Motion Representations for Learning Robots and Human-Robot Interaction

Polynomial Splines and Movement Primitives

Aleš Ude

Jožef Stefan Institute, Ljubljana
Faculty of Electrical Enginnering, University of Ljubljana

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani
COBISS.SI-ID 222235395
ISBN 978-961-243-475-5 (PDF)

URL: https://www.ijs.si/usr/aude/teaching/motion_reps.pdf

Naslov: Motion Representations for Learning Robots and Human-Robot Interaction
Avtor: Aleš Ude
Založnik: Založba FE, Ljubljana,
Izdajatelja: Fakuleta za elektrotehniko, Ljubljana
Institut Jožef Stefan, Ljubljana
Urednik: prof. dr. Sašo Tomažič
Recenzenta: Matjaž Mihelj, Andrej Gams
Kraj in leto izida: Ljubljana, 2025
1. elektronska izdaja

Preface

The performance of complex tasks in robotics often involves executing a variety of movements that require distinct control policies. For instance, using a tool might involve picking it up, positioning it appropriately, and performing a task – all of which demand separate motion trajectories. This textbook focuses on the specification of robot motion, bridging the gap between task-level planning and precise motion execution.

Traditional industrial robots often rely on straightforward linear or circular movements, whereas robots in unstructured environments require more complex and adaptive trajectories. Representing and learning such motions presents a significant challenge, as they are difficult to program directly using conventional methods. Consequently, learning from demonstration and reinforcement learning have become key methodologies for acquiring flexible motion representations. These methods not only enable robots to replicate demonstrated behaviors but also allow them to adapt learned skills to novel scenarios. This way their versatility can be enhanced.

In dynamic environments, static time-parameterized representations, such as polynomials and splines, are often insufficient due to their inability to accommodate interruptions or environmental changes. To mitigate this, motion representations based on autonomous dynamical systems have been developed, eliminating the dependency on time as a trajectory parameter. These representations ensure that desired motions remain adaptable to variations in task conditions, enabling smooth and safe operation even in unpredictable scenarios. The ability to dynamically adapt trajectories based on real-time feedback has become a cornerstone of modern robotics, ensuring both safety and task success in diverse environments.

The ability to encode multiple trajectories into a unified representation is another critical advancement in robotics. By leveraging data from multiple demonstrations, robots can learn generalized motion patterns that adapt to new situations. This requires combining computational efficiency with the ability to encode variability and uncertainty. Methods like probabilistic and dynamic movement primitives address this by providing parametric representations that support on-the-fly adaptations to changes in the environment or task objectives. Moreover, these representations facilitate the integration of motion planning and control into a cohesive framework, bridging the gap between abstract task planning and physical execution.

This textbook introduces a range of modern motion representation techniques, including polynomial splines, dynamic movement primitives (DMPs), probabilistic movement

primitives (ProMPs), and dynamic systems based on Gaussian mixture models (GMMs). These representations address the limitations of static trajectory planning, enabling adaptation to dynamic environments and variability in task requirements. Computational methods for learning and parameterizing these representations are presented. The emphasis is on their role in encoding complex robot skills. Practical examples and case studies illustrate how these techniques are applied in real-world scenarios. Readers are provided with a clear understanding of their strengths and limitations.

The materials are in part based on extensive research on motion representations at the Department of Automatics, Biocybernetics, and Robotics at Jožef Stefan Institute. They are intended for advanced students and researchers in robotics. The text requires a foundational understanding of mathematics, control engineering, and robotics. It serves as an essential resource for those wishing to deepen their knowledge of motion representation and learning in robotics. By providing a comprehensive overview of state-of-the-art methodologies, the text aims to equip readers with the tools to address current and future challenges in robot motion control and learning.

Contents

Chapter 1

Preliminaries

When specifying robot motion, we distinguish between the concepts of path and trajectory. A *path* specifies the robot's spatial motion without considering its time course, i.e. the time at which the robot reaches a point on the path remains undefined. The path becomes a *trajectory* once the timing is determined, i.e. the time at which the robot reaches each point on the path is specified. The term *path planning* refers to finding the spatial path of robot motion from the given initial position to the final position while satisfying spatial constraints, e.g. avoiding obstacles. If the path is specified, the term *trajectory planning* refers to finding the optimal velocity and acceleration profiles for the given path that satisfy time constraints, e.g. velocity and acceleration limits. However, the spatial course and timing of the trajectory can also be determined simultaneously, in which case the term trajectory planning refers to the specification of a complete robot trajectory. Finally, *robot control* deals with accurately tracking the specified trajectory with a real (physical) robot in all controllable degrees of freedom (DOF).

Trajectory planning can be used to generate short-term, continuous motions for robot manipulators. Such motions are often called *movement, motion or motor primitives*. They are considered the basic building blocks of robot behaviors. Generally speaking, motion primitives are the building blocks from which complex robot skills are generated by querying, sequencing and superposition of movement primitives. Here the term *robot skill* refers to an entire class of short-term motion plans that can be used to achieve the desired result, e.g. move the robot hand to different desired end-positions and orientations along a certain type of reaching trajectory, throwing motions so that an object hits the desired target locations, etc.

More complex behaviors such as cooking, assembling a product, making up the room, etc. are never planned as one super long trajectory or skill but are rather constructed from a sequential execution of multiple movement primitives or skills. The problem of finding an optimal sequence of robotic operations / skills to solve a given task is referred to as *task planning*.

In this textbook we focus on representations that can be used for the specification of movement primitives and on data-driven approaches for the learning of robot skills

constructed from movement primitives. Most of the motion representations discussed in this textbook have built-in mechanisms to specify complex skills from a combination of movement primitives. We primarily present optimizaton-based and statistical approaches and omit other approaches such as reinforcement learning, which would require another book to cover adequately.

1.1 Control policies

To analyze what kind of representation is most suitable to drive the motion of a robot in an unstructured, dynamic environment, we analyze the controllers used to track the desired joint space motion trajectory, which is for example specified as a superposition of basis functions with parameters $\boldsymbol{\alpha}$ (see Chapter 2). A proportional derivative (PD) controller may be used for this purpose

$$\Delta\mathbf{y}(\mathbf{y}, \boldsymbol{\alpha}, t) = \mathbf{K}_y(\mathbf{y}_d(t) - \mathbf{y}) + \mathbf{K}_{\dot{y}}(\dot{\mathbf{y}}_d(t) - \dot{\mathbf{y}}). \tag{1.1}$$

Here $\Delta\mathbf{y} \in \mathbb{R}^n$ is the computed change of the joint angle positions. It depends on the desired and current robot configuration and velocity $\mathbf{y}$, $\dot{\mathbf{y}} \in \mathbb{R}^n$, respectively. $\mathbf{K}$, $\mathbf{K}_d \in \mathbb{R}^{n\times n}$ are respectively the positional and velocity gains, $\mathbf{y}_d(t)$, $\dot{\mathbf{y}}_d(t) \in \mathbb{R}^n$ are the desired robot configuration and velocity at time t, and n is the number of robot degrees of freedom. If specified as a linear superposition of basis functions, the desired trajectory is computed as

$$\mathbf{y}_d(t) = \sum_i \boldsymbol{\alpha}_i \mathrm{B}_i(t), \tag{1.2}$$

$$\dot{\mathbf{y}}_d(t) = \sum_i \boldsymbol{\alpha}_i \dot{\mathrm{B}}_i(t), \tag{1.3}$$

where B_i are the basis functions used to encode the desired trajectory and $\boldsymbol{\alpha}_i$ from Eq. (1.1) specifies a time-dependent control policy that the robot follows to realize the desired motion.

Formulation (1.1) works well for robots operating in known environments where no disturbances occur. However, as explained in [31], such control policies are not very flexible. For example, if a robot arm is wiping a glass and is held stationary for a few seconds, the PD controller will try to catch up, i.e. reduce the discrepancy between the desired position $\mathbf{y}_d(t)$ and the actual robot position $\mathbf{y}$, by the shortest possible path once the robot arm is released. This can cause the robot to attempt moving through the glass, which will not result in successful task execution, may trigger the robot's safety mechanism to abort the motion, and potentially cause damage either to the environment or to the robot.

Thus a more flexible approach is needed to plan the motions in dynamic environments. An alternative is offered by autonomous dynamic systems, where the change of state is

computed using differential equation

$$\dot{\mathbf{y}} \;=\; \mathbf{f}(\mathbf{y}, \boldsymbol{\alpha}). \tag{1.4}$$

Given the current state of the robot $\mathbf{y}$ and the control parameters $\boldsymbol{\alpha}$, the above system is used to compute the desired change of robot configuration, $\Delta\mathbf{y} = \dot{\mathbf{y}}\Delta t$, and other kinematic target quantities, e.g. the desired velocity and acceleration. The computed target values can then be used by robot controllers to compute the motor commands [31]. This way it is easier to deal with perturbations in dynamic environments.

The above system is called autonomous because it removes the explicit time dependency from motion planning. When motion is computed according to (1.4), the system automatically deals with perturbations like when the robot is prevented from moving because the same target values are computed by Eq. (1.4) as long as the robot state $\mathbf{y}$ does not change. In addition, in some cases it is possible to build into Eq. (1.4) mechanisms that deal with changes to the desired final position of the robot, avoidance of robot joint limits, obstacle avoidance, motion speed scaling, etc.

1.2 Motion specification in Cartesian space

A rigid body is a solid object in which the distances between any two points remain constant, ensuring that the object does not deform and maintains its shape and size throughout its motion. In robotics, we are typically interested in motion of solid objects, such as the link of a robot manipulator or an object a robot manipulates. Rigid body motion can be divided into translation, where every point in the object moves uniformly in the same direction, and rotation, where the object turns about a fixed axis. Importantly, during rigid body motion, both the distances between any two points and the cross product of any two vectors within the object are preserved, maintaining the relative angles and orientations of vectors.

The specification of robot motion in 3-D Cartesian space involves the specification of the position and orientation of the robots's end-effector at any moment in time. The coordinate frame of the robot's end-effector is usually attached to the tool center point. The robot end-effector position is defined as the coordinates $\mathbf{p} = [x, y, z]^{\mathrm{T}} \in \mathbb{R}^3$ of the frame attached to the robot's tool center point in the robot's base system. The end-effector orientation is defined as the instantaneous rotation of the coordinate frame attached to the tool center point with respect to the robot's base frame. Unlike the position, the robot's orientation cannot be represented as a 3-D vector because the set of all orientations is not a vector space. Since any rotation can be characterized by a 3×3 rotation matrix, the space of all orientations can be represented by rotation matrices, which form a special orthogonal group SO(3)

$$\mathrm{SO}(3) = \left\{ \mathbf{R} \in \mathbb{R}^{3\times 3},\ \mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I},\ \det(\mathbf{R}) = 1 \right\}. \tag{1.5}$$

While SO(3) is not a vector space, it forms a group with respect to matrix multiplication. It is also a real three dimensional manifold [22]. Thus planning the orientational robot motion is equivalent to trajectory planning in SO(3). Every orientation can be identified with a unique $\mathbf{R} \in \mathrm{SO}(3)$. Any orientation trajectory on time interval $[0, T]$ can be written as $\mathbf{R}(t) \in \mathrm{SO}(3),\ 0 \leq t \leq T$.

As a 3-D manifold, SO(3) can be parameterized by three parameters, e. g. Euler angles. However, such parameterizations always contain singularities. It turns out that there exists no minimal (3-D) representation of orientation that 1. contains no singularities, i. e. a continuous orientational motion in 3-D space is continuous also in the 3-D parameter space, and 2. is continuously differentiable, i. e. the partial derivatives of the parameters with respect to any differential rotation, at any orientation, are finite.

It is preferable to use parameterizations that do not introduce artificial discontinuities not existing in the real world into motion planning. As outlined above, such representations of orientation have more than three parameters and must therefore fulfill additional constraints in a higher dimensional space to be guaranteed to lie on the constraint manifold. In the following we study two such representations: rotation matrices and quaternions.

1.2.1 Representation of orientation by rotation matrices

The position of a point $\mathbf{p}(t)$ on a rigid body that is rotating about some axis can be expressed as a function of the initial position vector $\mathbf{p}_0$ and a rotation matrix $\mathbf{R}(t)$

$$\mathbf{p}(t) = \mathbf{R}(t)\mathbf{p}_0. \tag{1.6}$$

To find the angular velocity $\boldsymbol{\omega}(t)$, we take the time derivative of the position vector $\mathbf{p}(t)$:

$$\dot{\mathbf{p}}(t) = \dot{\mathbf{R}}(t)\mathbf{p}_0. \tag{1.7}$$

Since $\mathbf{R}(t)\mathbf{R}(t)^{\mathrm{T}} = \mathbf{I}$, we can compute $\dot{\mathbf{R}}(t)\mathbf{R}(t)^{\mathrm{T}} + \mathbf{R}\dot{\mathbf{R}}^{\mathrm{T}} = 0$, We obtain $\dot{\mathbf{R}}(t)\mathbf{R}(t)^{\mathrm{T}} = -\mathbf{R}(t)\dot{\mathbf{R}}(t)^{\mathrm{T}} = -\left(\dot{\mathbf{R}}(t)\mathbf{R}(t)^{\mathrm{T}}\right)^{\mathrm{T}}$, which means that

$$\boldsymbol{\Omega}(t) = \dot{\mathbf{R}}(t)\mathbf{R}(t)^{\mathrm{T}} \tag{1.8}$$

is a skew symmetric matrix, i.e. $\boldsymbol{\Omega}(t)^{\mathrm{T}} = -\boldsymbol{\Omega}(t)$. Any skew symmetric matrix can be written as

$$\boldsymbol{\Omega}(t) = [\boldsymbol{\omega}(t)]_\times = \begin{bmatrix} 0 & -\omega_z(t) & \omega_y(t) \\ \omega_z(t) & 0 & -\omega_x(t) \\ -\omega_y(t) & \omega_x(t) & 0 \end{bmatrix},\ \boldsymbol{\omega}(t) = \begin{bmatrix} \omega_x(t) \\ \omega_y(t) \\ \omega_z(t) \end{bmatrix}. \tag{1.9}$$

We can write

$$\dot{\mathbf{p}}(t) = \dot{\mathbf{R}}(t)\mathbf{p}_0 = \dot{\mathbf{R}}(t)\mathbf{R}(t)^{\mathrm{T}}\mathbf{p}(t) = \boldsymbol{\Omega}(t)\mathbf{p}(t) = \boldsymbol{\omega}(t) \times \mathbf{p}(t). \tag{1.10}$$

$\boldsymbol{\omega}(t)$ defines the angular velocity of a rotational motion.

If a rigid body rotates with constant angular velocity $\boldsymbol{\omega}$, we obtain a homogeneous linear differential equation with constant coefficients

$$\dot{\mathbf{p}}(t) = \boldsymbol{\omega} \times \mathbf{p}(t) = [\boldsymbol{\omega}]_\times \mathbf{p}(t). \tag{1.11}$$

It is well known from calculus that such equations have an analytic solution given by a *matrix exponential map*,

$$\mathbf{p}(t) = e^{t[\boldsymbol{\omega}]_\times}\, \mathbf{p}(0), \tag{1.12}$$

where $\mathbf{p}(0)$ is the initial point position. By comparing Eq. (1.6) and (1.12), we obtain the following expression for rotation with constant angular velocity $\boldsymbol{\omega}$

$$\begin{aligned} \mathbf{R}(t) &= e^{[\boldsymbol{\omega} t]_\times} && (1.13)\\ &= e^{[\boldsymbol{\omega} t]_\times} = \mathbf{I} + [\boldsymbol{\omega} t]_\times + \frac{1}{2!}[\boldsymbol{\omega} t]_\times^2 + \frac{1}{3!}[\boldsymbol{\omega} t]_\times^3 + \dots && (1.14) \end{aligned}$$

where (1.14) is obtained using the definition of the matrix exponential.

We next show that the angular velocity $\boldsymbol{\omega}$ is equal to the product of rotation angle θ in unit time and rotation axis $\mathbf{n}$, i.e. $\boldsymbol{\omega} = \theta\mathbf{n}$. Under rotational motion, the points along the axis of rotation do not move. We can deduce from Eq. (1.11) that this is the case for points along $\boldsymbol{\omega}$ since $\boldsymbol{\omega} \times \boldsymbol{\omega} = 0$. Thus the angular velocity vector $\boldsymbol{\omega}$ is aligned with the axis of rotation, $\mathbf{n} = \boldsymbol{\omega}/\|\boldsymbol{\omega}\|$. Let's also define $\theta = \|\boldsymbol{\omega}\|$. Assuming small θ and using Rodrigues' formula (1.16), we obtain the following approximation for $\mathbf{R}(\mathbf{n}, \theta)$

$$\mathbf{R}(\mathbf{n}, \theta) \approx \mathbf{I} + \theta[\mathbf{n}]_\times.$$

We can then calculate the positional change as follows

$$\|\Delta\mathbf{p}\| = \|\mathbf{p}(t) - \mathbf{p}_0\| = \|(\mathbf{I} + \theta[\mathbf{n}]_\times)\mathbf{p}_0 - \mathbf{p}_0\| = \theta\|\mathbf{n} \times \mathbf{p}_0\| = \theta\|\mathbf{p}_0\| \sin(\beta),$$

where β is the angle between the vectors $\mathbf{n}$ and $\mathbf{p}_0$ and we applied the formula $\|\mathbf{n} \times \mathbf{p}_0\| = \|\mathbf{n}\|\|\mathbf{p}_0\| \sin(\beta)$. For the points orthogonal to the axis of rotation, we have $\beta = \pi/2$ and we obtain $\|\Delta\mathbf{p}\| = \theta\|\mathbf{p}_0\|$. For the point $\mathbf{p}_0$ on a circular path around the origin, the positional change is equal to $\theta\|\mathbf{p}_0\|$, which shows that θ is indeed the rotation angle.

From the angular velocity we can thus compute the axis of rotation $\mathbf{n} = [n_x, n_y, n_z]^{\mathrm{T}} = \boldsymbol{\omega}/\|\boldsymbol{\omega}\|$ and the angle $\theta = \|\boldsymbol{\omega}\|$ travelled per time unit. The rotation about axis $\mathbf{n}$ by angle θ in unit time can then be expressed by

$$\mathbf{R}(\mathbf{n}, \theta) = e^{[\boldsymbol{\omega}]_\times} = e^{\theta[\mathbf{n}]_\times} = \mathbf{I} + \theta[\mathbf{n}]_\times + \frac{\theta^2}{2!}[\mathbf{n}]_\times^2 + \frac{\theta^3}{3!}[\mathbf{n}]_\times^3 + \dots \tag{1.15}$$

Using the fact that for a unit axis vector $\mathbf{n}$ it holds $[\mathbf{n}]_\times^3 = -[\mathbf{n}]_\times$ and using the formulae for Taylor series expansion of sine and cosine, the exponential map (1.15) can be computed by Rodrigues' formula

$$\begin{aligned} e^{\theta[\mathbf{n}]_\times} &= \mathbf{I} + \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots\right)[\mathbf{n}]_\times + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} - \dots\right)[\mathbf{n}]_\times^2 \\ &= \mathbf{I} + \sin(\theta)[\mathbf{n}]_\times + (1 - \cos(\theta))\,[\mathbf{n}]_\times^2. \end{aligned} \tag{1.16}$$

It can be shown also algebraically (see [22], page 29) that $\mathbf{R}(\mathbf{n}, \theta) \in \mathbb{R}^{3\times 3}$ is a rotation matrix, i.e. it is orthogonal, $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$, and has a determinant equal to 1.

The exponential map (1.15) is surjective, i.e. for any rotation matrix $\mathbf{R}$ there exists unit axis $\mathbf{n}$ and angle $0 \leq \theta \leq \pi$ so that Eq. (1.16) holds (see Section 1.2.3 for a constructive proof). Furthermore, it is easy although somewhat tedious to verify that equation (1.16) can be rewritten as

$$\mathbf{R}(\mathbf{n}, \theta) = \begin{bmatrix} n_x^2(1-c)+c & n_x n_y(1-c) - n_z s & n_x n_z(1-c) + n_y s \\ n_y n_x(1-c) + n_z s & n_y^2(1-c)+c & n_y n_z(1-c) - n_x s \\ n_z n_x(1-c) - n_y s & n_z n_y(1-c) + n_x s & n_z^2(1-c)+c \end{bmatrix}, \tag{1.17}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$.

1.2.2 Quaternions

The parametrization of rotation matrices by exponential map defined in equation (1.15) shows that the space of all orientations is three-dimensional. Since rotation matrices have 9 parameters, it can be cumbersome to use them to represent orientation trajectories. An alternative representation is offered by quaternions, a generalization of complex numbers developed by William Rowan Hamilton.

A quaternion $\mathbf{q}$ is an ordered four-parameter vector that consists of a scalar part $v \in \mathbb{R}$ and vector part $\mathbf{u} = [u_x, u_y, u_z]^{\mathrm{T}} \in \mathbb{R}^3$. The space of all quaternions $\mathbb{H}$ forms an algebra, i.e. it is a vector space with multiplication $*$. The addition of two quaternions is defined as usually

$$\begin{bmatrix} v_1 \\ \mathbf{u}_1 \end{bmatrix} + \begin{bmatrix} v_2 \\ \mathbf{u}_2 \end{bmatrix} = \begin{bmatrix} v_1 + v_2 \\ \mathbf{u}_1 + \mathbf{u}_2 \end{bmatrix}. \tag{1.18}$$

By interpreting real numbers $v \in \mathbb{R}$ as quaternions with zero vector part v and real 3-D vectors $\mathbf{u} \in \mathbb{R}^3$ as quaternions with zero scalar part, we can write

$$\begin{bmatrix} v \\ \mathbf{u} \end{bmatrix} = v + \mathbf{u}. \tag{1.19}$$

The distinguishing property of quaternions is that we can define a quaternion multiplica-

tion

$$\begin{bmatrix} v_1 \\ \mathbf{u}_1 \end{bmatrix} * \begin{bmatrix} v_2 \\ \mathbf{u}_2 \end{bmatrix} = \begin{bmatrix} v_1 v_2 - \mathbf{u}_1^{\mathrm{T}} \mathbf{u}_2 \\ v_1 \mathbf{u}_2 + v_2 \mathbf{u}_1 + \mathbf{u}_1 \times \mathbf{u}_2 \end{bmatrix}. \tag{1.20}$$

Note that since in general $\mathbf{u}_1 \times \mathbf{u}_2 \neq \mathbf{u}_2 \times \mathbf{u}_1$, the quaternion multiplication is not commutative, i.e. $\mathbf{q}_1 * \mathbf{q}_2 \neq \mathbf{q}_2 * \mathbf{q}_1$. It is, however, easy to check that quaternion multiplication is associate, i.e. $\mathbf{q}_1 * (\mathbf{q}_2 * \mathbf{q}_3) = (\mathbf{q}_1 * \mathbf{q}_2) * \mathbf{q}_3$. The inverse of quaternion $\mathbf{q} \neq 0$ with respect to multiplication $*$ is given by

$$\mathbf{q}^{-1} = \begin{bmatrix} v \\ \mathbf{u} \end{bmatrix}^{-1} = \begin{bmatrix} v/(v^2 + \|\mathbf{u}\|^2) \\ -\mathbf{u}/(v^2 + \|\mathbf{u}\|^2) \end{bmatrix}. \tag{1.21}$$

The conjugation of a quaternion is defined as

$$\bar{\mathbf{q}} = \overline{\begin{bmatrix} v \\ \mathbf{u} \end{bmatrix}} = \begin{bmatrix} v \\ -\mathbf{u} \end{bmatrix}. \tag{1.22}$$

Note that for unit quaternions $\mathbf{q}$, $\|\mathbf{q}\| = \sqrt{\mathbf{q} * \bar{\mathbf{q}}} = \sqrt{v^2 + \|\mathbf{u}\|^2} = 1$, thus conjugation and inverse are in this case equivalent, i.e. $\bar{\mathbf{q}} = \mathbf{q}^{-1}$.

