Drake, P. E. Dupont, N. Hata, P. Kazanzides, S. Martel, R. V. Patel, et al. Medical robotics—regulatory, ethical, and legal considerations for increasing levels of autonomy. *Science Robotics*, 2(4):8638, 2017.

[141] O. I. Zhelezov. N-dimensional rotation matrix generation algorithm. *American Journal of Computational and Applied Mathematics*, 7(2):51–57, 2017.