Unit quaternions are a subset of all quaternions $\mathbf{q} \in \mathbb{H}$ such that $\|\mathbf{q}\| = 1$. They form a unit sphere S^3 in $\mathbb{R}^4$. Let's interpret an arbitrary point $\mathbf{p} \in \mathbb{R}^3$ as quaternion with zero scalar part. Such quaternions are called pure quaternions. For any unit quaternion $\mathbf{q}$, we define the following operator

$$\mathrm{L}_{\mathbf{q}}(\mathbf{p}) = \mathbf{q} * \mathbf{p} * \bar{\mathbf{q}}. \tag{1.23}$$

We can write

$$\begin{aligned}
\mathbf{q} * \mathbf{p} * \bar{\mathbf{q}} &= \begin{bmatrix} v \\ \mathbf{u} \end{bmatrix} * \begin{bmatrix} 0 \\ \mathbf{p} \end{bmatrix} * \begin{bmatrix} v \\ -\mathbf{u} \end{bmatrix} = \begin{bmatrix} -\mathbf{u}^{\mathrm{T}} \mathbf{p} \\ v\mathbf{p} + \mathbf{u} \times \mathbf{p} \end{bmatrix} * \begin{bmatrix} v \\ -\mathbf{u} \end{bmatrix} \\
&= \begin{bmatrix} -v\mathbf{u}^{\mathrm{T}} \mathbf{p} + -v\mathbf{p}^{\mathrm{T}} \mathbf{u} \\ (\mathbf{u}^{\mathrm{T}} \mathbf{p})\mathbf{u} + v(v\mathbf{p} + \mathbf{u} \times \mathbf{p}) - v\mathbf{p} \times \mathbf{u} - (\mathbf{u} \times \mathbf{p}) \times \mathbf{u} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ (\mathbf{u}^{\mathrm{T}} \mathbf{p})\mathbf{u} + v^2 \mathbf{p} + 2v\mathbf{u} \times \mathbf{p} - ((\mathbf{u}^{\mathrm{T}} \mathbf{u})\mathbf{p} - (\mathbf{p}^{\mathrm{T}} \mathbf{u})\mathbf{u}) \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 2(\mathbf{u}^{\mathrm{T}} \mathbf{p})\mathbf{u} + v^2 \mathbf{p} + 2v\mathbf{u} \times \mathbf{p} + (v^2 - 1)\mathbf{p} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ \left(2\mathbf{u}\mathbf{u}^{\mathrm{T}} + (2v^2 - 1)\mathbf{I} + 2v[\mathbf{u}]_{\times}\right)\mathbf{p} \end{bmatrix}.
\end{aligned} \tag{1.24}$$

From (1.24) we can see that just like $\mathbf{p}$, $\mathrm{L}_{\mathbf{q}}(\mathbf{p})$ is also a pure quaternion. We can write

$$\mathrm{L}_{\mathbf{q}}(\mathbf{p}) = \begin{bmatrix} 2(v^2+u_x^2)-1 & 2(u_xu_y - vu_z) & 2(u_xu_z + vu_y) \\ 2(u_xu_y + vu_z) & 2(v^2+u_y^2)-1 & 2(u_yu_z - vu_x) \\ 2(u_xu_z - vu_y) & 2(u_yu_z + vu_x) & 2(v^2+u_z^2)-1 \end{bmatrix} \mathbf{p}. \tag{1.25}$$

Let's assume $0 \leq \theta < 2\pi$ and $\mathbf{n} = [n_x, n_y, n_z]^{\mathrm{T}} \in \mathbb{R}^3$, $\|\mathbf{n}\| = 1$. We define a quaternion $\mathbf{q}(\mathbf{n}, \theta) \in \mathbb{H}$ as follows

$$\mathbf{q}(\mathbf{n}, \theta) = \begin{bmatrix} v \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} v \\ u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)n_x \\ \sin(\theta/2)n_y \\ \sin(\theta/2)n_z \end{bmatrix}. \tag{1.26}$$

We can easily see that $\|\mathbf{q}(\mathbf{n}, \theta)\| = \sqrt{\cos^2(\theta/2) + \sin^2(\theta/2)\mathbf{n}^{\mathrm{T}}\mathbf{n}} = 1$, thus $\mathbf{q}(\mathbf{n}, \theta)$ is a unit quaternion. Moreover, for any unit quaternion $\mathbf{q} = [v, \mathbf{u}^{\mathrm{T}}]^{\mathrm{T}}$, we can compute $\theta = 2\arccos(v)$, $\mathbf{n} = \mathbf{u}/\|\mathbf{u}\|$ (with $\mathbf{n} = 0$ for the special case of $v = 1$, $\mathbf{u} = 0$). Thus the map $\mathbf{q}(\mathbf{n}, \theta)$ is surjective, i.e. for any unit quaternion $\mathbf{q}$ there exists θ, $\mathbf{n}$ so that $\mathbf{q} = \mathbf{q}(\mathbf{n}, \theta)$.

This means that for any quaternion $\mathbf{q}$ that defines the operator $\mathrm{L}_{\mathbf{q}}$ in Eq. (1.25), there exists $\mathbf{n}, \theta$ so that $\mathbf{q} = \mathbf{q}(\mathbf{n}, \theta)$. Let's write

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{1.27}$$

Using standard trigonometric formulae and algebraic manipulations, we now show that the matrix $\mathbf{R}(\mathbf{n}, \theta)$ from Eq. (1.17) is equal to the matrix specified in Eq. (1.25), where $\mathbf{q} = \mathbf{q}(\mathbf{n}, \theta)$ is defined as in equation (1.26). For example,

$$\begin{aligned} r_{11} &= n_x^2(1-\cos(\theta)) + \cos(\theta) = n_x^2(1 - (1 - 2\sin^2(\theta/2)) + 2\cos^2(\theta/2) - 1 \\ &= 2\left(\cos^2(\theta/2) + n_x^2\sin^2(\theta/2)\right) - 1 = 2(v^2 + u_x^2) - 1. \end{aligned} \tag{1.28}$$

The proof for other coefficients of $\mathbf{R}(\mathbf{n}, \theta)$ is similar. Thus

$$\mathrm{L}_{\mathbf{q}}(\mathbf{p}) = \mathbf{R}(\mathbf{n}, \theta)\mathbf{p}. \tag{1.29}$$

We observe that $\mathbf{q} * \mathbf{p} * \bar{\mathbf{q}}$ is equivalent to the rotation of $\mathbf{p}$ about the rotation axis $\mathbf{u}$ by angle θ. Thus just like rotation matrices, unit quaternions can be used to compute rotations. This expression also makes it clear that we can use quaternions to a new compute rotation from two successive rotations:

$$\mathbf{q}_2 * (\mathbf{q}_1 * \mathbf{p} * \bar{\mathbf{q}}_1) * \bar{\mathbf{q}}_2 = (\mathbf{q}_2 * \mathbf{q}_1) * \mathbf{p} * (\overline{\mathbf{q}_2 * \mathbf{q}_1}). \tag{1.30}$$

Thus the quaternion multiplication can be used to compute a new quaternion $\mathbf{q}_2 * \mathbf{q}_1$ combining two successive rotations. It follows that the unit quaternion $\Delta\mathbf{q}$ that rotates the starting orientation $\mathbf{q}_1$ to the target orientation $\mathbf{q}_2$ can be computed as follows

$$\Delta\mathbf{q} = \mathbf{q}_2 * \bar{\mathbf{q}}_1. \tag{1.31}$$

Unit quaternions provide a singularity-free, non-minimal representation of orientation (it has four instead of three parameters). However, this representation is not unique. This can be seen by observing that the matrix in Eq. (1.25) only contains quadratic terms. Thus unit quaternion $\mathbf{q} \in \mathrm{S}^3$ and its antipod $-\mathbf{q} \in \mathrm{S}^3$ are transformed into the same rotation matrix $\mathbf{R}(\mathbf{n}, \theta)$. Since

$$\cos((2\pi - \theta)/2) = \cos(\pi - \theta/2) = -\cos(\theta/2),$$
$$\sin((2\pi - \theta)/2)(-\mathbf{n}) = -\sin(\pi - \theta/2)\mathbf{n} = -\sin(\theta/2)\mathbf{n},$$

and using the fact that each unit quaternion pair $\mathbf{q}$ and $-\mathbf{q}$ represents the same orientation, we can conclude that the rotation about axis $-\mathbf{n}$ by angle $2\pi - \theta$ results in the same orientation as the rotation about axis $\mathbf{n}$ by angle θ. Thus unit quaternions provide a double covering of the space of all orientations.

We already know that using formula (1.25), we can transform any unit quaternion to its corresponding rotation matrix. To derive the formulae for mapping a rotation matrix to the corresponding quaternion, we equate the matrices in Eq. (1.27) and (1.25). Using simple algebraic manipulations and considering that $\|\mathbf{q}\|^2 = v^2 + u_x^2 + u_y^2 + u_z^2 = 1$, we obtain the following formulae

$$v = \pm\frac{1}{4}\sqrt{(r_{11} + r_{22} + r_{33} + 1)^2 + (r_{32} - r_{23})^2 + (r_{13} - r_{31})^2 + (r_{21} - r_{12})^2}, \tag{1.32}$$
$$u_x = s_x\frac{1}{4}\sqrt{(r_{32} - r_{23})^2 + (r_{11} - r_{22} - r_{33} + 1)^2 + (r_{21} + r_{12})^2 + (r_{31} + r_{13})^2}, \tag{1.33}$$
$$u_y = s_y\frac{1}{4}\sqrt{(r_{13} - r_{31})^2 + (r_{21} + r_{12})^2 + (r_{22} - r_{11} - r_{33} + 1)^2 + (r_{32} + r_{23})^2}, \tag{1.34}$$
$$u_z = s_z\frac{1}{4}\sqrt{(r_{21} - r_{12})^2 + (r_{31} + r_{13})^2 + (r_{32} + r_{23})^2 + (r_{33} - r_{11} - r_{22} + 1)^2}, \tag{1.35}$$

where $s_x, s_y, s_z \in \{-1, 1\}$ respectively determine the sign of u_x, u_y and u_z. This choice arises because of the sign ambiguity (unit quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotations and thus map onto the same rotation matrix). If we select the nonnegative values of v, i.e. $v \geq 0$, then s_x, s_y and s_z must be assigned the signs of $r_{32} - r_{23}$, $r_{13} - r_{31}$, and $r_{21} - r_{12}$, respectively. This mapping was developed by Cayley and has been shown to be superior to other mappings for the conversion of rotation matrices to unit quaternions used in robotics, both due to its simplicity and numerical stability.

1.2.3 Logarithmic map

The rotation matrix logarithm is defined as an inverse of the exponential map (1.15). Given rotation matrix $\mathbf{R}$ and using Eq. (1.26) and (1.32), we can compute the angle of rotation as follows

$$\theta = 2 \arccos \left(\frac{1}{4} \sqrt{(r_{11} + r_{22} + r_{33} + 1)^2 + (r_{32} - r_{23})^2 + (r_{13} - r_{31})^2 + (r_{21} - r_{12})^2} \right). \tag{1.36}$$

In the above formula we selected the positive value of v from Eq. (1.32). For $\theta \neq 0$, the unit axis of rotation can be obtained by normalizing the vector $\mathbf{u}$ as computed by Eq. (1.33) – (1.35), $\mathbf{n} = \mathbf{u}/\|\mathbf{u}\|$. The matrix logarithm can now be computed as follows

$$\log(\mathbf{R}) = \begin{cases} \begin{bmatrix} 0, & 0, & 0 \end{bmatrix}^{\mathrm{T}}, & \mathbf{R} = \mathbf{I} \\ \theta \boldsymbol{n}, & \text{otherwise} \end{cases}. \tag{1.37}$$

The components of $\log(\mathbf{R}) \in \mathbb{R}^3$ are called exponential coordinates of $\mathbf{R}$. This representation of orientation is commonly referred to as axis-angle representation. Note that $\|\log(\mathbf{R})\| = \|\theta \mathbf{n}\| = \theta$ because $\mathbf{n}$ is a unit vector and we use the non-negative value of v from Eq. (1.32) to compute θ, thus $0 \leq \theta \leq \pi$. For $\theta = \pi$ we cannot compute $\boldsymbol{n}$ using the above formula because in this case, the off-diagonal terms of rotation matrices do not provide information about the direction of rotation axis (which is nevertheless defined up to a sign ambiguity). Recall that the rotation about axis $\mathbf{n}$ by angle θ is equal to the rotation about axis $-\mathbf{n}$ by angle $2\pi - \theta$. For $\theta = \pi$. Thus that the rotations by angle π about axis $\mathbf{n}$ and axis $-\mathbf{n}$ are the same. However, we can still compute $\|\log(\mathbf{R})\| = \|\pm \pi \mathbf{n}\| = \pi$. Thus the norm $\|\log(\mathbf{R})\|$ is defined for all $\mathbf{R} \in \mathrm{SO}(3)$. It can be used to define a metric on SO(3)

$$\mathrm{d}(\mathbf{R}_1, \mathbf{R}_2) = \begin{cases} \|\log(\mathbf{R}_2 \mathbf{R}_1^{\mathrm{T}})\|, & \theta < \pi \\ \pi, & \text{otherwise} \end{cases}. \tag{1.38}$$

Note that $\mathrm{d}(\mathbf{I}, \mathbf{R}) = \|\log(\mathbf{R})\|$.

To define a logarithmic map for unit quaternions, we first define the quaternion exponential. Note that for any pure unit quaternion $\mathbf{n}$, i.e. unit quaternion with scalar part equal to 0, we can compute $\mathbf{n} * \mathbf{n} = -1$, i.e. its square is equal to quaternion -1, which has zero vector part. Let's define

$$\begin{aligned} \exp(\theta \mathbf{n}) &= 1 + \theta \mathbf{n} + \frac{\theta^2}{2!} \mathbf{n}^2 + \frac{\theta^3}{3!} \mathbf{n}^3 + \ldots = (1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \ldots) + (\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \ldots) \mathbf{n} \\ &= \cos(\theta) + \sin(\theta) \mathbf{n}. \end{aligned} \tag{1.39}$$

Here $\theta \mathbf{n}$ is interpreted as pure quaternion with zero scalar part. By comparing Eq. (1.26)

and (1.39) we can write

$$\mathbf{q}(\mathbf{n},\theta) = \exp\left(\frac{\theta}{2}\mathbf{n}\right) = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)\mathbf{n}. \tag{1.40}$$

For any unit quaternion $\mathbf{q} = \left[v, \mathbf{u}^{\mathrm{T}}\right]^{\mathrm{T}} \neq [-1,0,0,0]^{\mathrm{T}}$, its logarithm is defined as inverse of the exponential. It can easily be computed analytically from (1.40)

$$\log(\mathbf{q}) = \log\left(\begin{bmatrix} v \\ \mathbf{u} \end{bmatrix}\right) = \begin{cases} \arccos(v)\dfrac{\mathbf{u}}{\|\mathbf{u}\|}, & \|\mathbf{u}\| \neq 0 \\ \begin{bmatrix} 0, & 0, & 0 \end{bmatrix}^{\mathrm{T}}, & \text{otherwise} \end{cases}. \tag{1.41}$$

Note that unlike with rotation matrix logarithm, where several choices needed to be made regarding signs to compute the logarithm as defined in Eq. (1.37), there are no choices regarding signs to be made in Eq. (1.41). This makes it easier to use quaternions instead of rotation matrices for planning orientational motions. The logarithm exists for any unit quaternion except for $\mathbf{q} = -1$, where the rotation axis is undefined. Nevertheless, we can compute the angle θ for any unit quaternion including $\mathbf{q} = -1$, where $\theta = \arccos(-1) = \pi$.

Let's now assume that we have two quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ and compute the difference quaternion $\Delta\mathbf{q} = \mathbf{q}_2 * \overline{\mathbf{q}}_1$ according to (1.31). We have shown in Eq. (1.26) that there exists angle θ and axis $\mathbf{n}$ so that $\Delta\mathbf{q} = \exp(1/2\,\theta\mathbf{n})$. We can thus compute the angular velocity that takes $\mathbf{q}_1$ to $\mathbf{q}_2$ within unit time as $\boldsymbol{\omega} = \theta\mathbf{n}$. Using the quaternion logarithm (1.41), we can now compute the constant angular velocity that rotates unit quaternion $\mathbf{q}_1$ to unit quaternion $\mathbf{q}_2$ within unit time as follows

$$\boldsymbol{\omega} = \theta\mathbf{n} = 2\log(\exp(\theta\mathbf{n}/2)) = 2\log(\mathbf{q}_2 * \overline{\mathbf{q}}_1). \tag{1.42}$$

The quaternion logarithm can also be used to define a distance metric on S^3 [34]

$$\mathrm{d}(\mathbf{q}_1,\mathbf{q}_2) = \begin{cases} 2\pi, & \mathbf{q}_1 * \overline{\mathbf{q}}_2 = [-1,0,0,0]^{\mathrm{T}} \\ 2\|\log(\mathbf{q}_1 * \overline{\mathbf{q}}_2)\|, & \text{otherwise} \end{cases}. \tag{1.43}$$

Note that d defined above is not a metric on SO(3) because $\mathrm{d}(\mathbf{q},-\mathbf{q}) = 2\pi$ although $\mathbf{q}$ and $-\mathbf{q}$ represent the same orientation. It is nevertheless possible to compute the distance on SO(3) also using the quaternion logarithm. This can be done by checking the norms of the two logarithms, $\|2\log(\mathbf{q}_1 * \overline{\mathbf{q}}_2)\|$ and $\|2\log(-\mathbf{q}_1 * \overline{\mathbf{q}}_2)\|$. The smaller value should then be taken to obtain the distance on SO(3).

Finally, we derive the relationship between quaternion derivative and angular velocity. For a small time interval Δt, the rotation can be approximated as a small rotation by an angle $\Delta\theta$ around the axis given by the angular velocity vector ω. According to (1.40),

such a rotation is defined by a quaternion

$$\mathbf{q}(\Delta\theta, \boldsymbol{\omega}/\|\boldsymbol{\omega}\|) = \cos\left(\frac{\Delta\theta}{2}\right) + \sin\left(\frac{\Delta\theta}{2}\right)\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \approx 1 + \frac{\Delta\theta}{2}\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \approx 1 + \frac{\boldsymbol{\omega}\Delta t}{2}. \tag{1.44}$$

The right-hand site of this equation was derived from the relationship between the angular velocity $\boldsymbol{\omega}$ and the rate of change of the rotation angle

$$\frac{\Delta\theta}{\Delta t} \approx \|\boldsymbol{\omega}\|. \tag{1.45}$$

We obtain

$$\mathbf{q}(t + \Delta t) \approx \left(1 + \frac{\boldsymbol{\omega}\Delta t}{2}\right) * \mathbf{q}(t) = \mathbf{q}(t) + \frac{\boldsymbol{\omega}\Delta t}{2} * \mathbf{q}(t) \tag{1.46}$$

and

$$\dot{\mathbf{q}} = \lim_{\Delta t \to 0} \frac{\mathbf{q}(t + \Delta t) - \mathbf{q}(t)}{\Delta t} = \frac{1}{2}\boldsymbol{\omega} * \mathbf{q}. \tag{1.47}$$

Note that Eq. (1.8) provides a similar relationship between rotation matrix derivative and angular velocity

$$\dot{\mathbf{R}} = [\boldsymbol{\omega}]_\times \mathbf{R}. \tag{1.48}$$

1.3 Programming by demonstration

Robot Programming by Demonstration (PbD), also known as Learning from Demonstration (LfD), Imitation Learning, or Teaching by Showing, is an approach that allows robots to learn tasks by observing and imitating human demonstrators. This method offers a more intuitive way to program robots, especially for those who may not have extensive technical expertise in robotics. By bridging the gap between human intuition and robotic execution, PbD provides a platform for teaching robots complex tasks in unstructured environments.

Traditional robot programming can be a daunting task. It requires in-depth knowledge of robotics principles and programming skills. Programming robots often involves intricate understanding of kinematics, dynamics, control systems, and sensor integration. Developers need to write detailed code to specify every movement and decision a robot must make, which is a time-consuming and error-prone process. Furthermore, the diversity of environments and tasks necessitates domain-specific expertise to tailor robotic behaviors appropriately. As robots are employed in more complex and varied applications, the limitations of traditional programming approaches become increasingly apparent. This complexity creates a barrier for those who wish to leverage robotic technology without the technical background. Programming by Demonstration offers a more accessible solutions for such users.

Programming by Demonstration aims to simplify robot programming by allowing robots to learn tasks from human demonstrations. Instead of explicitly programming

every detail of a task, PbD involves showing the robot how the task is performed, enabling it to learn and generalize from these examples. This method reduces the need for extensive coding and provides a more natural interface for interacting with robots.

The benefits of PbD are manifold. It makes robot programming more accessible to non-experts, significantly reduces the time and effort required to teach robots new tasks, and increases adaptability to different environments and tasks. By focusing on human-like learning and imitation, PbD allows robots to handle tasks with more flexibility and efficiency, particularly in unstructured or changing environments.

1.3.1 Programming by Demonstration Levels

Programming by Demonstration can be understood across several levels, each addressing different aspects of robot learning and task execution. These levels include

- trajectory level,
- skill level,
- task level,
- symbolic level, and
- hyrbid PbD.

Trajectory-level programming by demonstration

Trajectory-level programming focuses on capturing and replicating the precise paths and movements demonstrated by a human instructor. By using modern motion capture systems, trajectory-level Programming by Demonstration (PbD) involves recording the demonstrator's movements and translating them into robot-executable trajectories. This approach ensures that robots follow the exact movements necessary for task completion, making it ideal for tasks that require high precision and accuracy, such as surgical procedures, welding, or assembly tasks in manufacturing.

The strength of trajectory-level programming lies in its ability to replicate complex, continuous motions that may be difficult to describe through traditional programming methods. By directly capturing human motion and mapping it to the robot motion, robots can achieve smooth and natural movements that closely mimic human performance.

However, trajectory-level programming can struggle with adapting to variations in task execution, as it emphasizes specific motion reproduction rather than task understanding. This limitation can make it challenging for robots to adapt to changes in the environment or to different task conditions, such as varying object sizes or positions. Additionally, trajectory-level programming may not handle dynamic or unpredictable environments well, as it lacks the flexibility to adapt to real-time changes.

Skill-level programming by demonstration

Skill-level programming provides an intermediate layer between trajectory-level and task-level programming. It focuses on teaching robots reusable skills or behaviors that can be combined to perform complex tasks. Skills such as grasping, manipulating objects, or navigating obstacles are abstracted from specific tasks and can be reused in different contexts. These skills serve as building blocks for more intricate operations, emphasizing modularity and flexibility. This approach enables robots to adapt and generalize across various tasks by learning skills that can be transferred between different applications.

A critical component of skill-level programming is the concept of movement primitives, which are fundamental building blocks of complex behaviors. Movement primitives represent basic actions or motions that can be combined and sequenced to form complete skills. By learning these primitives through Programming by Demonstration (PbD), robots can capture essential movements from human demonstrations and use them as a basis for developing more complex skills. For instance, a movement primitive for a robot arm might involve the basic motion required to reach out and grasp an object, which can then be adapted and modified for various tasks involving different objects or environments.

In skill-level programming by demonstration, robots observe human demonstrations to learn these skills. The process involves capturing key actions and movements, abstracting them into skills and movement primitives that can be applied in various situations. For example, a robot might learn the skill of grasping by observing a human picking up objects of different shapes and sizes, then generalize this skill to handle new objects it has never encountered before. This ability to generalize makes skill-level PbD particularly valuable in dynamic environments where tasks vary widely.

Combining skill programming by demonstration with other techniques, such as reinforcement learning, can further enhance the learning process. While PbD is effective for initial skill acquisition, reinforcement learning can refine and optimize these skills based on feedback from the environment. This synergy allows robots to continuously improve their performance, making skills more robust and adaptable to different scenarios. A key challenge in this approach is identifying and segmenting skills and movement primitives from continuous demonstrations, which requires sophisticated algorithms to analyze and classify actions accurately. Moreover, ensuring that skills are transferable between different tasks and environments is crucial, necessitating a deep understanding of the underlying principles of each task and the ability to abstract these principles into generalized skills.

Task-level programming by demonstration

Task-level programming by demonstration (PbD) shifts the focus from replicating precise movements to achieving high-level goals and action sequences necessary for completing a task. At this level, tasks are abstracted into a series of steps that need to be executed in a specific order. The assumption is that the robot already knows how to execute low-

level operations, such as moving an arm or gripping an object, and these operations are combined into a sequence to perform a more complex task. Task-level PbD provides a structured approach to replicating demonstrated behaviors and decision-making processes by observing human actions and translating them into symbolic or abstract representations.

This approach is particularly useful for tasks requiring coordination and executing multiple actions in a precise sequence, such as assembly operations, navigation, or collaborative tasks with humans or other robots. By focusing on high-level goals rather than specific movements, task-level PbD allows robots to interpret the intent behind actions and understand how to adapt them to new or changing situations. This flexibility enables robots to adjust their actions based on real-time feedback and environmental changes, improving their ability to function autonomously and effectively in dynamic environments.

While task-level PbD offers generalization across similar tasks, it often requires adaptation to the specific context or environment in which the task is performed. This adaptation can be achieved through sensory feedback, machine learning techniques, or user input, which help the robot refine its behavior in varying scenarios. By incorporating these elements, robots can learn from demonstrations to not only replicate tasks but also enhance their decision-making capabilities. Additionally, task-level PbD can be integrated with other levels, such as trajectory-level programming, to ensure that high-level plans are executed with the necessary precision and accuracy. This integration creates a comprehensive framework for robotic learning, combining the strengths of both high-level abstraction and detailed execution.

Symbolic-level programming by demonstration

Symbolic-level programming by demonstration (PbD) involves using symbols and logical constructs to teach robots complex tasks through human demonstrations. This approach focuses on abstract reasoning and decision-making, allowing robots to learn from demonstrations by representing tasks as high-level symbols that capture the essential features and relationships within the task environment. Through symbolic-level PbD, robots are capable of understanding abstract concepts such as objects, actions, and goals, enabling them to generalize learned knowledge to new situations and adapt to changes more effectively than with lower-level programming.

In symbolic-level PbD, robots observe human actions and extract symbolic representations that define the task's structure and intent. For example, in a household setting, a robot might observe a human placing objects in specific locations and represent this task with symbols like "pick," "place," "object," and "destination." This symbolic representation allows the robot to plan and execute similar tasks even when conditions change, such as when objects or destinations differ from the original demonstration. By abstracting tasks into symbolic forms, symbolic-level PbD facilitates a higher level of autonomy and decision-making, as robots can reason about the task's goals and constraints and adjust their actions accordingly.

One of the main challenges of symbolic-level programming by demonstration is converting sensory data into meaningful symbolic representations. This requires sophisticated algorithms capable of interpreting and categorizing complex sensory inputs. Additionally, integrating symbolic reasoning with real-time control systems can be difficult, as it necessitates seamless coordination between high-level planning and precise execution.

Hybrid robot programming by demonstration

Hybrid robot programming by demonstration (PbD) combines multiple learning levels to enable robots to perform complex tasks efficiently and flexibly. For instance, in a furniture assembly task, a robot utilizes trajectory-level programming to capture precise movements from human demonstrations, such as aligning parts and inserting screws. These precise actions are then abstracted into skill-level programming, where the robot develops reusable skills like "screwing" and "aligning" that can be adapted to various assembly scenarios. Task-level programming coordinates these skills into a coherent sequence to achieve the overall assembly goal, allowing the robot to execute multiple actions efficiently and adjust its plan as needed. At the same time, symbolic-level programming provides high-level reasoning capabilities, enabling the robot to interpret assembly instructions and make decisions about the sequence and tools required for different furniture designs. This integration of programming levels allows the robot to adapt to changing conditions and perform complex tasks with precision and autonomy, enhancing its ability to operate in dynamic environments.

1.3.2 Challenges of Programming by Demonstration

While robot programming by demonstration offers significant advantages, it also presents various challenges. A major challenge is the *correspondence problem*, which arises due to differences in human and robot body structures, making it difficult to map human motions directly to robot actions. Human muscles allow for smooth, continuous, and adaptable movements due to their complex structure and ability to modulate force naturally, while traditional robot actuators produce less compliant movements due to their mechanical nature.

Additionally, it is generally difficult to capture and replicate the forces and torques arising in human demonstrations, especially when robots lack tactile feedback, which is crucial for tasks involving physical interactions. Dynamics play a critical role here, as robots need to understand not just the positions and velocities of movements but also how forces and torques affect interactions with objects and the environment. This complexity requires sophisticated models to translate the demonstrated dynamic tasks into robotic actions accurately.

To address these challenges comprehensively, several strategies are employed. Programming by demonstration is easier if robots with anthropomorphic features are used to imitate demonstrated tasks. This can simplify the mapping of human motions to robot

actions by, to a degree, mimicking human anatomy and movement patterns. In some cases, focusing on the goal of an action rather than replicating the exact motion can circumvent the need for direct motion mapping, allowing robots to achieve the desired outcomes without accurately replicating the demonstrated motion. Integrating sensors that provide feedback on force, torque, and environmental conditions further enhances robots' ability to adapt their actions dynamically. The integration of sensory feedback supports adaptive control strategies, which are especially important for handling dynamic interactions during complex tasks.

Another significant challenge is acquiring a sufficient number of high-quality human demonstrations for the desired tasks. Human demonstrations are often limited due to practical constraints such as time, effort, and variability among different human instructors. This scarcity of data can hinder the effectiveness of PbD approaches and the robots' ability to generalize across different tasks and environments. Advanced machine learning algorithms can help address this issue by enabling robots to learn from fewer examples through techniques such as data augmentation, transfer learning, and reinforcement learning. Additionally, simulation environments offer a safe and effective way to test and refine the computed programs before deploying them in the real world. This approach minimizes the risk of damage to both the robot and its surroundings and enables knowledge refinement without the need for the physical execution of tasks.

1.4 Data collection for robot programming by demonstration

In robot programming by demonstration (PbD), collecting accurate data on human actions and movements is essential for enabling robots to learn and replicate tasks effectively. Various systems are employed to capture these demonstrations, each offering unique methods to capture and interpret human behaviors. These systems can be categorized based on the technology used, and each has its own advantages and challenges.

1.4.1 Marker-Based Optical Trackers

Marker-based optical tracking systems are widely used in robotics for capturing detailed motion data. These systems rely on cameras and markers to track the position and orientation of objects in three-dimensional space. They are divided into two types: *passive* and *active* optical trackers.

Passive optical trackers use markers that reflect infrared light. These markers are often spherical and coated with retro-reflective material. The system includes multiple infrared cameras that emit light, which is then reflected by the markers. The cameras capture these reflections and use the data to triangulate the precise location of each marker in space. The simplicity and cost-effectiveness of passive markers make them appealing for many

applications. However, they are highly susceptible to occlusion and require a controlled environment to maintain accuracy.

Active optical trackers, on the other hand, use markers that emit their own infrared signals. These markers are often equipped with LEDs that actively send signals to the cameras, which then process the data to determine the marker's position. The advantage of active markers lies in their ability to maintain tracking even when occlusions occur. The unique signals emitted by active markers allow for continuous identification, making them more reliable in dynamic environments. However, active markers require a power source, which adds complexity and potential signal interference from other infrared devices. The power source are usually small batteries, which slightly increases the weight of markers.

One of the key challenges in using optical trackers is the need for data post-processing. After data collection, it is often necessary to correct for occlusions, noise, and inconsistencies in marker detection. This involves using sophisticated algorithms to smooth data, fill in missing information, and ensure the captured motion data accurately reflects the intended movements.

Motion capture pipeline for PbD

In PbD, captured motion data is used to teach robots how to perform tasks by replicating human actions. This involves translating human movements into robot kinematics. A typical processing pipeline is as follows:

- **Marker Placement**:

  Markers are placed on or near joints such as shoulders, elbows, wrists, hips, knees, and ankles. This placement is crucial to accurately capture the motion of the limbs. For example, placing markers at the lateral and medial sides of the knee joint helps capture knee flexion and extension. Markers can also be attached to body segments like torso, head, and limbs (middle of the upper arm, forearm, thigh, and calf) to track the movement and rotation of these parts accurately. On larger body segments, markers may be arranged in a triangular or tetrahedral configuration to provide more precise data about the segment's orientation and deformation. This setup helps in accurately defining the 3D space occupied by the limb.

  Ensuring symmetrical placement on bilateral body parts is essential for balanced data capture. This means placing markers at equivalent positions on both the left and right limbs. To minimize errors due to skin movement (soft tissue artifacts), markers should be securely attached and, when possible, placed over areas where skin movement is minimal. Strapping markers over tight clothing can also help reduce artifacts.

  Passive markers are often attached using double-sided adhesive tape, ensuring they remain securely in place during motion. Active markers, due to their additional weight, may be attached using straps or clips, providing more secure attachment, especially during dynamic activities.

Well-placed markers simplify the data processing pipeline, as the software can more easily identify and track markers throughout the motion sequence. This reduces the need for manual corrections.

- **Camera Setup and Calibration**:

  Multiple high-resolution cameras are arranged around the capture area to provide complete coverage. The camera placement should ensure that each marker is always observed by more than one camera. The cameras are equipped with infrared filters to detect light reflected or emitted by the markers. They are calibrated to define a global coordinate system in which marker motion is computed. Calibration involves capturing known points or patterns and adjusting camera settings to minimize distortion and align perspectives.

  Calibration often includes a process called "wand calibration", where a wand with known markers is moved through the capture volume to help align and synchronize the cameras. Proper calibration ensures that all cameras work in harmony to produce accurate 3D reconstructions of the captured motion.

- **Image Data Acquisition**:

  The subject, fitted with markers as explained above, performs movements within the capture volume. Cameras capture the motion from different angles simultaneously. The system synchronizes images from all cameras to ensure a coherent measurement of marker positions. This synchronization is crucial for accurately reconstructing the 3D trajectories of the markers.

- **Marker Data Acquisition and Processing**:

  The software identifies and labels each marker in the captured frames, distinguishing between markers based on patterns of movement and spatial relationships. Active systems simplify this process as each marker can emit a unique signal. The system triangulates the 3D positions of the markers using data from multiple camera perspectives. This involves solving mathematical equations to determine precise spatial coordinates of each marker. Filtering techniques, such as Kalman filters or spline smoothing, can then be applied to reduce noise and remove artifacts from the data, ensuring smoother motion paths.

  Additional post-processing steps may include correcting for any remaining noise, dealing with marker swaps (where markers might get mislabeled), and refining the trajectory data to ensure it accurately reflects the intended motion.

- **Skeleton Mapping and Kinematic Analysis**:

  The 3D marker data is mapped onto a skeleton model representing the subject's body structure. This mapping involves computing the joint angles of the skeleton

model so that the model markers match the measured 3D marker positions. Typically, the marker positions are first transformed into the 3D poses of body segments, and the joint angles are computed from the relative orientation of adjacent body segments. For example, the elbow joint angle is derived from the orientation of the forearm relative to the upper arm.

Skeleton mapping and human motion analysis perform best when personalized skeleton models are used. These models are derived from a generic skeleton model of the human body, which contains predefined joint locations and lengths of limbs and other body parts. The generic skeleton model serves as a baseline that can be adjusted to fit an individual's specific anatomical features. The scaling of the generic model to create personalized models is accomplished by using marker data to measure limb lengths and joint positions. This process often involves an automatic calibration step, where the system aligns the template's proportions with the subject's unique body dimensions, ensuring precise motion representation. By adjusting the template, the motion capture system can accurately map the subject's movements and provide detailed kinematic analysis.

There are several challenges that must be tackled by the described pipeline. A significant issue are soft tissue artifacts. Markers placed on the skin can move independently of the underlying bone structure, leading to inaccuracies in joint angle calculations. Techniques such as marker clusters and algorithms that estimate bone positions based on skin movement help mitigate this problem. Occlusions, where markers are temporarily hidden by other body parts or objects, can cause data gaps. Increasing the number of cameras can help maintain continuous marker visibility. A similar challenge is marker swapping due to fast or complex motions that cause the system to confuse markers. This can be addressed with uniquely identifiable active markers or advanced algorithms that predict each marker's trajectory.

Environmental conditions, such as poor lighting, reflections, or interference from other infrared sources, can also disrupt data capture. Controlled environments or using adaptive systems that can handle variable conditions are essential. Additionally, infrared-based systems may suffer from interference caused by external infrared sources like sunlight, reducing accuracy. Using filters and adjusting camera sensitivity can help mitigate this interference. Human error during marker placement or system configuration can lead to inaccurate data. Standardized procedures and automated setup systems can reduce the impact of human error, enhancing the reliability of motion capture systems. Lastly, in applications requiring real-time feedback, latency can be a critical issue, affecting the responsiveness of the system. Optimizing algorithms for faster processing and reducing the computational demands can minimize latency, providing more immediate feedback and improving the user experience.

1.4.2 Inertial Measurement Units (IMUs)

Inertial Measurement Units (IMUs) are advanced sensor systems that integrate accelerometers, gyroscopes, and sometimes magnetometers to provide comprehensive data about an object's orientation, velocity, and gravitational forces. They are widely used for motion capture in various applications, including robot programming by demonstration (PbD).

One of the primary advantages of IMUs is their portability. IMUs are small, lightweight, and easy to attach to different parts of the human body. This portability allows for the recording of natural human movements in diverse environments, including outdoor settings, without the need for a fixed setup. Unlike optical systems, which may require specific lighting conditions or line-of-sight visibility, IMUs can function effectively in a wide range of settings, providing flexibility in data collection. IMUs excel in capturing rotational data, offering precise measurements of body segment orientations. By attaching IMUs to key points on the body, such as the limbs and torso, it is possible to accurately compute joint angles and understand how different body parts interact during task execution.

IMUs can also measure the relative position of body parts by integrating the acceleration data captured by the accelerometers. This process involves integrating acceleration over time to obtain velocity and then integrating velocity to derive displacement. However, this method provides a way to estimate how much a particular body segment has moved relative to its initial position, and this approach is inherently subject to errors. Any inaccuracies in the initial acceleration measurements are compounded through the integration process, leading to drift over time. Drift refers to the accumulation of errors in the calculated position and orientation, which can result from sensor noise, bias, temperature fluctuations, and prolonged use.

Despite their benefits, IMUs present several challenges that need to be addressed to maximize their effectiveness in PbD. One significant issue is drift, a phenomenon where small errors in measurement accumulate over time, leading to inaccuracies in the reported position and orientation. Implementing algorithms to correct drift and recalibrate sensors regularly is essential to ensure reliable data capture. Kalman filters or other sensor fusion techniques are often used to mitigate drift by combining IMU data with information from other sensors,

1.4.3 Electromagnetic Trackers

Electromagnetic tracking systems use magnetic fields to determine the position and orientation of sensors relative to a fixed transmitter. Like IMUs and unlike optical systems, electromagnetic trackers do not require a direct line of sight between the transmitter and sensors, making them highly effective in cluttered or confined environments where visual interference is common. This capability allows for seamless motion capture and teaching of robots in complex setting. The real-time tracking capability of electromagnetic systems provides immediate feedback, which is vital for applications that demand fast and

accurate motion capture, such as PbD.

However, electromagnetic tracking systems face some significant challenges, primarily due to their sensitivity to metal and electronic interference. Such interference can distort the magnetic field and lead to inaccuracies in position and orientation data, necessitating careful setup to avoid potential disruption sources. Additionally, the effective range of these systems is typically limited to a few meters from the transmitter, which can restrict their use in large spaces. For robots to learn effectively in PbD, it is crucial to strategically position transmitters to cover the intended area adequately. While calibration is usually straightforward, it may need to be repeated if environmental conditions change, ensuring the accuracy and integrity of the captured data.

In the context of PbD, electromagnetic tracking systems are beneficial for applications that require continuous and unobstructed data capture. They are particularly useful in scenarios where optical systems might fail, such as in virtual reality environments.

1.4.4 Computer Vision Techniques

Computer vision systems leverage cameras and AI algorithms to analyze visual data and track movements without the need for markers. This markerless motion capture approach simplifies setup, reduces time and cost, and provides a more natural and unobtrusive method for capturing human motion. Such systems are versatile and applicable across various environments, making them a valuable tool in robotic programming by demonstration (PbD).

Deep learning has significantly enhanced computer vision systems, particularly for pose and posture estimation in robot PbD. Algorithms such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) extract features and patterns from complex visual data. CNNs, which excel in image recognition and classification tasks, are particularly effective for identifying and tracking body parts and objects during demonstrations. By employing multiple layers of convolutional filters, CNNs detect intricate patterns in visual data, facilitating accurate motion tracking without requiring manual feature extraction. RNNs complement this capability by processing sequential data, enabling the detection and reconstruction of complex action sequences demonstrated by humans.

Several systems have successfully integrated deep learning techniques for pose detection, which is critical for robot PbD. OpenPose and Google's PoseNet estimate human poses in real-time by identifying key points on the human body, such as joints and limbs. These systems use CNNs to analyze video input, translating pixel information into detailed pose estimations. OpenPose is widely recognized for its ability to detect multiple body parts simultaneously, including hands and facial landmarks, while PoseNet focuses on identifying key body joints and provides efficient solutions for single-person pose estimation. Both systems exemplify the power of CNNs in analyzing complex motion patterns.

Additionally, DeepLabCut is an open-source toolkit that uses CNNs for precise motion

capture. By training the network to recognize specific body parts in a small subset of labeled images, DeepLabCut generalizes this knowledge to new frames, which enables accurate tracking of posture and motion. MediaPipe, developed by Google, employs machine learning models to detect landmarks on the body, face, and hands. Through the combined use of CNNs and graphical models, MediaPipe ensures spatial and temporal consistency. This makes it a robust solution for understanding human motion in real-time.

Despite their advantages, computer vision systems face several challenges in robot PbD. Accurate processing requires complex algorithms, significant computational power, and high-performance hardware, which can limit real-world applications. Environmental factors such as lighting conditions, shadows, and occlusions can also affect system performance, leading to inconsistencies in image quality and tracking accuracy. Addressing these challenges requires the development of robust algorithms capable of adaptation to varying conditions and compensation of these disturbances. As technology advances, these challenges are gradually being mitigated, which makes the applicability of computer vision systems in robotic programming by demonstration increasingly viable.

1.4.5 Teleoperation

In a typical teleoperation setup for PbD, a human operator uses control interfaces such as joysticks or haptic devices to guide the robot's actions. As the operator controls the robot, the robot's movements are recorded, capturing the precise motion trajectories. If the robot is equipped with force-torque sensors, the applied forces due to environmental interactions can also be recorded. The operator can observe how the robot moves either directly or through live video feeds. Haptic devices offer tactile feedback, allowing the operator to "feel" the environment and perform delicate manipulations with precision.

The data collected through teleoperation focuses on the robot's actual movements rather than the operator's commands. By recording how the robot moves in response to human guidance, the system captures detailed information about the robot's kinematics and dynamics.

Despite its advantages, teleoperation for PbD faces challenges. Network latency can affect the responsiveness of the control system, leading to inaccuracies in executing commands, especially in fast-paced or safety-critical tasks. To mitigate these delays, robust communication systems and predictive algorithms are necessary to ensure smooth control. Additionally, translating human inputs into robot actions can be challenging, especially for intricate tasks.

1.4.6 Virtual Reality Systems

Virtual Reality (VR) systems provide a simulated environment where users can perform tasks and demonstrate actions to gather data for PbD. VR systems like Oculus Rift and Microsoft HoloLens offer immersive experiences that facilitate detailed task demonstrations by allowing users to interact with virtual objects and environments as if they were

real. The primary advantage of VR systems in PbD is the creation of a safe demonstration environment. This enables users to explore complex tasks without the fear of causing damage or harm. VR systems provide a rich, controlled environment for capturing demonstrations, allowing for precise manipulation and interaction that might be difficult to achieve in the real world.

However, one of the significant challenges when transitioning from VR to real-world applications is the reality gap. This gap arises due to differences between the simulated environment and the physical world, particularly concerning physical dynamics and interactions. For example, a robot learning a task in VR might not encounter the same friction, gravity, or collision responses when performing the task in the real world. These discrepancies can lead to errors or inefficiencies in the robot's performance when moving from the virtual to the physical environment. Addressing the reality gap requires sophisticated modeling techniques to ensure that the VR environment accurately reflects the real-world conditions.

Modeling environments with sufficient accuracy for PbD is a complex and challenging task. It involves creating detailed simulations of physical properties, such as material textures, forces, and kinematic constraints. This level of detail requires advanced software and computational power to achieve realistic simulations, making it a significant barrier to the widespread adoption of VR in PbD setups. For instance, simulating the weight and resistance of objects, the tactile feedback when interacting with surfaces, and the dynamic response of objects to forces all require computationally expensive algorithms.

Moreover, VR systems often require sophisticated hardware, such as high-resolution headsets and motion tracking equipment, to ensure realistic simulations. This adds to the complexity and cost of using VR for PbD.. Despite these challenges, ongoing advancements in VR technology continue to improve the fidelity and realism of simulations, gradually narrowing the reality gap and enhancing the effectiveness of VR in teaching robots through demonstration.

1.4.7 Wearable Sensor Systems

Wearable sensors, such as data gloves and sensesuits, are worn directly on the body to capture detailed motion data. In the case of humanoid robots, sensesuits are designed to replicate the kinematic structure of the target humanoid robot. Thesy consist of passive mechanical structures embedded with sensors, such as goniometers, which measure joint angles and their motion. This design ensures that the recorded human motion can be directly mapped onto the robot without complex transformation algorithms. This eliminats the correspondence problem often encountered in PbD. Wearable sensors offer high-resolution data, capturing even the most subtle movements and enabling the recording of complex trajectories essential for sophisticated robotic tasks.

Despite their advantages, wearable systems can be cumbersome and may interfere with the user's natural movement. The bulkiness can restrict fluidity, leading to less

naturalistic data capture. Moreover, data gloves require frequent calibration to maintain accuracy and can experience drift over time, compromising the precision of the data collected. While necessary, the calibration processes can be time-consuming and may not always eliminate all sources of error.

Sensesuits, much like clothing, are typically designed in specific sizes, which can be a limitation when working with human demonstrators of varying heights and body types. This means that a different sensesuit may be required for taller or shorter individual. Additionally, because sensesuits are constructed to mirror a specific kinematic structure, they are only suitable for robots with such a kinematic structure. This bespoke design approach makes sensesuits more expensive than other, more adaptable measurement devices.

1.4.8 Kinesthetic Guidance

Kinesthetic guidance involves physically guiding a robot through the desired motions. In this approach, the human instructor directly interacts with the robot, manually moving its end-effector to demonstrate the specific movements needed for a task. This hands-on method allows for precise control over the robot's movements, making it a highly effective way to teach robots complex motions.

The primary advantage of kinesthetic guidance is its directness and intuitiveness. Since the human instructor physically guides the robot, the motions are recorded directly in the robot's joint space, eliminating the correspondence problem often encountered in other Programming by Demonstration (PbD) methods. This direct mapping ensures that the robot accurately learns the desired movements without needing complex transformation algorithms to translate human motion into robotic actions. The instructor can specify key points for point-to-point movements or capture entire joint space trajectories, providing flexibility in how the motion data is recorded. This approach is particularly useful for tasks where the focus is on the robot's end-effector motion, such as manipulating objects, assembling components, or performing precise tasks like welding.

Some robots, like the Franka Emika, are equipped with button interfaces that enhance the kinesthetic guidance process. These interfaces allow the instructor to interact with the robot's control system directly during the guidance process. By pressing buttons located on the robot, the instructor can easily switch between different modes, such as teaching mode and playback mode, or lock specific joints to restrict movement to certain axes. This capability makes it easier to demonstrate complex motions and adjust the robot's posture without interrupting the teaching process. The button interfaces provide real-time feedback and control, allowing the operator to focus on the subtleties of the task at hand while ensuring that the robot accurately records the intended path.

Despite its advantages, kinesthetic guidance has some limitations and requires skilled operators to guide the robot accurately. The instructor must have a good understanding of the task and the robot's capabilities to ensure that the guided motion is both precise and

safe. Additionally, while kinesthetic guidance excels in teaching end-effector movements, it can be challenging to ensure specific robot configurations, especially for redundant robots with more than six degrees of freedom. Redundant robots have multiple ways to achieve the same end-effector position, making it difficult to control the robot's entire posture during guidance. This can lead to suboptimal configurations that might not be ideal for certain tasks or environments.

Furthermore, kinesthetic guidance is less practical for dual-arm robots and other robots requiring simultaneous control of multiple limbs. Since the instructor typically uses both hands to guide a single robot arm, managing two arms at once can be cumbersome and impractical. This limitation necessitates alternative approaches to handle multi-limb tasks effectively. The development of advanced interfaces and control strategies continues to be an area of active research, aiming to improve the usability and effectiveness of kinesthetic guidance for complex robotic systems.

1.4.9 Summary

Each of these systems offers unique benefits and limitations, making them suitable for different types of applications in robot programming by demonstration. The choice of data collection system depends on factors such as the task's complexity, the environment, and the desired level of precision and adaptability. Combining these technologies can often yield the best results, leveraging the strengths of each system to overcome the limitations of others. By employing a diverse range of data collection techniques, robots can achieve a more comprehensive understanding of human behavior, leading to more effective and versatile robotic systems.

Chapter 2

Polynomials and splines

A standard approach to planning robotic motions is to specify a time-dependent parametric trajectory that determines the robot's motion from the start at $t = 0$ to its end at $t = T$. Trajectories are typically defined in the robot's configuration space, specifying the joint coordinates $\mathbf{y}(t)$, $\forall t \in [0, T]$, $\mathbf{y}(t) \in \mathbb{R}^n$, where n is the number of the robot's degrees of freedom, or in the task space, such as 3-D Cartesian space. In task space, the robot's position $\mathbf{p}(t) \in \mathbb{R}^3$ and orientation $\mathbf{q}(t) \in \mathrm{SO}(3) \subset \mathbb{R}^4$, expressed as unit quaternions, are typically specified. Other task spaces, such as a 2-D plane, may also be relevant depending on task constraints.

Smoothness and continuity are essential for generating feasible trajectories. To ensure stable and dynamically consistent motion, trajectories should be continuous up to at least the second derivative, meaning the trajectory, velocity, and acceleration profiles are smooth throughout. For tasks requiring precise control, such as point-to-point movements or following a complex path, this continuity minimizes abrupt transitions, reduces wear on actuators, and ensures energy-efficient motion.

For simple point-to-point motions, minimum jerk trajectories provide an effective solution. Minimum jerk trajectories generate smooth straight-line spatial paths in Cartesian space with speed profiles determined by fifth-order polynomials. Orientation changes are handled using SLERP (Spherical Linear Interpolation), which ensures the shortest path in quaternion space. However, more complex tasks, such as following curved paths or adapting to obstacles, require more flexible representations.

Splines, particularly B-splines, address these challenges by enabling smooth trajectories over multiple segments with high-order continuity. Splines allow local control, meaning changes in one segment affect only a small neighborhood. This makes them suitable for handling dynamic constraints and adapting to new tasks. This flexibility is especially valuable for learning robots, which must balance noisy sensor data, task constraints, and trajectory smoothness.

The differences between polynomials and splines stem from their structure and flexibility. While polynomials are simple and computationally efficient, they are limited to describing single-segment trajectories and may suffer from oscillations when used for

longer paths. Splines, on the other hand, are composed of piecewise polynomial segments, making them more robust and versatile for complex trajectory generation. Splines such as B-splines and natural splines offer additional benefits like smooth transitions, the ability to interpolate or approximate data, and ease of handling constraints.

In this chapter, we explore the mathematical tools of polynomials and splines for trajectory generation. We begin with point-to-point motion using minimum jerk trajectories and SLERP for orientation, followed by a detailed discussion of spline representations. This includes polynomial splines and B-splines, and their role in generating complex, smooth robot motions.

2.1 Point-to-point motion

Minimum jerk point to point movements between two positions in Cartesian space with zero initial and final velocities and accelerations result in straight lines spatial paths with speed profiles defined by a fifth order polynomial. Note that a straight line is the shortest path between two points in space. Equivalently, a SLERP trajectory defines the shortest path between two orientations in the orientation space. The term SLERP has been coined in computer graphics and stands for spherical linear interpolation. To define such trajectories, the following data needs to be available:

- the initial pose $\left[\mathbf{t}_i^{\mathrm{T}}, \mathbf{q}_i^{\mathrm{T}}\right]^{\mathrm{T}}$ and the final pose $\left[\mathbf{t}_f^{\mathrm{T}}, \mathbf{q}_f^{\mathrm{T}}\right]^{\mathrm{T}}$, where $\mathbf{t}_i, \mathbf{t}_f \in \mathbb{R}^3$ respectively denote the initial and final position and $\mathbf{q}_i,\ \mathbf{q}_f \in \mathrm{S}^3 \subset \mathbb{R}^4$ the initial and final orientation expressed as unit quaternions, and

- the desired travel time T.

We write

$$\mathbf{p}_{\mathrm{minJrk}}(t; \left[\mathbf{t}_i^{\mathrm{T}}, \mathbf{q}_i^{\mathrm{T}}\right]^{\mathrm{T}}, \left[\mathbf{t}_f^{\mathrm{T}}, \mathbf{q}_f^{\mathrm{T}}\right]^{\mathrm{T}}, T) = \left[\mathbf{t}(t; \mathbf{t}_i, \mathbf{t}_f, T)^{\mathrm{T}}, \mathbf{q}(t; \mathbf{q}_i, \mathbf{q}_f, T)^{\mathrm{T}}\right]^{\mathrm{T}}. \tag{2.1}$$

Next, we first provide the formulas for the position trajectory $\mathbf{t}(t)$, followed by the formulas for SLERP trajectory $\mathbf{q}(t)$ expressed in the space of unit quaternions.

2.1.1 Minimum jerk trajectories

Given the initial and final positions $\mathbf{t}_i,\ \mathbf{t}_f \in \mathbb{R}^3$, the minimum jerk position trajectory $\mathbf{t}(t) \in \mathbb{R}^3$ is given by a fifth order polynomial

$$\mathbf{t}(t; \mathbf{t}_i, \mathbf{t}_f, T) = \mathbf{a}_6 t^5 + \mathbf{a}_5 t^4 + \mathbf{a}_4 t^3 + \mathbf{a}_3 t^2 + \mathbf{a}_2 t + \mathbf{a}_1, \tag{2.2}$$

with the coefficients computed as follows:

$$\begin{aligned}
\mathbf{a}_1 &= \mathbf{t}_i, && (2.3)\\
\mathbf{a}_2 &= 0, && (2.4)\\
\mathbf{a}_3 &= 0, && (2.5)\\
\mathbf{a}_4 &= \frac{10(\mathbf{t}_f - \mathbf{t}_i)}{T^3}, && (2.6)\\
\mathbf{a}_5 &= -\frac{15(\mathbf{t}_f - \mathbf{t}_i)}{T^4}, && (2.7)\\
\mathbf{a}_6 &= \frac{6(\mathbf{t}_f - \mathbf{t}_i)}{T^5}. && (2.8)
\end{aligned}$$

Here $\mathbf{a}_i \in \mathbb{R}^3,\ \forall i$. Note that

$$\mathbf{t}(t; \mathbf{t}_i, \mathbf{t}_f, T) = \mathbf{t}_i + (\mathbf{t}_f - \mathbf{t}_i)\left(6\,(t/T)^5 - 15\,(t/T)^4 + 10\,(t/T)^3\right). \quad (2.9)$$

2.1.2 SLERP trajectories

Given the initial and final orientations $\mathbf{q}_i,\ \mathbf{q}_f \in \mathrm{S}^3$, a SLERP trajectory is defined as follows:

$$\mathbf{q}(t; \mathbf{q}_i, \mathbf{q}_f, T) = \vartheta_1(t)\mathbf{q}_i + \vartheta_2(t)\mathbf{q}_f,\ 0 \le t \le T, \quad (2.10)$$

where

$$\begin{aligned}
\vartheta_1(t) &= \frac{\sin\left((1 - s(t))\,\theta\right)}{\sin(\theta)}, && (2.11)\\
\vartheta_2(t) &= \frac{\sin\left(s(t)\theta\right)}{\sin(\theta)}, && (2.12)
\end{aligned}$$

and θ is equal to the half of the rotation angle needed to rotate $\mathbf{q}_i$ to $\mathbf{q}_f$. It can be calculated as

$$\theta = \arccos(v),\ (v, \mathbf{u}) = \mathbf{q}_f * \overline{\mathbf{q}_i},\ v \in \mathbb{R},\ \mathbf{u} \in \mathbb{R}^3, \quad (2.13)$$

where $*$ denotes the quaternion product and $\bar{}$ the quaternion conjugation as defined in Eqs. (1.20) and (1.22), respectively. To compute the shortest path in SO(3), we need to select the quaternion ($\mathbf{q}_f$ or $-\mathbf{q}_f$) closer to $\mathbf{q}_i$. If $v \ge 0$, then $\mathbf{q}_f$ is closer to $\mathbf{q}_i$, otherwise we replace $\mathbf{q}_f$ with $-\mathbf{q}_f$. This way, v is always non-negative.

The temporal course of motion is determined by a monotonously increasing function $s(t) \in \mathbb{R},\ 0 \le s(t) \le 1,\ s(0) = 0,\ s(T) = 1$. A good choice is the fifth order polynomial that provides the speed profile for minimum minimum jerk trajectories

$$s(t) = 6\,(t/T)^5 - 15\,(t/T)^4 + 10\,(t/T)^3. \quad (2.14)$$

It can be shown that $\mathbf{q}(t) \in \mathrm{S}^3,\ \forall t$.

Since $\dot{s}(0) = \ddot{s}(0) = \dot{s}(T) = \ddot{s}(T) = 0$, the robot is at rest with zero angular velocity

and acceleration at the beginning and the end of motion. The angular velocity $\boldsymbol{\omega}$ at time t can be computed using the formula

$$\boldsymbol{\omega}(t) = 2\,\dot{\mathbf{q}}(t) * \overline{\mathbf{q}(t)}, \tag{2.15}$$

where $\mathbf{q}(t)$ is defined as in (2.10),

$$\dot{\mathbf{q}}(t) = \dot{\vartheta}_1(t)\mathbf{q}_i + \dot{\vartheta}_2(t)\mathbf{q}_f, \tag{2.16}$$

$$\dot{\vartheta}_1(t) = -\frac{\cos\left((1-s(t))\,\theta\right)}{\sin(\theta)}\dot{s}(t)\theta, \tag{2.17}$$

$$\dot{\vartheta}_2(t) = \frac{\cos\left(s(t)\theta\right)}{\sin(\theta)}\dot{s}(t)\theta, \tag{2.18}$$

and $s(t)$ is defined as in (2.14).

2.2 Polynomial splines

Due to their flexibility and ease of computation, *spline functions* have an important role in approximation theory and statistics. From the statistical point of view, they are described in great detail in [6, 37], whereas the classic book of de Boor [5] provides their detailed description from the numerical analysis and approximation theory point of view. They have also been applied for the generation of complex robot trajectories [33]. A polynomial spline function of order $2m$ on the strictly increasing knot sequence $t_1 < \ldots < t_N$ is a polynomial of order $2m-1$ on every interval $t_i \le t < t_{i+1}$, $i < N$, and $t_{N-1} \le t \le t_N$

$$s_{i,k}(t) = \sum_{j=0}^{k-1} \gamma_{ij} t^j,\ i = 1, \ldots, N-1. \tag{2.19}$$

For $k = 1$, we obtain linear splines, for $k = 2$ cubic splines and for $k = 6$ quintic splines. These are the most useful polynomial splines for robotic applications.

As there are $N-1$ intervals, a polynomial spline of form (2.19) on knot sequence $\{\tau_i\}_{i=1}^N$ has $k(N-1)$ parameters. However, a spline function useful for robotic applications need to be continuous and have continuous derivatives up to at least second order. To achieve continuity, we impose constraints

$$s_{i,k}^{(j)}(t_{i+1}) = s_{i+1,k}^{(j)}(t_{i+1}),\ i = 1, \ldots, N-2,\ j = 0, \ldots, d, \tag{2.20}$$

where d is the highest derivative order that is required to be continuous. Besides $k(N-1)$ parameters, we now also have $(d+1)(N-2)$ constraints. In this case there are altogether $(k-d-1)N+2(d+1)-k)$ free parameters. For cubic splines with continuous derivatives up to the the second order, we have $k = 4$, $d = 2$. Thus in this case there are $N+2$ free parameters. Similarly, in case of quintic splines with continuous derivatives up to the

fourth order, we obtain $N + 4$ free parameters.

A linear combination of polynomials of a certain order is a polynomial of the same order. Thus the space of all spline functions as defined in (2.19) is a vector space. A linear combination of spline functions that fulfill constraints (2.20) also fulfills these same constraints. This means that the space of all spline functions of order k that fulfill constraints (2.20) is also a vector space. Its dimension is $(k - d - 1)N + 2(d + 1 - k)$.

In case of polynomial splines with continuous derivatives, the representation from equation (2.19) is not ideal because it has more parameters than the dimension of the vector space it represents. In the following we show how to construct a minimal set of basis spline functions that can be used to represent the vector space of polynomial splines with continuous derivatives.

2.2.1 B-splines

Let's assume that we a have a nondecreasing knot sequence $\boldsymbol{\tau} = \{\tau_1, \ldots, \tau_{\tilde{N}}\}$, $\tau_i \leq \tau_{i+1}$. A B-spline of order 1 is defined as follows

$$B_{i,1}(t) = \begin{cases} 1, \ \tau_i \leq t < \tau_{i+1} \\ 1, \ \tau_i < \tau_{i+1} = \tau_{\tilde{N}}, \ t = \tau_{\tilde{N}} \\ 0, \ \text{otherwise} \end{cases}, \ i = 1, \ldots, \tilde{N}. \tag{2.21}$$

Unlike at the beginning of Section (2.2), here we allow that a knot appears more than once in the knot sequence $\boldsymbol{\tau}$, i.e. $\tau_i = \tau_{i+1}$. Note that according to this definition, $B_i = 0$ if $\tau_i = \tau_{i+1}$. Moreover, $B_i(t) \neq 0$ only on the interval $[\tau_i, \tau_{i+1})$ (or $[\tau_i, \tau_{i+1}]$ for one B-spline at the right end of the knot sequence $\boldsymbol{\tau}$). Note also that for any $t \in [\tau_1, \tau_{\tilde{N}}]$, we obtain

$$\sum_{i=1}^{\tilde{N}-1} B_{i,1}(t) = 1.$$

From the first order splines defined in equation (2.21), we recursively obtain the B-splines of higher orders ($k > 1$)

$$B_{i,k}(t) = \gamma_{i,k}(t) B_{i,k-1}(t) + (1 - \gamma_{i+1,k}(t)) B_{i+1,k-1}(t), \tag{2.22}$$

$$\gamma_{i,k}(t) = \begin{cases} \dfrac{t - \tau_i}{\tau_{i+k-1} - \tau_i}, \ \tau_i \neq \tau_{i+k-1} \\ 0, \ \text{otherwise} \end{cases}.$$

Note that for $\tau_i \neq \tau_{i+k-1}$, $\tau_{i+1} \neq \tau_{i+k}$, equation (2.22) can be rewritten as

$$B_{i,k}(t) = \frac{t - \tau_i}{\tau_{i+k-1} - \tau_i} B_{i,k-1}(t) + \frac{\tau_{i+k} - t}{\tau_{i+k} - \tau_{i+1}} B_{i+1,k-1}(t), \tag{2.23}$$

For the B-splines of order k, $\tilde{N} - k$ basis spline functions of order k can be obtained by the recursive formulas (2.21), (2.22).

Since $B_{i,1}$ are defined as constant step functions and higher order B-splines are obtained from lower order B-splines via multiplications that are linear in t, every B-spline of order k is a piecewise polynomial of order $k-1$ on knot sequence $\boldsymbol{\tau}$.

Next we prove by induction that $B_{i,k}(t) = 0$ outside of the support interval $[\tau_i, \tau_{i+k}]$ and that $B_{i,k}(t) > 0,\ \forall t \in (\tau_i, \tau_{i+k})$. By definition (2.21) of B-splines of order 1, both is true for $k = 1$. Higher order B-splines are defined by the recursive formula (2.22), where $B_{i,k}$ is computed as a linear combination of $B_{i,k-1}$ and $B_{i+1,k-1}$. By induction assumption, $B_{i,k-1}$ and $B_{i+1,k-1}$ vanish outside of the intervals $[\tau_i, \tau_{i+k-1}]$ and $[\tau_{i+1}, \tau_{i+k}]$, respectively. Thus their linear combination $B_{i,k}$ is zero outside of the interval $[\tau_i, \tau_{i+k}]$. Similarly, by induction assumption $B_{i,k-1}$ and $B_{i+1,k-1}$ are greater than zero on (τ_i, τ_{i+k-1}) and (τ_{i+1}, τ_{i+k}), respectively. Given that both $\gamma_{i,k}(t)$ and $1 - \gamma_{i+1,k}(t)$ are greater than zero on these same intervals and $B_{i,k-1}(\tau_{i+k-1}) = 0,\ B_{i+1,k-1}(\tau_{i+k-1}) > 0$, we have proven that $B_{i,k}(t) > 0,\ \forall t \in (\tau_i, \tau_{i+k})$.

An important property of B-splines is that although the lowest order B-splines $B_{i,1}$ are not continuous, the two weights $\gamma_{i,1}(t)$ and $1 - \gamma_{i+1,1}(t)$ are exactly right to cancel at each intermediate point τ_{i+1} the jump discontinuity of $B_{i,1}(t)$ and $B_{i+1,1}(t)$ so that $B_{i,2}(t)$ are continuous also at the internal knot points τ_{i+1}. The order of continuity increases also in derivatives as higher order B-splines are computed. In there are no multiple internal knot points, the B-splines of order k are continuously differentiable up to the derivative of order $k-2$ at the internal knot points. The multiplicity of internal knot points reduces the order of continuity.

This proof of this property relies on Marsden's identity, a fundamental result in spline theory. Intuitively, Marsden's identity provides a recursive relationship that combines adjacent B-splines to construct higher-order B-splines. The identity ensures that the resulting B-splines maintain the necessary continuity and smoothness properties. Specifically, Marsden's identity guarantees that the weighting functions used to combine adjacent B-splines inherently smooth out discontinuities in both the function and its derivatives, up to the required order. This mathematical mechanism underpins the construction of B-splines and explains their continuity properties. For a comprehensive proof and deeper understanding of Marsden's identity, we refer the reader to de Boor's seminal book on splines [5].

In summary, B-splines defined by equations (2.21) and (2.22) have the following properties

1. Each B-spline $B_{i,k}$ depends only on the knots $\{\tau_i, \ldots, \tau_{i+k}\}$.

2. Each B-spline $B_{i,k}$ is a piecewise polynomial on interval $[\tau_i, \tau_{i+k})$, thus it can be written as

$$B_{i,k}(t) = \sum_{j=0}^{k-1} s_{i+j,k}(t) B_{i+j,0}(t), \tag{2.24}$$

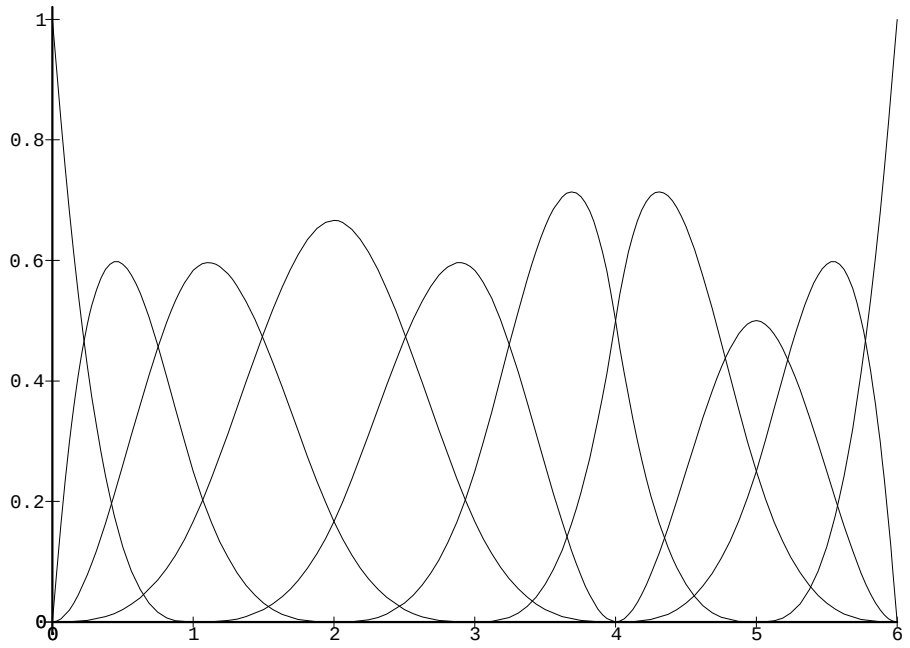

Figure 2.1: Cubic B-Splines on the knot sequence $\{0, 0, 0, 0, 1, 2, 3, 4, 4, 5, 6, 6, 6, 6\}$

where $s_{i+j,k}$ are defined as in equation (2.19).

3. $B_{i,k}(t) = 0$ outside of the interval $[\tau_i, \tau_{i+k}]$ and $B_{i,k}$ is equal to zero everywhere if $\tau_i = \tau_{i+k}$.

4. $B_{i,k}(t) > 0, \forall t \in (\tau_i, \tau_{i+k})$.

5. Each $B_{i,k}$ is $k - 1 - m_j$ continuously differentiable at every knot point τ_j, where m_j is the multiplicity of the knot τ_j.

Example cubic B-splines with multiple knots at the edge of the support interval are shown in Fig. 2.1.

2.2.2 Natural and complete smoothing splines

Let's now assume a monotonically strictly increasing sequence $\{t_i\}_{i=1}^{N}$, $t_i < t_{i+1}$, and define a knot sequence $\boldsymbol{\tau} = \{\tau_i\}_{i=1}^{N+4m-2}$, where $\tau_1 = \ldots = \tau_{2m} = t_1$, $\tau_{N+2m-1} = \ldots = \tau_{N+4m-2} = t_N$, and $\tau_{2m+1} = t_2, \ldots, \tau_{N-2+2m} = t_{N-1}$. As the multiplicity of all internal knot points in $\boldsymbol{\tau}$ is equal to 1, the B-splines of order $2m$ defined on knot sequence $\boldsymbol{\tau}$ are $2m - 2$ continuously differentiable on $[t_1, t_N]$. The multiplicity of edge knot points t_1 and t_N does not cause discontinuities on interval $[t_1, t_N]$ because the B-splines at the edges are respectively right and left continuous.

By induction we can show that B-splines $\{B_{i,2m}\}_{i=1}^{N+2m-2}$ associated with knot sequence

$\boldsymbol{\tau}$ form a local partition of unity, i.e.

$$\sum_{i=1}^{N+2m-2} B_{i,k}(t) = 1, \forall t \in [t_1, t_N]. \tag{2.25}$$

This property holds on the complete support interval $[t_1, t_N]$ because of $2m$ multiple knots at the edges. Without multiple knots, additional knots outside of the support interval $[t_1, t_N]$ would be needed to obtain the local partition of unity property in the neighborhood of t_1 and t_N.

Next we observe that B-splines $\{B_{i,2m}\}_{i=1}^{N+2m-2}$ are linearly independent. To prove this we must show that if $\sum_{i=1}^{N+2m-2} \alpha_i B_{i,2m}(t) = 0,\ \forall t \in (t_j, t_{j+1})$, then $\alpha_i = 0,\ \forall i$. For any interval (t_j, t_{j+1}), the local support property implies that there are only $2m$ B-splines different from zero on this interval, i.e. $B_{j,2m}, \ldots, B_{j+2m-1,2m}$. Thus for any $t \in (t_j, t_{j+1})$, we obtain $\sum_{i=0}^{2m-1} \alpha_{j+i} B_{j+i,2m}(t) = 0$. But on the interval (t_j, t_{j+1}), B-splines are just polynomials of degree $2m-1$. Using the Mardsen's identity, one can show that each basis polynomial t^i, $i = 0, , \ldots, 2m-1$, can be written as a linear combination of $B_{j+i,2m}(t)$. Thus $\{B_{j+i,2m}\}_{i=0}^{2m-1}$ are linearly independent on (t_j, t_{j+1}). It follows that if $\sum_{i=0}^{2m-1} \alpha_{j+i} B_{j+i,2m}(t) = 0$, then $\alpha_{j+i} = 0, \forall i$. Since this is true for any interval (t_j, t_{j+1}), it follows that B-splines are linearly independent.

In Section 2.2 we have seen that the space of polynomial splines with d continuous derivatives is at least $(k-d-1)N+2(d+1)-k$ dimensional. Inserting $k = 2m$, $d = 2m-2$ into this formula, we obtain $(2m-d-1)N + 2(d+1-m) = N + 2m - 2$. This means that B-splines $\{B_{i,2m}\}_{i=1}^{N+2m-2}$ form the basis for the space of polynomial splines with d continuous derivatives.

In summary for B-splines $\{B_{i,2m}\}_{i=1}^{N+2m-2}$ defined on knot sequence $\boldsymbol{\tau}$, we can add the following properties to the list from Section 2.2.1:

6. $\{B_{i,2m}\}_{i=1}^{N+2m-2}$ form a local partition of unity.
7. $\{B_{i,2m}\}_{i=1}^{N+2m-2}$ form the basis of polynomial splines with $2m-2$ continuous derivatives.

Thus each polynomial spline can be written as

$$s(t) = \sum_{i=1}^{N+2m-2} \alpha_i B_{i,2m}(t). \tag{2.26}$$

Moreover, since the support interval of every $B_{i,2m}$ is equal to $[t_i, t_{i+2m}]$, we obtain for $t_j \le t \le t_{j+1}$

$$s(t) = \sum_{i=j-2m+1}^{j} \alpha_i B_{i,2m}(t) \tag{2.27}$$

Thus for any $t \in [t_1, t_N]$, we need to consider the values of only $2m$ B-splines to calculate the value of s at t.

Let's assume now that for a given robot degree of freedom, we have a sequence of desired values $\{t_i, f_i\}_{i=1}^N$. Let's consider the following variational problem

$$\begin{aligned} \text{minimize} \quad & \int_{t_1}^{t_N} (f^m(t))^2\, dt,\ f \in L_2^m[t_1, t_N] \\ \text{under the conditions} \quad & f^{(i)}(t_j) = f_i,\ i = 1, \ldots, N, \end{aligned} \tag{2.28}$$

where $L_2^m[t_1, t_N]$ denotes the space of functions with continuous derivatives up to $f^{(m-1)}$ and square-integrable derivative $f^{(m)}$. It is well known [6, 37] that the optimization problem (2.28) has a unique solution s that belongs to the space of polynomial splines of order $2m$ with $2m-2$ continuous derivatives with a further condition that

$$s^{(m+i)}(t_1) = s^{(m+i)}(t_N) = 0,\ i = 0, \ldots, m-2. \tag{2.29}$$

Its solution can thus be written in the form (2.26). Polynomial splines that solve optimization problem (2.28) are called *natural (smoothing) splines* [5]. For cubic natural splines we obtain the additional conditions $\ddot{s}(t_1) = \ddot{s}(t_N) = 0$, while for quintic natural splines, the additional conditions are given as $s^{(3)}(t_1) = s^{(3)}(t_N) = s^{(4)}(t_1) = s^{(4)}(t_N) = 0$. We already know that the dimension of the space of polynomial splines of order $2m$ on knot sequence $\boldsymbol{\tau}$ is equal to $N + 2m - 2$. Thus we can compute the natural spline interpolating the values f_i at t_i by solving the following linear interpolation system

$$\begin{aligned} &\sum_{i=1}^{N+2m-2} \alpha_i B_i^{(m+j)}(t_1) = 0,\ j = m-2, \ldots, 0 \\ &\sum_{i=1}^{N+2m-2} \alpha_i B_i(t_j) = f_j,\ j = 1, \ldots, N \\ &\sum_{i=1}^{N+2m-2} \alpha_i B_i^{(m+j)}(t_1) = 0,\ j = 0, \ldots, m-2 \end{aligned} \tag{2.30}$$

This system can be rewritten in a matrix form

$$\mathbf{X}\boldsymbol{\alpha} = \mathbf{f}, \tag{2.31}$$

where $\mathbf{X} \in \mathbb{R}^{(N+2m-2)\times(N+2m-2)}$, $\boldsymbol{\alpha} \in \mathbb{R}^{N+2m-2}$, and $\mathbf{f} \in \mathbb{R}^{N+2m-2}$ are given by

$$\mathbf{X} = \begin{bmatrix} B_{1,2m}^{(2m-2)}(t_1) & B_{2,2m}^{(2m-2)}(t_1) & \dots & B_{N+2m-2,2m}^{(2m-2)}(t_1) \\ \vdots & \vdots & \vdots & \vdots \\ B_{1,2m}^{(m)}(t_1) & B_{2,2m}^{(m)}(t_1) & \dots & B_{N+2m-2,2m}^{(m)}(t_1) \\ B_{1,2m}(t_1) & B_{2,2m}(t_1) & \dots & B_{N+2m-2,2m}(t_1) \\ B_{1,2m}(t_2) & B_{2,2m}(t_2) & \dots & B_{N+2m-2,2m}(t_2) \\ & \ddots & \ddots & \\ B_{1,2m}(t_N) & B_{2,2m}(t_N) & \dots & B_{N+2m-2,2m}(t_N) \\ B_{1,2m}^{(m)}(t_N) & B_{2,2m}^{(m)}(t_N) & \dots & B_{N+2m-2,2m}^{(m)}(t_N) \\ \vdots & \vdots & \vdots & \vdots \\ B_{1,2m}^{(2m-2)}(t_N) & B_{2,2m}^{(2m-2)}(t_N) & \dots & B_{N+2m-2,2m}^{(2m-2)}(t_N) \end{bmatrix}, \ \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{\tilde{N}} \end{bmatrix}, \ \mathbf{f} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ f_1 \\ f_2 \\ \vdots \\ f_N \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{2.32}$$

where $\tilde{N} = N + 2m - 2$. Due to the limited support of B-splines $B_{i,2m}$, the system matrix $\mathbf{X}$ is banded with only $2m$ values different from 0 in every row. Thus the system (2.31) can be solved efficiently. The partition of unity property of B-splines further contributes to the stability of numerical computations.

For robot trajectories, it is often required that the robot starts and ends its motion with velocities and accelerations equal to zero. It turns out that variational problem

$$\begin{aligned} &\text{minimize} && \int_{t_1}^{t_N} (f^m(t))^2 \, dt, \ f \in L_2^m[t_1, t_N] \\ &\text{under the conditions} && f^{(i)}(t_j) = f_i, \ i = 1, \dots, N, \\ & && \sum_{i=j-2m+1}^{j} \alpha_i B_{i,2m}(t), \ j = 1, \dots, m-1, \end{aligned} \tag{2.33}$$

has a unique solution that belongs to the space of polynomial splines of order $2m$ on knot sequence $\boldsymbol{\tau}$. Polynomial splines that solve optimization problem (2.33) are called *complete smoothing splines* [11]. Unlike in the case of natural splines, there are no additional conditions in this case. This is not surprising as variational problem (2.33) already contains $N + 2m - 2$ conditions, which is equal to the dimensionality of the space of polynomial splines of order $2m$ on knot sequence $\boldsymbol{\tau}$. Similarly to variation problem (2.28), we can solve (2.33) by solving equation system (2.31), where the right hand side vector $\mathbf{f}$ and variable vector $\boldsymbol{\alpha}$ are exactly the same as in (2.32), whereas system matrix $\mathbf{X}$ has different first and last $m - 1$ rows. Instead of derivatives of order $m + j, \ j = 0, \dots, m - 2$, we compute the derivatives of order $1 + j, \ j = 0, \dots, m - 2$.

Interpolation works well if there is no noise in the desired positions f_j that need to be interpolated. However, if the desired positions are noisy, it is better to only approximate the desired positions. Let's now assume that we have a sequence of noisy desired positions

$\{\tilde{t}_j, \tilde{f}_j\}_{j=1}^{M}$. We assume that $M > N + 2m - 2$ and $t_1 = \tilde{t}_1,\ t_N = \tilde{t}_M$, whereas the intermediate times $\tilde{t}_2, \ldots, \tilde{t}_{M-1}$ may or may note coincide with the knot points $t_2, \ldots, t_{N-1}$. We can now solve the following overdetermined system of linear equations with equality constraints

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{X}\boldsymbol{\alpha} - \tilde{\mathbf{f}}\|^2 \\ \text{under the conditions} \quad & \sum_{i=1}^{N+2m-2} \alpha_i B_{i,2m}^{(j)}(t_1) = 0,\ j = 1, \ldots, m-1 \\ & \sum_{i=1}^{N+2m-2} \alpha_i B_{i,2m}^{(j)}(t_N) = 0,\ j = 1, \ldots, m-1 \end{aligned} \tag{2.34}$$

These types of problems are called constrained linear least squares. In (2.34), $\mathbf{X} \in \mathbb{R}^{M\times(N+2m-2)}$, $\boldsymbol{\alpha} \in \mathbb{R}^{N+2m-2}$, and $\tilde{\mathbf{f}} \in \mathbb{R}^{M}$ are computed as follows

$$\mathbf{X} = \begin{bmatrix} B_{1,2m}(\tilde{t}_1) & B_{2,2m}(\tilde{t}_1) & \ldots & B_{N+2m-2,2m}(\tilde{t}_1) \\ B_{1,2m}(\tilde{t}_2) & B_{2,2m}(\tilde{t}_2) & \ldots & B_{N+2m-2,2m}(\tilde{t}_2) \\ & \ddots & \ddots & \\ B_{1,2m}(\tilde{t}_M) & B_{2,2m}(\tilde{t}_M) & \ldots & B_{N+2m-2,2m}(\tilde{t}_M) \end{bmatrix}, \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{N+2m-2} \end{bmatrix}, \mathbf{f} = \begin{bmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \vdots \\ \tilde{f}_M \end{bmatrix} \tag{2.35}$$

The relationship between the number of measurements M and the number of knot points $N+2m-2$ determines the amount of smoothing versus interpolation. If N is too large, the polynomial spline computed by solving constrained linear least squares problem (2.34) will result in a noisy trajectory because the measurements $\{\tilde{f}_j\}_{j=1}^{M}$ will be nearly interpolated. If N is too small, the resulting trajectory will be oversmoothed and will not follow the data well. One can start with a large N so that the resulting spline nearly interpolates the data and then gradually decrease it until the resulting errors $\{\sum_{i=1}^{N+2m-2} \alpha_i B_{i,2m}(\tilde{t}_j) - \tilde{f}_j\}_{j=1}^{M}$ reach the expected standard deviation in the data. Bisection can be used to determine the optimal N.

2.3 Summary

Polynomial and spline-based trajectory representations are foundational tools in robotics for generating smooth and flexible robot motions. Their ability to ensure continuity up to the second or higher derivatives makes them particularly suitable for robotic applications where dynamic constraints and smooth transitions are essential. Polynomial trajectories, such as minimum jerk motions, provide an analytically simple and computationally efficient way to achieve point-to-point movements with zero initial and final velocities and accelerations. These trajectories guarantee straight-line paths in Cartesian space and shortest-path orientations using SLERP, which are advantageous for tasks requiring pre-

cision and minimal computational overhead. However, their limitation lies in their lack of flexibility for complex, multi-segment paths.

Splines, particularly B-splines, address this limitation by enabling the construction of complex, continuous trajectories over multiple segments. Their flexibility in handling variable knot placement and their ability to maintain high-order continuity make them a robust choice for generating smooth paths in robotics. B-splines offer local control, as changes to one segment of the trajectory only affect a small neighborhood, and their partition of unity property ensures numerical stability. Natural and complete splines further extend these benefits by providing solutions to variational problems that balance smoothness and adherence to boundary or data constraints. These properties make splines highly effective for interpolation and approximation tasks, such as fitting noisy data or generating motion paths that satisfy specified conditions.

Despite their advantages, spline-based methods have some shortcomings. The choice of the number and placement of knots can significantly impact the quality of the trajectory, and improper settings may lead to oversmoothing or noisy paths. Moreover, while modern software packages simplify spline computation, understanding their underlying mathematical principles remains critical for designing trajectories with desired properties. These challenges highlight the trade-offs between simplicity, flexibility, and computational complexity in spline-based trajectory design.

Overall, the combination of polynomials for simple tasks and splines for complex, segmented paths offers a versatile approach to trajectory generation in robotics, catering to a wide range of applications while addressing the unique requirements of robotic motion planning and control.

Chapter 3

Dynamic Movement Primitives

As explained in Section 1.1, policies defined by PD controllers and desired time-dependent trajectories are typically only valid near the desired trajectory. They lack flexibility and are not suitable for dynamic and unpredictable environments, such as our homes. As an alternative, representations based on parametric systems of autonomous differential equations (1.4) can be used.

The goal of a motion generation system is to ensure that the robot successfully completes the desired task. In many cases, this involves point-to-point movements for activities like pick-and-place operations and grasping. There are also many tasks involving periodic movements, such as polishing and repetitive pick-and-place operations. Whether the specific path taken during these tasks matters can vary depending on the application. In this chapter, we will show that these two types of movements can be formalized using attractor dynamics: point attractors for point-to-point movements and limit cycle attractors for repetitive, periodic movements.

To address these movement challenges, we describe an approach to specifying robot control policies based on autonomous dynamic systems. We establish a modeling framework that addresses both discrete and periodic movements within a unified and coherent dynamic systems framework. The resulting representation is called Dynamic Movement Primitives (DMPs), which were developed by Schaal, Ijspeert, and others [13, 14, 32, 12]. A notable feature of DMPs is the inclusion of adjustable parameters that do not compromise the stability of the dynamic system. This provides the basis for their integration with optimization techniques to generate a wide range of robot motions.

We will demonstrate that DMPs provide a comprehensive framework for the specification of robot control policies. They result in smooth kinematic control policies, which are essential for implementing robust robot behaviors. DMPs are highly suitable for controlling robots in uncertain environments because they: 1) guarantee smooth transitions in the event of sudden changes in movement description or disturbances, 2) are not explicitly dependent on time, 3) provide an appropriate framework for trajectory learning and adaptation using imitation and reinforcement learning algorithms, and 4) allow the learning of complete families of trajectories from multiple demonstrations using statistical

learning algorithms. One of the unique properties of DMPs is their ability to modulate trajectories in real time based on sensory feedback. For example, the phase-stopping mechanism enables the robot to halt the progress of the phase and track the desired trajectory even when a large position error arises. Such control responses are much more difficult to implement with time-dependent representations, such as polynomial splines.

3.1 Control Policies as Dynamic Systems

We continue by explaining theoretical foundations of dynamic movement primitives. First we define discrete movement primitives that can encode control policies for discrete point-to-point movements. This type of DMPs is based on a set of nonlinear differential equations with a well-defined attractor dynamics. There are a few different but essentially equivalent formulations in the literature, in the following we use the one outlined in [31, 12]. For a single robot degree of freedom, here denoted by y, which can either be one of the internal joint angles or one of the external task-space coordinates, the following system of linear differential equations with constant coefficients has been used to derive DMPs

$$\tau \dot{z} = \alpha_z(\beta_z(g - y) - z), \tag{3.1}$$

$$\tau \dot{y} = z. \tag{3.2}$$

Note that the auxiliary variable z is just a scaled velocity of the control variable, i. e. $\dot{y}$. The gain constants α_z, $\beta_z > 0$ have an interpretation in terms of spring stiffness and damping. With the parameters appropriately set, these equations form a globally stable linear dynamic system with g as a unique *point attractor*, which means that for any start configuration $y = y_0$, the parameter y would reach g after a transient motion, just like a stretched spring, upon release, will return to its equilibrium point [31]. $\tau > 0$ is called a *time constant* and can be used to speed up (when τ decreases) or slow down (when τ increases) the motion.

Let's analyze why the above system is useful. We start by writing down a general solution of the non-homogenous linear differential equation system (3.1) – (3.2). It is well known that the general solution of such a system can be written as a sum of the particular and homogeneous solution, i.e. $[z(t), y(t)]^T = [z_p(t), y_p(t)]^T + [z_h(t), y_h(t)]^T$. Here $[z_p(t), y_p(t)]^T$ denotes any function that solves system (3.1) – (3.2), while $[z_h(t), y_h(t)]^T$ is the general solution of the homogeneous part of Eq. (3.1) – (3.2), i. e.

$$\begin{bmatrix} \dot{z} \\ \dot{y} \end{bmatrix} = \frac{1}{\tau} \begin{bmatrix} -\alpha_z(\beta_z y + z) \\ z \end{bmatrix} = \mathbf{A} \begin{bmatrix} z \\ y \end{bmatrix}, \quad \mathbf{A} = \frac{1}{\tau} \begin{bmatrix} -\alpha_z & -\alpha_z\beta_z \\ 1 & 0 \end{bmatrix}. \tag{3.3}$$

It is easy to check that a constant function $[z_p(t), y_p(t)]^T = [0, g]^T$ solves the system (3.1) – (3.2). Additionally, it is well known that the general solution of homogeneous system (3.3)

is given by $[z_h(t), y_h(t)]^T = \exp(\mathbf{A}t)\,\mathbf{c}$. Thus, the general solution of Eq. (3.1) – (3.2) can be written as

$$\begin{bmatrix} z(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 0 \\ g \end{bmatrix} + \exp(\mathbf{A}t)\,\mathbf{c}, \tag{3.4}$$

where $\mathbf{c} \in \mathbb{R}^2$ should be calculated from the initial conditions, $[z(0), y(0)]^T = [z_0, y_0]^T$. The eigenvalues of $\mathbf{A}$ are given by $\lambda_{1,2} = \left(-\alpha_z \pm \sqrt{\alpha_z^2 - 4\alpha_z\beta_z}\right)/(2\tau)$. Solution (3.4) converges to $[0, g]^T$ if the real part of $\lambda_{1,2}$ is smaller than 0, which is true for any α_z, β_z, $\tau > 0$. The system is critically damped, which means that y converges to g without oscillating and faster than for any other choice of $\mathbf{A}$ if $\mathbf{A}$ has two negative eigenvalues. This happens at $\alpha_z = 4\beta_z$.

It is important to note here that changing τ or g does not compromise the stability of dynamic system (3.1) – (3.2). Regardless of their choice, the system remains critically damped. Only the attractor point and the speed of convergence towards the attractor point change with different τ and g.

3.1.1 Discrete (point-to-point) movements

Differential equations (3.1) – (3.2) ensure that y converges to g and can therefore be used to realize discrete point-to-point movements. To increase a rather limited set of trajectories that can be encoded by (3.1) – (3.2) and thus enable the representation of general point-to-point movements, we can add a nonlinear component to Eq. (3.1). One possibility is to add a linear combination of radial basis functions [31]

$$f(x) = \frac{\sum_{i=1}^{N} w_i \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0), \;\; \Psi_i(x) = \exp\left(-h_i\,(x - c_i)^2\right), \tag{3.5}$$

where c_i are the centers of radial basis function distributed along the trajectory and $h_i > 0$. The term $g - y_0,\; y_0 = y(0)$, is used to scale the trajectory if the initial and / or final configuration change. As long as the beginning and the end of movement are kept constant, this scaling factor has no effect and can also be omitted. A phase variable x is used in Eq. (3.5) instead of time to make the dependency of the resulting control policy on time more implicit. Its dynamics is defined by

$$\tau \dot{x} = -\alpha_x x, \tag{3.6}$$

with the initial value $x(0) = 1$. (3.6) is called a *canonical system*. The gain α_x must be positive. Given the initial condition $x(0) = 1$, we can compute the analytical solution for the first-order differential equation (3.6)

$$x(t) = \exp\left(-\frac{\alpha_x}{\tau} t\right). \tag{3.7}$$

Note that according to this definition, phase decreases from 1 to 0 over time. As shown in Section 3.5.1, the appealing property of using the phase variable x instead of explicit time is that by appropriately modifying Eq. (3.6), the evolution of time can be stopped to account for perturbations during motion. There is no need to manage the internal clock of the system.

We now define the following system of nonlinear differential equations

$$\tau \dot{z} = \alpha_z(\beta_z(g - y) - z) + f(x), \tag{3.8}$$

$$\tau \dot{y} = z. \tag{3.9}$$

In this formulation, the nonlinear function f is called a *forcing term*. It is defined as a function of phase variable x to avoid explicit time dependency. By setting the initial value of phase variable to 1, i.e. $x(0) = 1$, we can observe that the phase tends to 0 as time increases because the solution to Eq. (3.6) with the initial value $x(01) = 1$ is given by $x(t) = \exp(-\alpha_x/\tau t)$. Note that the phase variable x and consequently $f(x)$ tend to 0 as time increases. Hence the influence of the nonlinear forcing term $f(x)$ vanishes with time and the system (3.8), (3.9), (3.6) is guaranteed to converge to $[0, g]^T$, just like the system (3.1), (3.2). The control policy specified by variable y and its first- and second-order derivatives defines what we call a *Dynamic Movement Primitive* (DMP). For a robot with several degrees of freedom, each DOF is represented by its own differential equation system (3.8), (3.9), whereas the phase x defined by Eq. (3.6) remains common across all robot degrees of freedom. Note that the free parameters $\boldsymbol{\alpha}$ from Eq. (1.1) and (1.4) consists of free parameters $\{w_i\}_{i=1}^N$, g and τ.

As the differential equation system (3.8), (3.9) are independent of each other, we can write

$$\tau \dot{\mathbf{z}} = \alpha_z(\beta_z(\mathbf{g} - \mathbf{y}) - \mathbf{z}) + \mathbf{f}(x), \tag{3.10}$$

$$\tau \dot{\mathbf{y}} = \mathbf{z}, \tag{3.11}$$

where $\mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ are the vectors combining all robot joins and auxiliary parameters while n is the number of degrees of freedom. Note that each degree of freedom has its own forcing term defined as a superposition of radial basis functions, which are gathered in vector forcing term $\mathbf{f}(x) = [f_1(x), \ldots, f_n(x)]^{\mathrm{T}}$,

The parameters c_i and h_i of Eq. (3.5) are usually defined so that Gaussian basis functions are uniformly distributed in the time domain. This can be achieved by setting the following distribution pattern in the phase domain, given N basis functions

$$c_i = \exp\left(-\alpha_x \frac{i-1}{N-1}\right), \; i = 1, \ldots, N, \tag{3.12}$$

$$h_i = \frac{2}{(c_{i+1} - c_i)^2}, \; i = 1, \ldots, N-1, \; h_N = h_{N-1}. \tag{3.13}$$

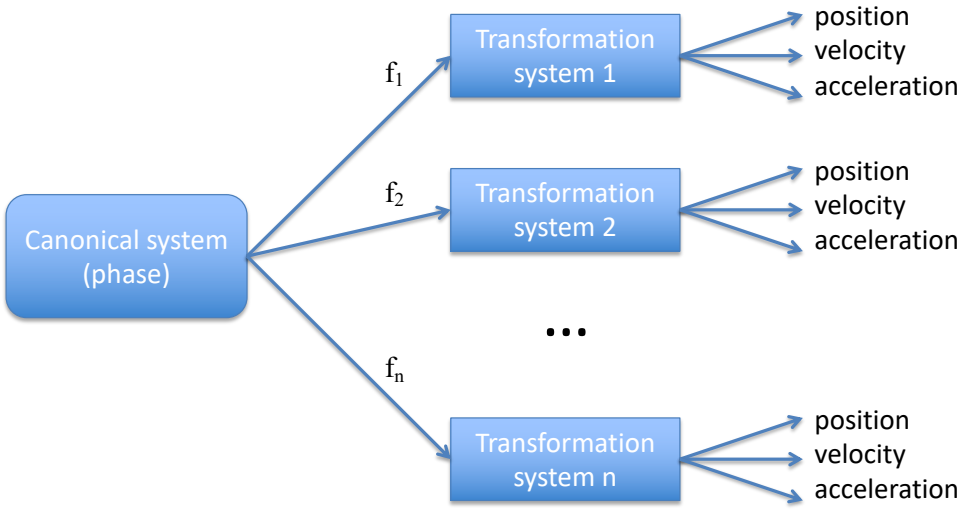

Figure 3.1: Schematic presentation of the canonical system and the transformation system that together form a DMP. The canonical system generates the phase that drives the transformation systems, which generate motor commands for all degrees of freedom.

Note that $c_1 = 1 = x(0)$ and $c_N = \exp(-\alpha_x) = x(t_T)$, where t_T is the time at which the dynamic movement primitive ends.

System (3.10), (3.11) is called *transformation system*. In dynamic movement primitives, the canonical system (3.6) and transformation systems work together to model and generate complex movements. The canonical system plays the role of a temporal scaling mechanism, controlling the progression of the phase of the movement. It provides a phase variable that evolves monotonically from a starting point to an end point. This phase variable serves as a reference for the transformation system, ensuring that the movement is time-independent and can be adjusted for different durations without altering the fundamental characteristics of the motion.

The transformation system, on the other hand, is responsible for shaping the movement trajectory itself. It takes the phase variable provided by the canonical system and applies it to a set of basis functions or nonlinear terms that define the desired trajectory. These basis functions are weighted and combined to produce the movement, allowing the DMP to generate complex, smooth, and adaptable trajectories. The transformation system also includes components that ensure the generated trajectory smoothly converges to a predefined goal, making the motion robust to changes in starting or goal positions.

Together, the canonical and transformation systems allow DMPs to encode and reproduce flexible, goal-directed movements that can be easily modified for different tasks and environmental conditions, making them highly effective for robotic learning and control applications. Their interplay is shown in Fig. 3.1.

3.1.2 Periodic (rhythmic) movements

In the case of periodic movements, we form the forcing term by a superposition of periodic functions [31]

$$\mathbf{f}(\phi) = \frac{\sum_{i=1}^{N} \mathbf{w}_i \Gamma_i(\phi)}{\sum_{i=1}^{N} \Gamma_i(\phi)} r,\ \Gamma_i(\phi) = \exp\left(h_i\left(\cos\left(\phi - c_i\right) - 1\right)\right). \tag{3.14}$$

Here r is the amplitude of the oscillator and $h_i > 0$. Writing $\Omega = 1/\tau$, Eqs. (3.10) and (3.11) are replaced by

$$\dot{\mathbf{z}} = \Omega\left(\alpha_z(\beta_z(\mathbf{g} - \mathbf{y}) - \mathbf{z}) + \mathbf{f}(\phi)\right), \tag{3.15}$$

$$\dot{\mathbf{y}} = \Omega\mathbf{z}. \tag{3.16}$$

The phase variable ϕ has been introduced in this case to avoid the explicit dependency on time. The phase is assumed to move with constant speed

$$\dot{\phi} = \Omega, \tag{3.17}$$

where Ω is the frequency of oscillation. In the case of periodic functions, we distribute the basis functions uniformly along the phase interval $[0, 2\pi]$, i. e.

$$c_i = \frac{(2 * i - 1)\pi}{2N},\ i = 1, \ldots, N, \tag{3.18}$$

and $h_i = 2.5\pi/N, \forall i$. The phase interval $[0, 2\pi]$ describes one period of motion.

3.2 Computing DMP parameters from a single demonstration

Next we will show that Eq. (3.8), (3.9) , and (3.6) can be used to approximate any discrete point-to-point movement.

Discrete DMPs were designed to provide a representation with free parameters that enables accurate representation of a desired point-to-point movement and at the same time permits the modulation of different properties of the encoded trajectory. The shape parameters w_i are the key when it comes to accurately representing a desired motion trajectory.

As explained in Section 1.3, programming by demonstration is an effective way to specifying motion trajectories. In this approach, a human instructor demonstrates the desired motion using one of the approaches described in Section 1.4. We obtain the following measurements

$$\{\mathbf{y}_d(t_j),\ t_j\}_{j=0}^{T}, \tag{3.19}$$

where $\mathbf{y}_d(t_j) \in \mathbb{R}^n$ are the desired joint angles and $T+1$ is the number of the measured robot configurations on the robot trajectory.

To compute a DMP (Dynamic Movement Primitive) from this data, we must first numerically compute the velocities and accelerations from the measured data. Central finite differences can be used for this purpose. In the general case of non-uniformly sampled data, we obtain the following formulas:

$$\dot{\mathbf{y}}_d(t_j) = \frac{(t_j - t_{j-1})(\mathbf{y}_d(t_{j+1}) - \mathbf{y}_d(t_j) + (t_{j+1} - t_j)(\mathbf{y}_d(t_j) - \mathbf{y}_d(t_{j-1}))}{(t_{j+1} - t_{j-1})(t_{j+1} - t_j)}, \tag{3.20}$$

$$\ddot{\mathbf{y}}_d(t_j) = \frac{2}{(t_{j+1} - t_j)(t_j - t_{j-1})} \left(\frac{\mathbf{y}_d(t_{j+1}) - \mathbf{y}_d(t_j)}{t_{j+1} - t_j} + \frac{\mathbf{y}_d(t_j) - \mathbf{y}_d(t_{j-1})}{t_{j+1} - t_{j-1}} \right). \tag{3.21}$$

For uniform sampling, i.e. $t_{j+1} - t_j = t_j - t_{j-1} = \Delta t$, the formulas simplify significantly:

$$\dot{\mathbf{y}}_d(t_j) = \frac{\mathbf{y}_d(t_{j+1}) - \mathbf{y}_d(t_{j-1})}{2\Delta t}, \tag{3.22}$$

$$\ddot{\mathbf{y}}_d(t_j) = \frac{\mathbf{y}_d(t_{j+1}) - 2\mathbf{y}_d(t_j) + \mathbf{y}_d(t_{j-1})}{\Delta t^2}. \tag{3.23}$$

These formulas are valid for $j = 1, \ldots, T-1$. If the robot should be at rest at the beginning and end of the motion, we can set:

$$\dot{\mathbf{y}}_d(t_0) = \mathbf{y}_d(t_T) = \ddot{\mathbf{y}}_d(t_0) = \ddot{\mathbf{y}}(t_T) = 0.$$

The demonstrated data (3.19) must be consistent with this assumption for the real robot to be at rest at the beginning and end of the motion. If the initial and final velocities are not known, we can estimate them using left and right finite differences:

$$\dot{\mathbf{y}}_d(t_0) = \frac{\mathbf{y}_d(t_1) - \mathbf{y}_d(t_0)}{t_1 - t_0}, \tag{3.24}$$

$$\dot{\mathbf{y}}_d(t_T) = \frac{\mathbf{y}_d(t_T) - \mathbf{y}_d(t_{T-1})}{t_T - t_{T-1}}. \tag{3.25}$$

Similarly, for accelerations we obtain

$$\ddot{\mathbf{y}}_d(t_1) = \frac{2}{(t_2 - t_0)(t_1 - t_0)} \left(\frac{\mathbf{y}_d(t_2) - \mathbf{y}_d(t_1)}{t_2 - t_1} - \frac{\mathbf{y}_d(t_1) - \mathbf{y}_d(t_0)}{t_1 - t_0} \right), \tag{3.26}$$

$$\ddot{\mathbf{y}}_d(t_T) = \frac{2}{(t_T - t_{T-2})(t_T - t_{T-1})} \left(\frac{\mathbf{y}_d(t_T) - \mathbf{y}_d(t_{T-1})}{t_T - t_{T-1}} - \frac{\mathbf{y}_d(t_{T-1}) - \mathbf{y}_d(t_{T-2})}{t_{T-1} - t_{T-2}} \right). \tag{3.27}$$

We obtain the following data that can be used to compute a DMP

$$\{\mathbf{y}_\mathrm{d}(t_j),\ \dot{\mathbf{y}}_\mathrm{d}(t_j),\ \ddot{\mathbf{y}}_\mathrm{d}(t_j)\}_{j=0}^{T}, \tag{3.28}$$

where $\mathbf{y}_\mathrm{d}(t_j),\ \dot{\mathbf{y}}_\mathrm{d}(t_j),\ \ddot{\mathbf{y}}_\mathrm{d}(t_j)$ are the measured positions, velocities, and accelerations on

the training trajectory and $T+1$ is the number of sampling points.

Computing DMP parameters by solving a system of linear equations

To compute the DMP parameters, we now rewrite the system of two first-order linear equations (3.8) – (3.9) as a second-order equation. This is done by replacing $\mathbf{z}$ with $\tau\dot{\mathbf{y}}$ in Eq. (3.8)

$$\tau^2\ddot{\mathbf{y}} + \alpha_z\tau\dot{\mathbf{y}} - \alpha_z\beta_z(\mathbf{g} - \mathbf{y}) = \mathbf{f}(x), \tag{3.29}$$

with the components of $\mathbf{f} = [f_1, \ldots, f_n]^T$ defined as in Eq. (3.5). Note that time constant τ is the same for all degrees of freedom. A possible choice is $\tau = t_T$, where t_T is the duration of the training movement. On the other hand, the attractor point $\mathbf{g}$ varies across the degrees of freedom. It can be extracted directly from the data: $\mathbf{g} = \mathbf{y}_{\mathrm{d}}(t_T)$.

For every degree of freedom k, $1 \leq k \leq n$ and using the desired values $\mathbf{y}_{\mathrm{d}}(t_j)$, $\dot{\mathbf{y}}_{\mathrm{d}}(t_j)$, $\ddot{\mathbf{y}}_{\mathrm{d}}(t_j)$, we can compute the following values

$$F_{k,j} = \tau^2\ddot{y}_{\mathrm{d},k}(t_j) + \alpha_z\tau\dot{y}_{\mathrm{d},k}(t_j) - \alpha_z\beta_z(g_k - y_{\mathrm{d},k}(t_j)), \tag{3.30}$$

By considering the right hand side of differential equation (3.29) and the definition of function $\mathbf{f}(x)$ in (3.5), we obtain the following system of linear equations

$$F_{k,j} = x_j(g_k - y_{0,k})\sum_{i=1}^{N}\frac{w_{k,i}\Psi_i(x_j)}{\sum_{i=1}^{N}\Psi_i(x_j)},\ j = 0, \ldots, T, \tag{3.31}$$

where $\mathbf{y}_0 = \mathbf{y}_{\mathrm{d}}(t_0)$ and the phase sampling points x_j, $j = 0, \ldots, T$, are computed using Eq. (3.7). Writing

$$\mathbf{F}_k = \begin{bmatrix} F_{k,0} \\ \ldots \\ F_{k,T} \end{bmatrix},\ \mathbf{F}_k \in \mathbb{R}^{T+1},\ \mathbf{w}_k = \begin{bmatrix} w_{k,1} \\ \ldots \\ w_{k,N} \end{bmatrix},\ \mathbf{w}_k \in \mathbb{R}^N,$$

we can rewrite the equation system (3.31) in the matrix form

$$\mathbf{X}\mathbf{w}_k = \frac{1}{g_k - y_{k,0}}\mathbf{F}_k, \tag{3.32}$$

where the system matrix $\mathbf{X} \in \mathbb{R}^{(T+1)\times N}$ is given by

$$\mathbf{X} = \begin{bmatrix} \frac{\Psi_1(x_0)}{\sum_{i=1}^{N}\Psi_i(x_0)}x_0 & \ldots & \frac{\Psi_N(x_0)}{\sum_{i=1}^{N}\Psi_i(x_0)}x_0 \\ \ldots & \ldots & \ldots \\ \frac{\Psi_1(x_T)}{\sum_{i=1}^{N}\Psi_i(x_T)}x_T & \ldots & \frac{\Psi_N(x_T)}{\sum_{i=1}^{N}\Psi_i(x_T)}x_T \end{bmatrix}. \tag{3.33}$$

The equation system (3.32) need to be solved to estimate the weights $\mathbf{w}_k$ specifying a DMP encoding the k-th degree of freedom of the desired motion. The weights $\mathbf{w}_k$ can

be calculated by solving the above system of linear equations in a least-squares sense. This system is overdetermined because typically $T > N$. The full DMP is specified by computing separate DMPs for every robot degree of freedom.

The resulting DMP ensures that the robot reaches the attractor point $\mathbf{g}$ at time t_T. Since discrete DMPs have been designed to represent discrete point-to-point movements, the training movement must come to a full stop at the end of the demonstration if the robot is to stay at the attractor point after t_T. If any other type of motion is approximated by a DMP, the robot will overshoot the attractor point and returned back to it after the dynamics of the second-order system of differential equations starts dominating the motion. At least theoretically, the velocity does not need to be zero at the beginning of movement, but this is not usual in real PbD systems.

In the equations above, α_x, α_z, and β_z are constant. They are set so that the convergence of the underlying dynamic system is ensured, which is for example the case if we set $\alpha_x = 2$, $\beta_z = 3$, $\alpha_z = 4\beta_z = 12$.

Computing DMP parameters by locally weighted regression

Ijspeert et al. [14, 12] suggested to estimate the DMP weights $w_{k,i}$ independently of each other using locally weighted regression [1]. Locally weighted regression computes the weights $w_{k,i}$ using equations

$$F_{k,j} = x_j(g_k - y_{0,k})w_{k,i}. \tag{3.34}$$

This equation is obtained from (3.31) by assuming the constant forcing term $w_{k,i}$. The values of the kernel function $\Psi_i(x_j)$ are in this case used to weight the importance of each of these equations, which results in the locally weighted least squares problem

$$\sum_{j=0}^{T} \Psi_i(x_j) \left(F_{k,j} - x_j(g_k - y_{0,k})w_{k,i}\right)^2. \tag{3.35}$$

It is easy to show that this optimization problem is solved by

$$w_{k,i} = \frac{\boldsymbol{\xi}_k^{\mathrm{T}} \boldsymbol{\Gamma}_i \mathbf{F}_k}{\boldsymbol{\xi}_k^{\mathrm{T}} \boldsymbol{\Gamma}_i \boldsymbol{\xi}_k}, \tag{3.36}$$

where

$$\boldsymbol{\xi}_k = \begin{bmatrix} x_0(g_k - y_{0,k}) \\ x_1(g_k - y_{0,k}) \\ \vdots \\ x_T(g_k - y_{0,k}) \end{bmatrix}, \; \boldsymbol{\Gamma}_i = \begin{bmatrix} \Psi_i(x_0) & 0 & \dots & 0 \\ 0 & \Psi_i(x_1) & \dots & 0 \\ & \ddots & \ddots & \\ 0 & 0 & \dots & \Psi_i(x_T) \end{bmatrix} \tag{3.37}$$

Note that no matrix inversion is needed to compute $\mathbf{w}_{k,i}$ in this case.

The advantage of applying a full linear system (3.32) to estimate $\mathbf{w}$ is that this way we can approximate trajectories more accurately with less basis function by considering the interplay between the neighboring basis functions Ψ_i of Eq. (3.5). However, the

separate estimation of $w_{k,i}$ by solving optimization problem (3.35) also has its advantages, especially in the presence of noise when overfitting can become a problem [13].

Incremental estimation of periodic DMPs

In case of periodic DMPs, the equation system that need to be solved is quite similar. The second order differential equation system equivalent to (3.29) becomes

$$\frac{1}{\Omega^2}\ddot{\mathbf{y}} + \frac{\alpha_z}{\Omega}\dot{\mathbf{y}} - \alpha_z\beta_z(\mathbf{g} - \mathbf{y}) = \mathbf{f}(\phi), \tag{3.38}$$

with the components of $\mathbf{f} = [f_1, \ldots, f_n]^T$ defined as in Eq. (3.14). From the measurements we obtain

$$F_{k,j} = \frac{1}{\Omega^2}\ddot{y}_{\mathrm{d},k}(t_j) + \frac{\alpha_z}{\Omega}\dot{y}_{\mathrm{d},k}(t_j) - \alpha_z\beta_z(g_k - y_{\mathrm{d},k}(t_j)), \tag{3.39}$$

while the linear equation system (3.31) transforms to

$$F_{k,j} = r\sum_{i=1}^{N}\frac{w_{k,i}\Psi_i(\phi_j)}{\sum_{i=1}^{N}\Psi_i(\phi_j)},\ j = 0, \ldots, T,\ k = 1, \ldots .n. \tag{3.40}$$

The phase variables are simply computed as $\phi_j = \Omega t_j$, which we obtain by solving Eq. (3.17) with the initial value $\phi_0 = 0$.

Besides the weights $w_{k,j}$, there are three additional parameters that need to be estimated from the data: the amplitude r, the center of oscillation $\mathbf{g}$, and the frequency of oscillation Ω. r and $\mathbf{g}$ can easily be estimated from the data:

$$\mathbf{g} = \frac{1}{T+1}\sum_{j=0}^{T}\mathbf{y}_{\mathrm{d}}(t_j), \tag{3.41}$$

$$r = \max_{0\le j\le T}\|\mathbf{y}_{\mathrm{d}}(t_j) - \mathbf{g}\|. \tag{3.42}$$

The estimation of frequency Ω is more involved and is described in the following section.

Assuming r, $\mathbf{g}$, and Ω are known, the shape parameters $\mathbf{w}$ could be computed by solving the equation system (3.40)

$$\mathbf{X}\mathbf{w}_k = [F_{k,0}, F_{k,1}, \ldots, F_{k,T}]^{\mathrm{T}} \tag{3.43}$$

with the system matrix

$$\mathbf{X} = r\begin{bmatrix} \frac{\Gamma_1(\phi_0)}{\sum_{i=1}^{N}\Gamma_i(\phi_0)} & \cdots & \frac{\Gamma_N(\phi_0)}{\sum_{i=1}^{N}\Gamma_i(\phi_0)} \\ \vdots & \ddots & \vdots \\ \frac{\Gamma_1(\phi_T)}{\sum_{i=1}^{N}\Gamma_i(\phi_T)} & \cdots & \frac{\Gamma_N(\phi_T)}{\sum_{i=1}^{N}\Gamma_i(\phi_T)} \end{bmatrix}. \tag{3.44}$$

However, periodic motions do not necessarily have an ending and new data is supplied continuously. In such cases, shape parameters $\mathbf{w}$ can be estimated by applying recursive least squares with a forgetting factor. For the k-th degree of freedom, we obtain the following iteration

$$\mathbf{P}_j = \frac{1}{\lambda}\left(\mathbf{P}_{j-1} - \frac{\mathbf{P}_{j-1}\mathbf{x}_j\mathbf{x}_j^T\mathbf{P}_{j-1}}{\lambda + \mathbf{x}_j^T\mathbf{P}_{j-1}\mathbf{x}_j}\right), \tag{3.45}$$

$$\mathbf{w}_{k,j} = \mathbf{w}_{k,j-1} + (F_{k,j} - \mathbf{x}_j^T\mathbf{w}_{k,j-1})\mathbf{P}_j\mathbf{x}_j, \tag{3.46}$$

where j is the iteration index, $\mathbf{w}_{k,j} = [w_{1,j}, \ldots, w_{N,j}]^T \in \mathbb{R}^N$ and $\mathbf{P}_j \in\in \mathbb{R}^{N\times N}$ are the current estimates for the shape parameters and auxiliary covariance matrix $\mathbf{P}$, respectively, with the initial estimates $\mathbf{P}_0 = \mathbf{I}$, $\mathbf{w}_{k,0} = 0$. Note that the covariance matrix only depends on the choice of basis functions and is independent of the measurements $\{\mathbf{y}_d(t_j),\ t_j\}_{j=0}^T$. It is therefore the same for all robot degrees of freedom. The desired forcing term values $F_{k,j}$ are as in Eq. (3.39), $\mathbf{x}_j$ is the M dimensional column vector associated with the corresponding row of the system matrix $\mathbf{X}$ from Eq. (3.44), $0 < \lambda \leq 1$ is the forgetting factor, and the final weights are given as $\mathbf{w} = \mathbf{w}_T$. The forgetting factor should be set to 1 if only one period of example motion is available, but in such cases the standard least-squares solution is preferable.

The recursive formulation is typically used when learning periodic movements online, i. e. when estimating the DMP parameters while the human demonstrates the desired periodic motion. If in parallel the robot executes the current estimate of the periodic DMP, the human demonstrator can observe its performance and continue demonstrating until a satisfactory performance has been achieved.

3.2.1 Adaptive frequency oscillators

Unlike the time duration τ in the case of discrete movements, the frequency of oscillation is not directly observable in the data. To estimate the frequency, Righetti et al. [29] suggested to replace the constant speed assumption (3.17) by a system

$$\dot{\phi}_i = \Omega_i - Ke\,(t)\sin\,(\phi_i)\,, \tag{3.47}$$

$$\dot{\Omega}_i = -Ke\,(t)\sin\,(\phi_i)\,, \tag{3.48}$$

$$\dot{\alpha}_i = \eta\cos\,(\phi_i)\,e\,(t)\,, \tag{3.49}$$

where $e\,(t) = y_{\mathrm{d}}(t) - \hat{y}(t)$ and $\hat{y}(t) = \sum_{i=1}^{L}\alpha_i\cos\,(\phi_i)$. Note that if $e\,(t) = 0$, the system (3.47) – (3.49) becomes equivalent to (3.17). It has been shown that by integrating this system, the frequencies Ω_i contained in the observed motion trajectory can be estimated. The most significant frequency is selected as the base or fundamental frequency Ω for the DMP (see [8] for the integration of adaptive frequency oscillators with DMPs).

Alternatively, Petrič et al. [27] suggested to extract the motion frequency based on an adaptive frequency oscillator combined with an adaptive Fourier series. This results in a

second order system of differential equations

$$\dot{\Omega} = -Ke\sin(\phi), \tag{3.50}$$
$$\dot{\phi} = \Omega - Ke\sin(\phi), \tag{3.51}$$

where Ω is the extracted frequency, ϕ is the phase, K is the coupling constant and $e(t) = y_{\mathrm{d}}(t) - \hat{y}(t)$ is the difference between the actual control value y and the estimated control value $\hat{y}$, where $\hat{y}$ is given by

$$\hat{y}(t) = \sum_{c=0}^{m} (\alpha_c \cos(c\phi(t)) + \beta_c \sin(c\phi(t))). \tag{3.52}$$

Here, m denotes the size of the Fourier series. The weights α_c and β_c are updated according to the following learning rule

$$\dot{\alpha}_c = \eta \cos(c\phi)e, \tag{3.53}$$
$$\dot{\beta}_c = \eta \sin(c\phi)e, \tag{3.54}$$

where η is the learning constant (see [27] for more details). The size of the Fourier series is normally set to a low value such as $m = 10$.

3.2.2 Integration of DMPs for robot control

To control the robot, we need to integrate the DMP equations (3.10), (3.11), and (3.6) for discrete movements or (3.15), (3.16), and (3.17), for periodic movements. The simplest method to implement the integration is the Euler's method, which is a first-order numerical technique for approximating solutions to ordinary differential equations by iteratively advancing the solution using the derivative at each step. Beginning with an initial value, it estimates the next point based on the current value and its derivative. This procedure is repeated over the desired interval to construct an approximate solution. For discrete DMPs, we obtain the following integration formulae

$$\mathbf{z}_{j+1} = \mathbf{z}_j + \frac{1}{\tau}\left(\alpha_z(\beta_z(\mathbf{g} - \mathbf{y}_j) - \mathbf{z}_j) + \mathbf{f}(x_j)\right)\Delta t, \tag{3.55}$$
$$\mathbf{y}_{j+1} = \mathbf{y}_j + \frac{1}{\tau} z_j \Delta t, \tag{3.56}$$
$$x_{j+1} = x_j - \frac{\alpha_x}{\tau} x_j \Delta t, \tag{3.57}$$

where $\Delta t > 0$ is the integration step. For the initial values we set $x_0 = 1$, $\mathbf{y}_0$ should be set to the robot's initial configuration, while $z_0 = \tau \mathbf{v}_0$, where $\mathbf{v}_0$ is the initial robot velocity. Usually the initial velocity is equal to 0. The integration step Δt can be set to the servo rate of the robot controller. If the integration accuracy at this rate is not sufficient, Δt can be set to a smaller value, with the robot's servo rate being a multiple

of the integration rate. In this case, only the integrated values computed at the robot controller's servo rate are used to control the robot.

For periodic DMPs, we obtain similar integration formulae:

$$\mathbf{z}_{j+1} = \mathbf{z}_j + \Omega\left(\alpha_z(\beta_z(\mathbf{g} - \mathbf{y}_j) - \mathbf{z}_j) + \mathbf{f}(\phi_j)\right)\Delta t, \tag{3.58}$$

$$\mathbf{y}_{j+1} = \mathbf{y}_j + \Omega z_j \Delta t, \tag{3.59}$$

$$x_{j+1} = x_j + \Omega \Delta t. \tag{3.60}$$

In this case, the initial phase value is set to $\phi_0 = 0$, whereas $\mathbf{y}_0$ and $\mathbf{z}_0$ are initialized in the same way as for discrete DMPs.

Besides decreasing the integration step, the accuracy of integration can also be improved by replacing Euler's method with one of the Runge-Kutta methods, which improve upon Euler's method by using multiple intermediate evaluations of the derivative to achieve higher accuracy without reducing the step size. While Euler's method is a first-order method (linear error reduction with step size), the Runge-Kutta methods, such as the commonly used fourth-order Runge-Kutta (RK4), achieve higher-order accuracy (quartic error reduction with step size) by considering the derivative at multiple points within the step interval. This results in better approximations of the true DMP.

An important property of DMPs is that they form a control policy, which means that even if the current state of the robot is not close to the trajectory used for training, the DMP system generates motor commands that smoothly move the robot towards the trained motion. For every phase x, an attractor field is generated that moves the robot

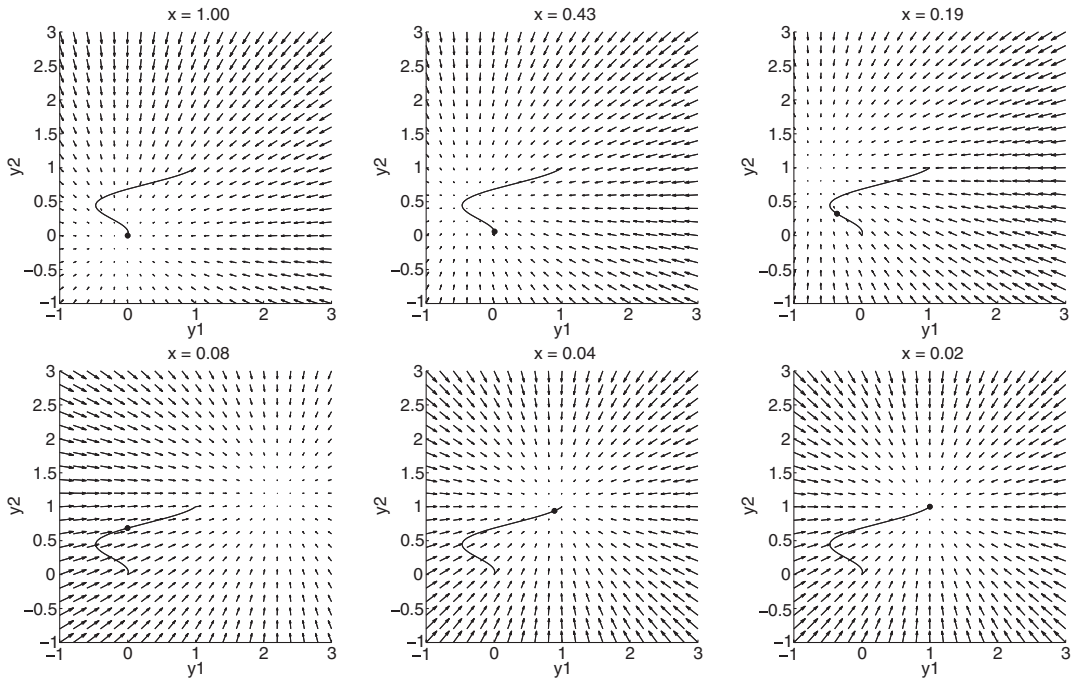

Figure 3.2: Attractor landscape generated by a DMP (from [12])

towards the correct part of the trajectory. This is illustrated in Fig. 3.2 for a system with two degrees of freedom.

3.3 Cartesian Space DMPs

As explained in Section 1.2, all minimal, i.e. 3-D representations of orientation contain singularities and are therefore not suitable for representing orientation trajectories. Non-minimal representations of SO(3) should be used instead. This makes it more difficult to integrate differential equations on SO(3) because general integration methods do not have any information about the structure of SO(3) and can cause the integrated parameters to depart from the constraint manifold. A different approach should therefore be used to formulate and integrate DMPs for Cartesian space trajectories [36].

While positions are represented as vectors and involve linear operations, orientations are typically represented using quaternions, which require non-linear operations like quaternion multiplication. This difference necessitates using quaternion algebra in the transformation system to ensure valid and smooth rotational trajectories.

3.3.1 Quaternion based DMPs

As explained in Section 1.2.2, orientations can be represented by unit quaternions $\mathbf{q} = v + \mathbf{u} \in \mathrm{S}^3$, where S^3 is a unit sphere in $\mathbb{R}^4$, $v \in \mathbb{R}$, $\mathbf{u} \in \mathbb{R}^3$. Quaternions provide a singularity-free, non minimal representation of orientation with only four parameters. They have been utilized to represent orientation in various contexts including robot control [3] and vision-based tracking [34]. This representation is not unique because unit quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same orientation. We can thus fully specify orientation trajectories by specifying unit quaternions $\mathbf{q}$ at all times t.

To fully describe a movement of the robot's end effector in Cartesian space, we can specify its position trajectory $\mathbf{p}(t) \in \mathbb{R}^3$ and orientation trajectory $\mathbf{q}(t) \in \mathrm{S}^3$. Each of the 4 quaternion parameters could be represented by its own DMP system (3.8) – (3.9). However, the coefficients of the orientation trajectory $\mathbf{q}(t)$ are not independent of each other, thus if we integrate them independently, the integrated quaternion will gradually start deviating from the unit quaternion space. Instead, we transform the standard DMP equations (3.10) – (3.11) into the quaternion form as follows

$$\tau \dot{\boldsymbol{\eta}} = \alpha_z \left(\beta_z 2 \log \left(\mathbf{g}_o * \overline{\mathbf{q}} \right) - \boldsymbol{\eta} \right) + \mathbf{f}_o(x), \tag{3.61}$$

$$\tau \dot{\mathbf{q}} = \frac{1}{2} \boldsymbol{\eta} * \mathbf{q}, \tag{3.62}$$

where $\mathbf{g}_o \in \mathrm{S}^3$ denotes the goal quaternion orientation, while the time constant $\tau > 0$ and the control gains $\alpha_z, \beta_z > 0$ have the same role as in standard DMPs specified by (3.10) – (3.11). x is again the phase variable defined by Eq. (3.6). See Section 1.2.2 for the definition of quaternion conjugation (1.22), quaternion product (1.20), and quaternion

logarithm (1.41). The auxiliary variable $\boldsymbol{\eta} \in \mathbb{R}^3$ is treated as quaternion with 0 scalar part in (3.62). The motivation for equation (3.61) is provided by observing that $\mathbf{g} - \mathbf{y}$ in Eq. (3.10) can be interpreted as linear velocity that takes $\mathbf{y}$ to $\mathbf{g}$ within unit time, just like $2\log(\mathbf{g}_o * \overline{\mathbf{q}})$ is the angular velocity that takes orientation $\mathbf{q}$ to orientation $\mathbf{g}_o$ within unit time, as shown in (1.42). Equation (3.62) is motivated by the relationship between quaternion derivative $\dot{\mathbf{q}}$ and angular velocity $\boldsymbol{\omega}$, $\dot{\mathbf{q}} = 1/2\,\boldsymbol{\omega} * \mathbf{q}$, as shown in Eq. (1.47). From (3.62) and (1.47) we can thus derive that the auxiliary variable $\boldsymbol{\eta}$ is a scaled angular velocity,

$$\boldsymbol{\eta} = \tau\boldsymbol{\omega}. \tag{3.63}$$

The nonlinear forcing term

$$\mathbf{f}_o(x) = \mathbf{D}_o \frac{\sum_{i=1}^{N} \mathbf{w}_i^o \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} x \tag{3.64}$$

contains free parameters $\mathbf{w}_i^o \in \mathbb{R}^3$, which need to be determined to follow any given orientation trajectory. $\mathbf{D}_o = \mathrm{diag}(2\log(\mathbf{g}_o * \overline{\mathbf{q}}_0)) \in \mathbb{R}^{3\times 3}$ is a diagonal matrix with scaling factors $2\log(\mathbf{g}_o * \overline{\mathbf{q}}_0)$ on the diagonal, where $\mathbf{q}_0$ is the initial orientation on the trajectory. The modulation by $\mathbf{D}_o$ leads to useful scaling properties of the DMP system when the goal configuration $\mathbf{g}_o$ and consequently the amplitude of the movement change (see also Section 3.3.4).

Note that the logarithmic map defined on S^3 is largely singularity-free except for the singularity at a single quaternion $\mathbf{q} = -1 + [0, 0, 0]^{\mathrm{T}}$, which corresponds to unlikely rotation by a full angle 2π. As full rotations are rare in orientation trajectories, this singularity usually does not cause any issues when specifying orientation trajectories in unit quaternion space.

The data for computing a Cartesian space DMP can be gathered through programming by demonstration, which involves mapping the demonstrated motion to the positions and orientations of the end-effector

$$\{\mathbf{p}_d(t_j),\, \mathbf{q}_d(t_j),\, t_j\}_{j=0}^{T}. \tag{3.65}$$

Here $\mathbf{p}_d(t_j) \in \mathbb{R}^3$ are the desired positions of the end-effector and $\mathbf{q}_d(t_j) \in \mathrm{S}^3$ the desired orientations specified by unit quaternions. The linear velocitoes $\dot{\mathbf{p}}_d(t_j)$ and accelerations $\ddot{\mathbf{p}}_d(t_j)$ can be obtained by numerical differentiation as explained in Section 3.2. To compute also the desired angular velocities $\boldsymbol{\omega}_d(t_j)$ and angular accelerations $\dot{\boldsymbol{\omega}}_d(t_j)$, we first numerically differentiate $\mathbf{q}_d(t_j)$ using central difference formula

$$\dot{\mathbf{q}}_d(t_j) = \frac{(t_j - t_{j-1})(\mathbf{q}_d(t_{j+1}) - \mathbf{q}_d(t_j) + (t_{j+1} - t_j)(\mathbf{q}_d(t_j) - \mathbf{q}_d(t_{j-1}))}{(t_{j+1} - t_{j-1})(t_{j+1} - t_j)}. \tag{3.66}$$

These formulas are valid for $1 \leq j \leq T - 1$. If the angular velocity is 0 at the beginning and the end of the motion, we can set $\dot{\mathbf{q}}_d(t_0) = \dot{\mathbf{q}}_d(t_T) = 0$, otherwise we have to use left

and right finite differences like in Eq. (3.24) and (3.25) to compute $\dot{\mathbf{q}}_d(t_0)$ and $\dot{\mathbf{q}}_d(t_T)$. Using Eq. (1.47), we can now compute the desired angular velocities

$$\boldsymbol{\omega}_d(t_j) = 2\,\dot{\mathbf{q}}_d(t_j) * \overline{\mathbf{q}}_d(t_j). \tag{3.67}$$

The desired angular accelerations $\dot{\boldsymbol{\omega}}_d(t_j)$ are then obtained using the same central difference formula like in (3.20) and (3.66). This way we obtain all of the data needed to compute a quaternion DMP

$$\{\mathbf{q}_d(t_j),\, \boldsymbol{\omega}_d(t_j),\, \dot{\boldsymbol{\omega}}_d(t_j),\, t_j\}_{j=0}^{T}. \tag{3.68}$$

To compute the parameters $\mathbf{w}_i^o$, we first rewrite Eq. (3.61)

$$\tau^2\dot{\boldsymbol{\omega}} - \alpha_z\left(\beta_z 2\log\left(\mathbf{g}_o * \overline{\mathbf{q}}\right) - \tau\boldsymbol{\omega}\right) = \mathbf{f}_o(x), \tag{3.69}$$

The parameters $\mathbf{w}_i^o$ can then be computed by solving the following system of linear equations

$$\frac{\sum_{i=1}^{N}\mathbf{w}_i^o\Psi_i(x(t_j))}{\sum_{i=1}^{N}\Psi_i(x(t_j))}x(t_j) = \mathbf{D}_o^{-1}\left(\tau^2\,\dot{\boldsymbol{\omega}}_d(t_j) - \alpha_z\left(\beta_z 2\log\left(\mathbf{g}_o * \overline{\mathbf{q}}_d(t_j)\right) - \tau\boldsymbol{\omega}_d(t_j)\right)\right), \tag{3.70}$$

where phases $x(t_j)$ are obtained as in (3.83). Similarly as for the standard DMPs, we can compute the goal orientation $\mathbf{g}_o = \mathbf{q}_d(t_T)$ and time constant $\tau = t_T - t_0$ directly from the data.

Pastor et al. [26] used in Eq. (3.61) just the vector part of the quaternion product $v + \mathbf{u} = \mathbf{g}_o * \overline{\mathbf{q}}$, i. e. $\mathbf{u} \in \mathbb{R}^3$, instead of $2\log\left(\mathbf{g}_o * \overline{\mathbf{q}}\right)$. A similar type of error was used also in [38]. Defining $\mathbf{u} = \mathrm{vec}\left(\mathbf{g}_o * \overline{\mathbf{q}}\right)$, their system can be written as

$$\tau\dot{\boldsymbol{\eta}} = \alpha_z\left(\beta_z \mathrm{vec}\left(\mathbf{g}_o * \overline{\mathbf{q}}\right) - \boldsymbol{\eta}\right) + \mathbf{f}_o(x). \tag{3.71}$$

While this is equivalent as far as the direction of change is concerned, such a formulation does not fully take into account the geometry of SO(3). By considering Eq. (1.42), we can see that only the logarithmic map multiplied by 2 can provide proper mapping of the quaternion product $\mathbf{g}_o * \overline{\mathbf{q}}$ onto the angular velocity. Another difference is that there is no scaling factor $\mathbf{D}_o$ in [26]. See Section 3.3.4 for the analysis of differences between the two approaches.

For the integration of (3.61), (3.62), and (3.6), we can use the formulae

$$\boldsymbol{\eta}(t+\Delta t) = \boldsymbol{\eta}(t) + \frac{1}{\tau}\left(\alpha_z\left(\beta_z 2\log\left(\mathbf{g}_o * \overline{\mathbf{q}(t)}\right) - \boldsymbol{\eta}(t)\right) + \mathbf{f}_o(x(t))\right)\Delta t, \tag{3.72}$$

$$\mathbf{q}(t+\Delta t) = \exp\left(\frac{\Delta t}{2}\frac{\boldsymbol{\eta}(t)}{\tau}\right) * \mathbf{q}(t), \tag{3.73}$$

$$x(t+\Delta t) = x(t) - \frac{\alpha_x}{\tau}x(t)\Delta t, \tag{3.74}$$

where the quaternion exponential is defined as in Eq. (1.39), i.e. $\exp(\theta\mathbf{n}) = \cos(\theta) + \sin(\theta)\mathbf{n}$ and quaternion logarithm as in Eq. (1.41). $\Delta t > 0$ is the integration interval.

3.3.2 Position trajectories

For the sake of completeness we also rewrite DMP equations (3.10) – (3.11), which are used to encode the positional part of the trajectory, in variable $\mathbf{p}$

$$\tau\dot{\mathbf{z}} = \alpha_z(\beta_z(\mathbf{g}_p - \mathbf{p}) - \mathbf{z}) + \mathbf{f}_p(x), \tag{3.75}$$

$$\tau\dot{\mathbf{p}} = \mathbf{z}, \tag{3.76}$$

where $\mathbf{g}_p \in \mathbb{R}^3$ denotes the final position on the recorded trajectory. The forcing term $\mathbf{f}_p$ is defined as

$$\mathbf{f}_p(x) = \mathbf{D}_p \frac{\sum_{i=1}^N \mathbf{w}_i^p \Psi_i(x)}{\sum_{i=1}^N \Psi_i(x)} x, \tag{3.77}$$

where $\mathbf{D}_p = \mathrm{diag}\,(\mathbf{g}_p - \mathbf{p}_0) \in \mathbb{R}^{3\times 3}$. As mentioned above, the diagonal matrices $\mathbf{D}_p$ in (3.77) and $\mathbf{D}_o$ in (3.64) are used to scale the movement amplitude if the goal configuration $\mathbf{g}_p$ and/or $\mathbf{g}_o$ change. To track the desired Cartesian space trajectories, we need to integrate (3.75) – (3.76) and (3.79) – (3.80) along with the common phase (3.6).

Given the robot tool center point trajectory $\{\mathbf{p}_j,\, \dot{\mathbf{p}}_j,\, \ddot{\mathbf{p}}_j,\, t_j\}_{j=0}^T$, the free parameters can be calculated in a similar way as in (3.82), i. e. by first rewriting (3.75) – (3.76) as one second order differential equation [12] and then solving the resulting system of linear equations

$$\frac{\sum_{i=1}^N \mathbf{w}_i^p \Psi_i(x(t_j))}{\sum_{i=1}^N \Psi_i(x(t_j))} x(t_j) = \mathbf{D}_p^{-1}\left(\tau^2\,\ddot{\mathbf{p}}_d(t_j) + \alpha_z\tau\dot{\mathbf{p}}_d(t_j) - \alpha_z\beta_z\left(\mathbf{g}_p - \mathbf{p}_d(t_j)\right)\right), \tag{3.78}$$

where the phases $x(t_j)$ are calculated as in (3.83).

3.3.3 Rotation matrix based DMPs

Rotation matrices offer an alternative, singularity-free representation of SO(3). We can easily rewrite Eq. (3.61) – (3.62) in rotation matrix form

$$\tau\dot{\boldsymbol{\eta}} = \alpha_z(\beta_z \log(\mathbf{R}_g\mathbf{R}^{\mathrm{T}}) - \boldsymbol{\eta}) + \mathbf{f}_o(x). \tag{3.79}$$

$$\tau\dot{\mathbf{R}} = [\boldsymbol{\eta}]_\times \mathbf{R}, \tag{3.80}$$

where $\mathbf{R}_g$ denotes the goal orientation and all other parameters are just like in (3.61) – (3.62). The matrix logarithm $\log(\mathbf{R}_g\mathbf{R}^{\mathrm{T}})$ has the same role as $2\log(\mathbf{g}_o * \overline{\mathbf{q}})$ in Eq. (3.61), as shown by Eq. (1.37) in Section 1.2.3. Equation (3.80) is motivated by (1.8). Again we have $\boldsymbol{\eta} = \tau\boldsymbol{\omega}$, as can be easily deduced from (3.80) and (1.8). Thus $\boldsymbol{\eta}$ is the scaled angular velocity. The nonlinear forcing term is defined as in (3.64) and contains free parameters $\mathbf{w}_i^o \in \mathbb{R}^3$, which need to be determined to follow any given rotation matrix trajectory.

The data for computing them is usually given as

$$\{\mathbf{R}_d(t_j), \boldsymbol{\omega}_d(t_j), \dot{\boldsymbol{\omega}}_d(t_j), t_j\}_{j=0}^{T}, \tag{3.81}$$

where ,$\boldsymbol{\omega}_d(t_j)$, $\dot{\boldsymbol{\omega}}_d(t_j)$ are computed by numerical differentiation in a similar way as in the case of quaternion DMPs in Sec. 3.3.1. The parameters $\mathbf{w}_i^o$ are calculated by solving the following system of linear equations, which we obtain from (3.79)

$$\frac{\sum_{i=1}^{N} \mathbf{w}_i^o \Psi_i(x(t_j))}{\sum_{i=1}^{N} \Psi_i(x(t_j))} x(t_j) = \mathbf{D}_o^{-1} \left(\tau^2 \dot{\boldsymbol{\omega}}_d(t_j) + \alpha_z \tau \boldsymbol{\omega}_d(t_j) - \alpha_z \beta_z \log\left(\mathbf{R}_g \mathbf{R}_d(t_j)^{\mathrm{T}}\right)\right), \tag{3.82}$$

where phases $x(t_j)$ are calculated by integrating (3.6), i.e.

$$x(t_j) = \exp\left(-\frac{\alpha_x}{\tau} t_j\right) \tag{3.83}$$

and $j = 0, \ldots, T$. The scaling factor $\mathbf{D}_o = \operatorname{diag}\left(\log\left(\mathbf{R}_g \mathbf{R}_0^{\mathrm{T}}\right)\right) \in \mathbb{R}^{3\times 3}$ is a diagonal matrix.

Standard Euler formulas can be applied to integrate (3.79). To integrate (3.80) we use the expression

$$\mathbf{R}(t + \Delta t) = \exp\left(\Delta t \frac{[\boldsymbol{\eta}]_\times}{\tau}\right) \mathbf{R}(t), \tag{3.84}$$

which is guaranteed to generate a rotation matrix because the matrix exponential map $\exp\left(\Delta t [\boldsymbol{\eta}]_\times / \tau\right)$ results in a rotation matrix and the product of two rotation matrices is a rotation matrix. Note that the matrix exponential is defined in a similar way as the quaternion exponential (1.40), i.e. as a rotation matrix corresponding to the rotation by angle θ about rotation axis $\mathbf{n}$.

3.3.4 Comparison of DMPs defined on SO(3)

Figure 3.3 and 3.4 show that quaternion-based DMPs, which use the term $2\log\left(\mathbf{g}_o * \overline{\mathbf{q}}\right)$ (3.61), and rotation matrix-based DMPs generate better responses in terms of angular velocities than quaternion difference vector $\operatorname{vec}\left(\mathbf{g}_o * \overline{\mathbf{q}}\right)$ DMP as in (3.71). Both quaternion- and rotation-matrix-based DMPs converge to the attractor point much faster than the approach proposed in [26], which is the desired characteristics of the linear part of the DMP system that should ensure convergence to the attractor point. Part of the reason for this is the lack of multiplication by 2 in (3.71). If we modify (3.71) to

$$\tau \dot{\boldsymbol{\eta}} = \alpha_z \left(\beta_z 2\operatorname{vec}\left(\mathbf{g}_o * \overline{\mathbf{q}}\right) - \boldsymbol{\eta}\right) + \mathbf{f}_o(x), \tag{3.85}$$

the DMP system (3.61) – (3.62) still generates a significantly quicker response than (3.85), (3.62), and converges faster, but the difference is smaller, as we can see by comparing Fig. 3.3 and 3.6. Nevertheless, the response of (3.61) – (3.62) is clearly preferable. Fig. 3.3 and 3.4 show that as expected, the quaternion based DMP (3.61) – (3.62) and the rotation matrix based DMP (3.79) – (3.80) generate exactly the same response in terms of angular

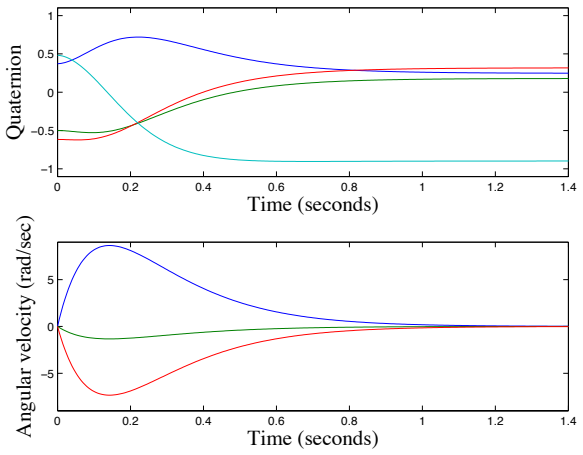

Figure 3.3: Response of the quaternion-based DMP system (3.61) – (3.62) without nonlinear term $\mathbf{f}_o$. The upper graph shows the time trajectories of the four unit quaternion components, and the lower graph the time trajectories of the three components of the angular velocity.

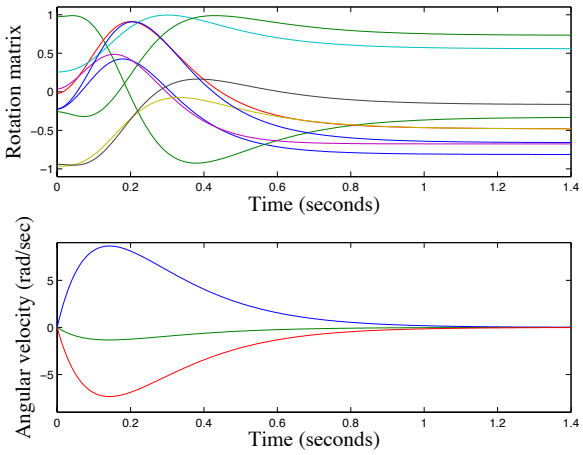

Figure 3.4: Response of the rotation matrix based DMP system (3.79) – (3.80) without nonlinear term $\mathbf{f}_o$. The upper graph shows the time trajectories of the nine rotation matrix components, and the lower graph the time trajectories of the three components of the angular velocity.

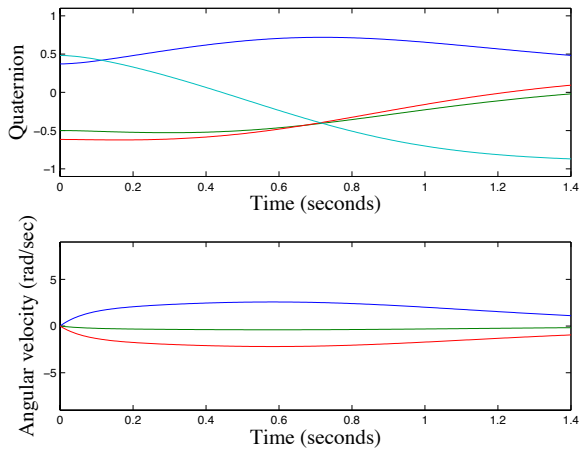

Figure 3.5: Response of the DMP system (3.71), (3.62) with quaternion difference instead of the logarithmic map (as proposed in [26]) and without nonlinear term $\mathbf{f}_o$.

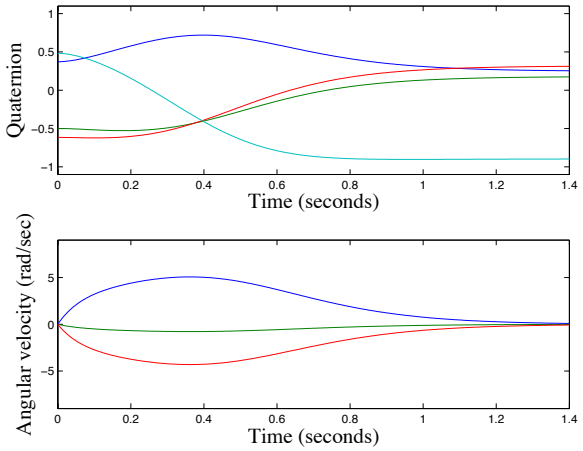

Figure 3.6: Response of the DMP system (3.85), (3.62) with double quaternion difference instead of the logarithmic map and without nonlinear term $\mathbf{f}_o$.

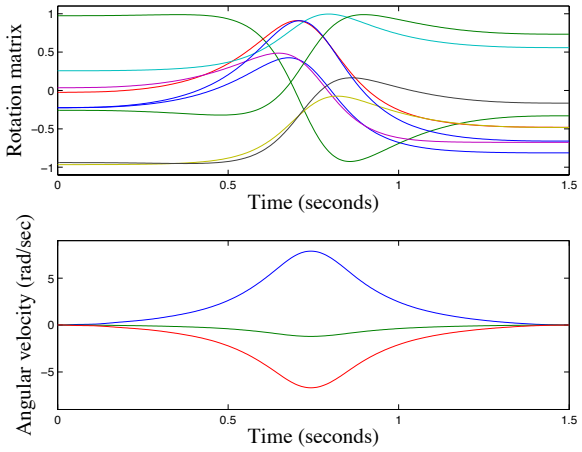

Figure 3.7: Reproduction of the desired minimum jerk polynomial with the rotation matrix based DMP system (3.79) – (3.80).

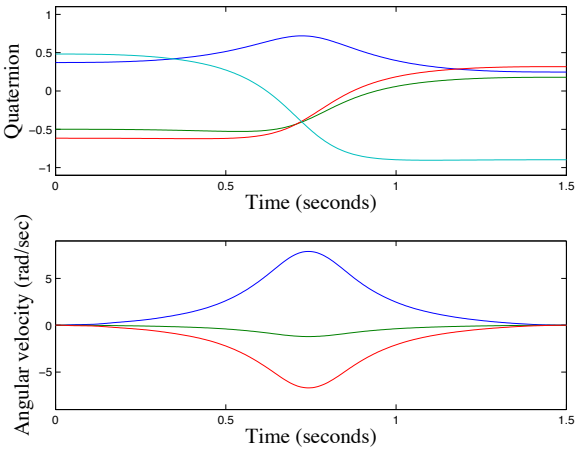

Figure 3.8: Reproduction of the desired minimum jerk polynomial with the quaternion based DMP system (3.61) – (3.62).

velocity. The starting orientation was set to $\mathbf{q}_0 = 0.3717 + [-0.4993, -0.6162, 0.4825]^{\mathrm{T}}$ and the goal orientation to $\mathbf{g}_o = 0.2471 + [0.1797, 0.3182, -0.8974]^{\mathrm{T}}$. The other constants were specified as follows: $\alpha_x = 2, \alpha_z = 48, \beta_z = 12, \tau = 3.5$.

We have thus seen theoretically and experimentally that the proposed systems (3.79) – (3.80) and (3.61) – (3.62) are significantly better than (3.85), (3.62), even though the multiplication by 2 has been added compared to the system (3.71), (3.62) from [26]. Note that 2 is the correct multiplier to obtain a critically damped system in (3.61) – (3.62). If we further increase the multiplier, the resulting system is not critically damped but starts oscillating towards the goal orientation, which is suboptimal for robot control.

Next, we evaluate the performance of orientation trajectory approximation using both quaternion- and rotation-matrix-based DMPs. For this experimental evaluation, the same parameters as above were used, except for τ, which was set to 1.5. To generate an example orientation trajectory, we sampled a minimum jerk polynomial between the start quaternion $\mathbf{q}_0$ and the goal quaternion $\mathbf{g}_o$, and then normalized the resulting quaternion trajectory. The results shown in Fig. 3.7 and 3.8 demonstrate that both approaches can accurately reproduce the desired trajectories. However, this experiment also revealed a subtle issue that should be considered when using rotation-matrix-based DMPs. Specifically, the rotation matrix logarithm, as defined in (1.37), exhibits a discontinuity at $|\log(\mathbf{R})| = \pi$. This occurs because, when an object is rotated by 180 degrees, the final orientation is the same regardless of the rotation direction. This discontinuity can cause problems when estimating DMPs through the system of equations (3.82), as $\mathbf{f}_o$ can become discontinuous when this boundary is crossed, even if the movement itself is smooth. While this issue can be resolved by adding appropriate constants to ensure the continuity of $\mathbf{f}_o$ in (3.79), such an approach requires the user to implement a cumbersome bookkeeping procedure. On the other hand, quaternion-based DMPs do not encounter this problem because there is no discontinuity boundary in the quaternion logarithm (1.41) on the unit

sphere S^3. The only singularity occurs at $\mathbf{q} = -1 + [0, 0, 0]^T$, which is rarely encountered in practice and can be easily handled (for example, by assuming the previous direction vector in the logarithm when $-1 + [0, 0, 0]^T$ is reached on the trajectory). For this reason, we prefer to use quaternion representation to avoid the tedious bookkeeping associated with rotation-matrix-based DMPs.

3.4 Third-order DMPs and Sequencing

High performance robot control systems, which can take into account robot dynamics, utilize the desired positions, velocities, and accelerations to generate motor commands. The second order system (3.8) – (3.9) ensures only continuous positions and velocities if the goal g suddenly changes during robot motion. This problem can be alleviated by instead using a third-order system [32]

$$\tau \dot{z} = \alpha_z(\beta_z(r - y) - z) + f(x), \tag{3.86}$$
$$\tau \dot{y} = z, \tag{3.87}$$
$$\tau \dot{r} = \alpha_g(g - r), \tag{3.88}$$

The phase x is again given by Eq. (3.6). It is easy to see that the general solution of the linear part of equation system (3.86) – (3.88), i. e. the above expression without $f(x)$, is given by

$$\begin{bmatrix} z(t) \\ y(t) \\ r(t) \end{bmatrix} = \begin{bmatrix} 0 \\ g \\ g \end{bmatrix} + \exp(\mathbf{A}t)\,\mathbf{c}, \quad \mathbf{A} = \frac{1}{\tau}\begin{bmatrix} -\alpha_z & -\alpha_z\beta_z & \alpha_z\beta_z \\ 1 & 0 & 0 \\ 0 & 0 & -\alpha_g \end{bmatrix}. \tag{3.89}$$

Matrix $\mathbf{A}$ has eigenvalues at $\lambda_1 = -\alpha_g/\tau$, $\lambda_{2,3} = \left(-\alpha_z \pm \sqrt{\alpha_z^2 - 4\alpha_z\beta_z}\right)/(2\tau)$, which all have negative real parts if τ, α_z, β_z, $\alpha_g > 0$. Thus for such parameters the system is guaranteed to converge to its unique attractor point $[0, g, g]^T$. Just like before, the system is critically damped if $\alpha_z = 4\beta_z$, in which case $\lambda_{2,3} = -\alpha_z/(2\tau)$. Also like before, since $f(x)$ vanishes as $t \to \infty$, the nonlinear system (3.86) – (3.88) is guaranteed to converge to this same attractor point.

Note that by setting $r(t) = g$, the third-order system (3.86) – (3.88) becomes equivalent to the second order system (3.8) – (3.9). Therefore we can (and should) use the original second-order system when learning DMP parameters from a single demonstration. The third-order system should only be used at execution time. If the final destination g suddenly changes, the third order system ensures that r gradually transitions to the new goal position and the robot acceleration, which is defined by Eq. (3.86) and $\ddot{y} = \dot{z}/\tau$, remains continuous.

The third-order system (3.86) – (3.88) is also useful to continuously transition from

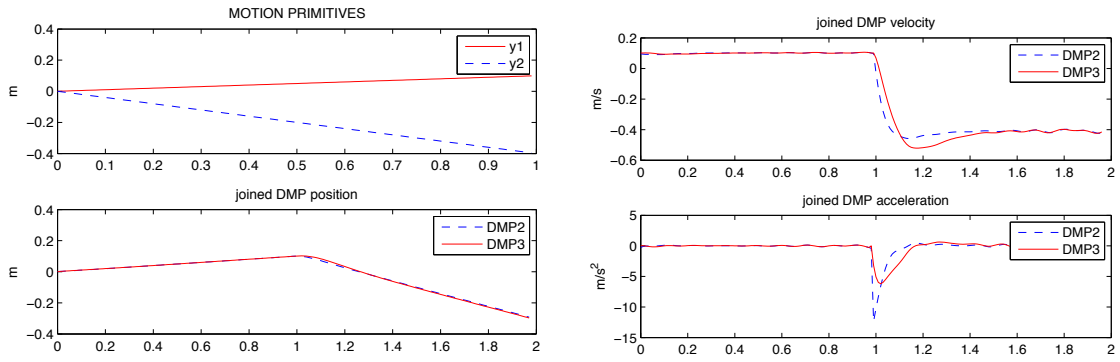

Figure 3.9: Sequencing of movement primitives. The application of third order DMPs (red full line) prevents the jump in acceleration, which can be observed when using second-order DMPs (blue dashed line).

one DMP to another. From (3.86) and (3.87) we obtain

$$r = \frac{\tau^2 \ddot{y} + \tau \alpha_z \dot{y} + \alpha_z \beta_z y - f(x)}{\alpha_z \beta_z}. \tag{3.90}$$

Thus if the robot position, velocity, and acceleration at transition time are given by y_c, $\dot{y}_c$, and $\ddot{y}_c$, the next DMP should be initialized to

$$y(0) = y_c, \quad z(0) = \tau \dot{y}_c, \quad r(0) = \frac{\tau^2 \ddot{y}_c + \tau \alpha_z \dot{y}_c + \alpha_z \beta_z y_c - f(1)}{\alpha_z \beta_z}, \quad x(0) = 1. \tag{3.91}$$

In this way the positions, velocities, and accelerations remain continuous as the robot transitions from one DMP to another.

3.4.1 Third-order DMPs in Cartesian space

For quaternion based DMPs, we can define the following third-order system

$$\tau \dot{\boldsymbol{\eta}} = \alpha_z \left(\beta_z 2 \log \left(\mathbf{r} * \overline{\mathbf{q}} \right) - \boldsymbol{\eta} \right) + \mathbf{f}_o(x), \tag{3.92}$$

$$\tau \dot{\mathbf{q}} = \frac{1}{2} \boldsymbol{\eta} * \mathbf{q}, \tag{3.93}$$

$$\tau \dot{\mathbf{r}} = \alpha_r 2 \log(\mathbf{g}_o * \overline{\mathbf{r}}) * \mathbf{r}. \tag{3.94}$$

Equivalently to r in (3.86), $\mathbf{r}$ in (3.92) is now a variable that should continuously transitions to the goal $\mathbf{g}_o$, even if it suddenly changes. Equation (3.94) should be integrated together with (3.61) – (3.62) using formula (3.73).

3.5 Modulation of DMPs

Modulation of DMPs is a powerful technique that enhances the flexibility and adaptability of robotic motion. By modulating the parameters of the DMPs, such as the goal position,

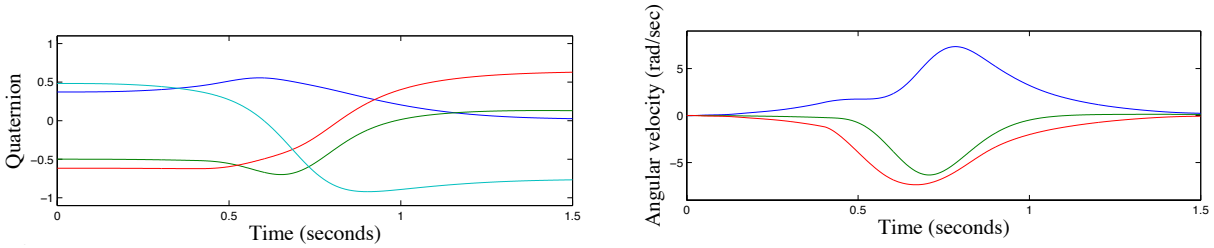

Figure 3.10: Response of the DMP system (3.61), (3.62), (3.94) to the switching of goal orientation during the movement. The goal of the movement was changed at 0.4 seconds. The change of goal orientation corresponded to a rotation of 30.4 degrees. The original movement is shown in Fig. 3.8.

trajectory shape, or even the speed of movement, robots can dynamically adjust their actions in response to changes in the environment or the task at hand. This capability is crucial for tasks that require adaptation to unforeseen obstacles, variations in object placement, or changes in task requirements. For instance, modulation can be used to scale a movement trajectory to reach a new goal location, modify the trajectory to avoid an obstacle, or adjust the speed of execution to meet timing constraints. This adaptability makes DMPs particularly useful in real-world applications where conditions can change rapidly and unpredictably, allowing robots to perform complex tasks with a level of autonomy and robustness that would be difficult to achieve with static, pre-programmed movements.

3.5.1 Phase stopping

Normally, Eq. (3.6) is not analytically integrated to (3.7). Instead, at execution time (3.6) and (3.9) are modified to

$$\tau \dot{x} = -\frac{\alpha_x x}{1 + \alpha_{px}|\tilde{y} - y|}, \tag{3.95}$$

$$\tau \dot{y} = z + \alpha_{py}(\tilde{y} - y), \tag{3.96}$$

where y and $\tilde{y}$ respectively denote the desired and the actual robot position. Note that if the robot cannot follow the desired motion, $\alpha_{px}|\tilde{y} - y|$ becomes large, which in turn makes the phase change $\dot{x}$ small. Thus the phase evolution is stopped until the robot catches up with the desired configuration y (see Fig. 3.97). On the other hand, if the robot follows the desired movement precisely, Eq. (3.95) and (3.96) are no different from (3.6) and (3.9), respectively.

We can also define this behavior for Cartesian-space DMPs. In this case the original equation for phase (3.6) is replaced with

$$\tau \dot{x} = -\frac{\alpha_x x}{1 + \alpha_{px}\left(\|\tilde{\mathbf{p}} - \mathbf{p}\| + \gamma\, \mathrm{d}(\tilde{\mathbf{q}}, \mathbf{q})\right)}, \tag{3.97}$$

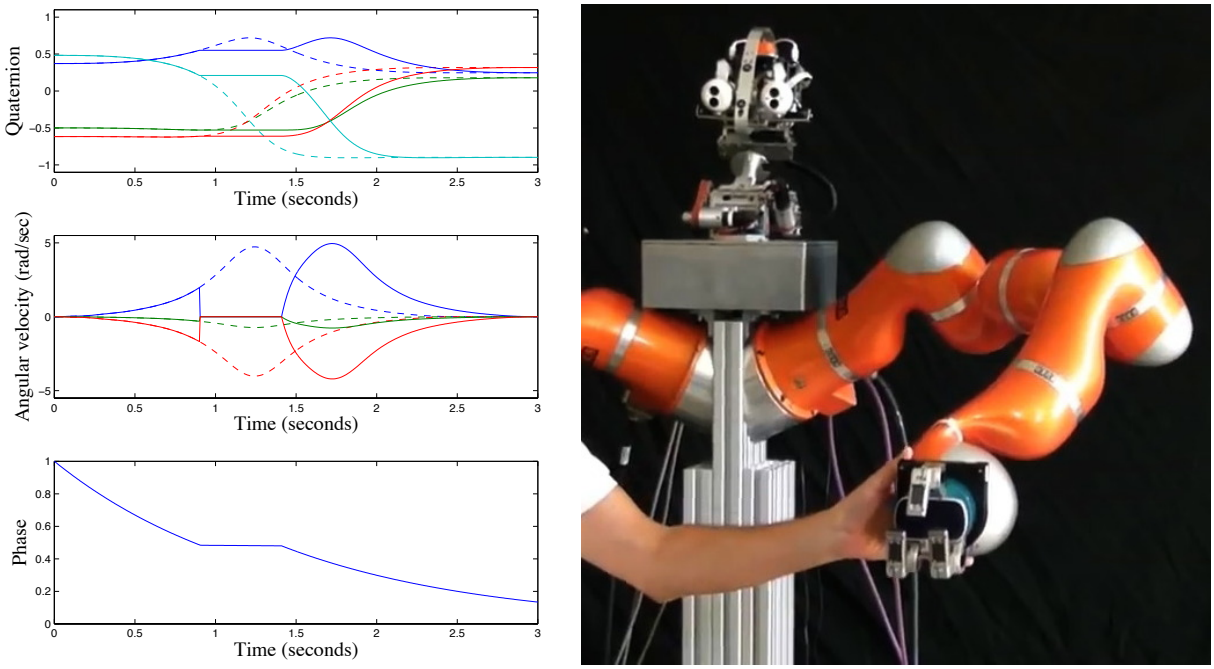

Figure 3.11: The effects of phase stopping on quaternion based DMPs defined by (3.97) and (3.99). The original, unperturbed trajectory is indicated with the dashed lines, and the perturbed movement, which was stopped between 0.9 and 1.4 seconds, with full lines. Note that the robot can smoothly resume its movement once the perturbation has been removed.

where $\|\tilde{\mathbf{p}} - \mathbf{p}\| + \gamma\, \mathrm{d}(\tilde{\mathbf{q}}, \mathbf{q})$ is the trajectory tracking error, $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ are the actual position and orientation of the tool center point, respectively, and $\mathbf{p}$ and $\mathbf{q}$ the corresponding DMP control outputs. Note that in the case of large tracking errors, the error value $\|\tilde{\mathbf{p}} - \mathbf{p}\| + \gamma\, \mathrm{d}(\tilde{\mathbf{q}}, \mathbf{q})$ becomes large which in turn makes the phase change $\dot{x}$ small. Thus the phase evolution is stopped until the robot reduces the tracking error.

In the context of Cartesian space DMPs, Eq. (3.96) becomes different for the positional and orientational part of the trajectory, which are respectively encoded by Eq. (3.76) and (3.62). We obtain

$$\tau \dot{\mathbf{p}} = \mathbf{z} + \alpha_{pp}(\tilde{\mathbf{p}} - \mathbf{p}), \tag{3.98}$$

$$\tau \dot{\mathbf{q}} = \frac{1}{2}\left(\boldsymbol{\eta} + \alpha_{pr} 2 \log\left(\tilde{\mathbf{q}} * \overline{\mathbf{q}}\right)\right) * \mathbf{q}, \tag{3.99}$$

Eq. (3.99) is integrated using a formula analog to (3.73).

3.5.2 Robustness to perturbations through coupling of DMPs

Another advantage of DMPs is that they are robust against perturbations [12], e. g. when a robot is physically pushed away from the desired motion trajectory while moving. The standard approach to perturbations is – in case of Cartesian space DMPs – to simply continue integrating Eq. (3.6), (3.75) – (3.76), (3.61) – (3.62), (3.6) after the perturbation

has arisen. This way the robot gradually returns to the desired trajectory. However, this approach does not ensure that the complete movement is reproduced, just that the robot returns to the original movement. The phase stopping mechanism can be used if it is necessary to execute the complete movement. For this purpose, we enhance Eq. (3.97), (3.98), (3.99) as follows

$$\tau\dot{x} = -\frac{\alpha_x x}{1+\alpha_{px}\varepsilon(\tilde{\mathbf{p}},\mathbf{p}_d,\mathbf{p},\tilde{\mathbf{q}},\mathbf{q}_d,\mathbf{q})}, \tag{3.100}$$

$$\tau\dot{\mathbf{p}} = \mathbf{z}+\alpha_{pp}(\tilde{\mathbf{p}}+\mathbf{p}_d-2\mathbf{p}), \tag{3.101}$$

$$\tau\dot{\mathbf{q}} = \frac{1}{2}\left(\boldsymbol{\eta}+\alpha_{pr}2\left(\log\left(\tilde{\mathbf{q}}*\overline{\mathbf{q}}\right)+\log\left(\mathbf{q}_d*\overline{\mathbf{q}}\right)\right)\right)*\mathbf{q}, \tag{3.102}$$

where

$$\varepsilon = \|\tilde{\mathbf{p}}-\mathbf{p}\|+\|\mathbf{p}_d-\mathbf{p}\|+\gamma\left(\mathrm{d}(\tilde{\mathbf{q}},\mathbf{q})+\mathrm{d}(\mathbf{q}_d,\mathbf{q})\right),$$

$\mathbf{p}_d$ and $\mathbf{q}_d$ are the desired position and orientation, respectively, and all other variables and constants are as in (3.97), (3.98), (3.99). The unit quaternion distance metric d defined in (1.43) can be used here. The complete approach to control the robot is specified by the following coupled DMP system

- A DMP system constructed from (3.61), (3.75), (3.101), (3.102), and (3.6). Its outputs are used to generate motor commands.
- A separate DMP that uses standard DMP equations (3.61) – (3.62) and (3.75) – (3.76) with the same constants and variables as the above system. This DMP generates the desired position and orientation for (3.101) – (3.102).
- The coupling of both DMP systems occurs through the phase equation (3.100), which is used by both.

The described coupled system halts the phase evolution whenever the robot controller is unable to track the commanded position and orientation or whenever the commanded position and orientation deviate from the desired values. The second term in (3.101), (3.102) ensures that the system can recover once the perturbation is removed. After recovery the robot continues to perform the movement at the spot on the trajectory where it was before the onset of perturbation. Note that the original DMP system is equivalent to the one proposed in this section if the robot tracks the desired trajectory perfectly and there are no unexpected perturbations.

3.5.3 Obstacle avoidance

To realize obstacle avoidance in Cartesian space, the trajectory should be planned in this space as well. Let's assume that we have a three degree-of-freedom discrete DMP that models point-to-point reaching in Cartesian space. We denote the 3-D position vector of

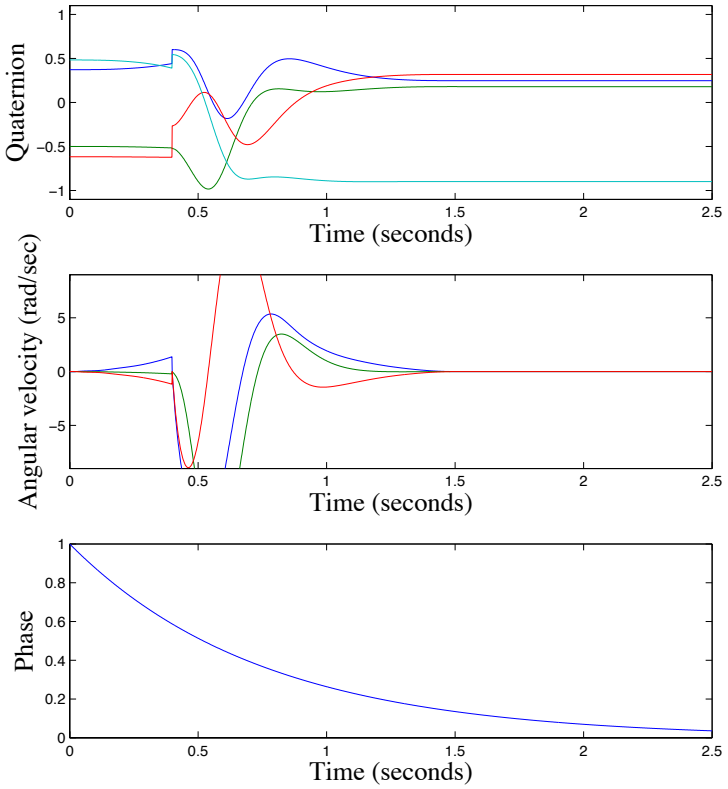

Figure 3.12: Response of the DMP system (3.61), (3.62) to a perturbation. Here the orientation trajectory was perturbed by rotation of 28.3 degrees. The angular velocity was simultaneously set to 0. The graph shows the convergence of the perturbed movement to the final goal orientation.

the 3 DOF discrete dynamical system by $\mathbf{p} = [p_x, p_2, p_x]^T$. The objective is to generate a reaching movement to the goal state $\mathbf{g} = [g_x, g_y, g_x]^T$ without hitting obstacles . On the way to the goal state, an obstacle is positioned at $\mathbf{o} = [o_x, o_y, o_y]^T$. The goal is to avoid this obstacle. A suitable coupling term $\mathbf{C}_t = [C_{t,x}, C_{t,y}, C_{t,z}]^T$ for obstacle avoidance can be formulated as follows:

$$\mathbf{C}_t(\mathbf{y}, \dot{\mathbf{y}}) = \gamma \, \mathrm{sig}\left(\|\mathbf{o} - \mathbf{y}\|\right) \mathbf{R}\dot{\mathbf{y}} \left(\pi - \varphi\right) \exp\left(-\beta\varphi\right), \tag{3.103}$$

where

$$\varphi = \arccos\left(\frac{(\mathbf{o} - \mathbf{y})^T \dot{\mathbf{y}}}{\|\mathbf{o} - \mathbf{y}\| \|\dot{\mathbf{y}}\|}\right), \tag{3.104}$$

$$\mathrm{sig}(x) = \frac{1}{1 + e^{\eta(x-d)}}, \tag{3.105}$$

$$\mathbf{R} = \exp\left(\left(\frac{\pi}{2} - \varphi\right)\mathbf{n}\right), \tag{3.106}$$

$$\mathbf{n} = \frac{(\mathbf{o} - \mathbf{y}) \times \dot{\mathbf{y}}}{\|\mathbf{o} - \mathbf{y}\| \|\dot{\mathbf{y}}\|}. \tag{3.107}$$

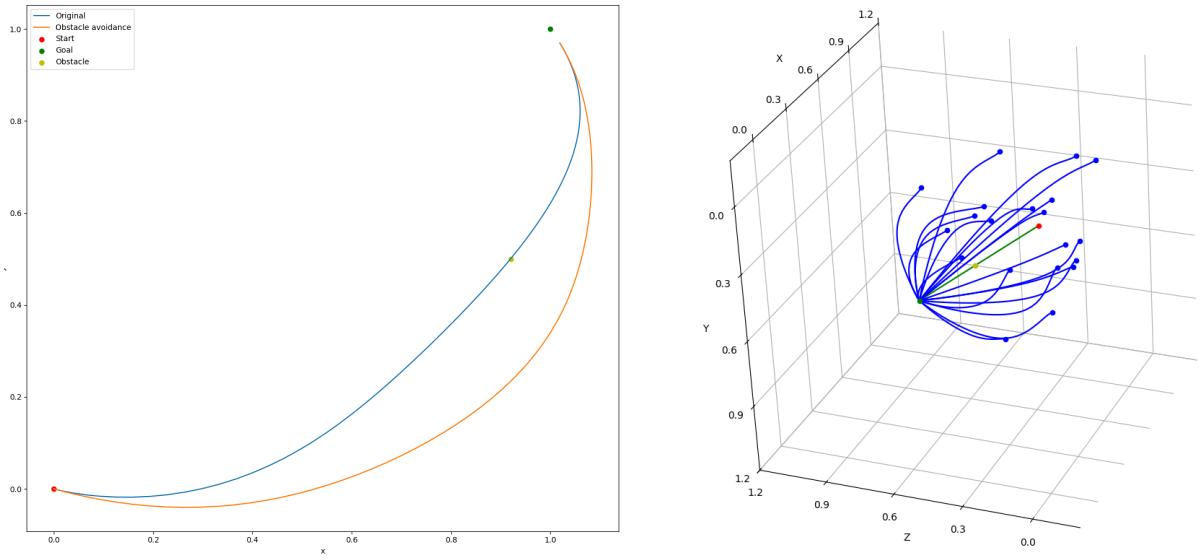

Figure 3.13: **Obstacle avoidance**: In both graphs, the obstacle is represented by a yellow sphere located on the trajectory that starts at a red circle. The left graph shows two scenarios: one where the DMP trajectory passes through the point obstacle when there is no coupling term for obstacle avoidance, and another where a 2D DMP successfully avoids the obstacle using a coupling term. The right graph illustrates DMP paths in 3D. Without a coupling term for obstacle avoidance, the DMP trajectory passes through the point obstacle. However, when the same DMP starts from multiple random positions with the coupling term activated, all resulting paths successfully avoid the obstacle, demonstrating the effectiveness of the obstacle avoidance mechanism.

γ, β, and η are the scaling factors and d is the distance at which the obstacle should start affecting the robot motion. The above coupling term generates a velocity component that lies in a plane defined by vectors $\mathbf{o}-\mathbf{y}$ and $\dot{\mathbf{y}}$. It is orthogonal to the line $\mathbf{o}-\mathbf{y}$ connecting the robot tip and the obstacle.

We can ensure that the robot tip avoids the obstacle by adding the coupling term $\mathbf{C}_t$ to Eq. (3.9)

$$\tau\dot{\mathbf{z}} = \alpha_z(\beta_z(\mathbf{g}-\mathbf{y}) - \mathbf{z}) + \mathbf{f}(x) + \mathbf{C}_t(\mathbf{y}, \mathbf{z}/\tau). \tag{3.108}$$

The resulting behavior is shown in Fig. 3.13. Note that in this way we can only ensure that the robot tip avoids the obstacle. However, the rest of the robot could still collide with it.

3.6 Learning of DMPs from Multiple Demonstrations

Since DMPs have not been designed to represent whole classes of movements, a standard DMP typically reproduces a specific movement, which can be modulated like described in Section 3.5. However, it is not possible to encode multiple trajectories within one DMP. It turns out, however, that by computing local task models, it is possible to generate a specific DMP that is adapted to the current configuration of the external world [35]. This

approach has the advantage that the generalized DMPs are computed by local regression methods, which apply local, linear optimization as compared to some other approaches, which apply global, nonlinear optimization methods to compute more complex motion representations [9].

3.6.1 Action generalization using dynamic movement primitives and local weighting

Next we describe a methodology for generalizing multiple example trajectories to new, previously unobserved situations. This approach differs from the scenario in Section 3.2, where only a single example trajectory is available. In this context, we assume that each example trajectory is associated with parameters that describe the key characteristics of the task, typically its goal. These parameters act as *query points* into an example database.

To illustrate this, consider the problem of throwing a ball towards a specific target. Suppose we have a set of example robot trajectories that resulted in successful ball throws toward various targets. The goal of generalization is to generate a trajectory that will accurately direct the ball toward any given target in space. In this case, the parameters defining the goal of the task are the target positions.

We begin with a set of example trajectories, each accompanied by parameters that characterize the task:

$$\mathcal{Z} = \{\mathbf{y}_{\mathrm{d}}^i(t_{i,j}), \dot{\mathbf{y}}_{\mathrm{d}}^i(t_{i,j}), \ddot{\mathbf{y}}_{\mathrm{d}}^i(t_{i,j}); \mathbf{q}_i | \; i = 1, \ldots, M, \; j = 0, \ldots, T_i\}, \tag{3.109}$$

where $\mathbf{y}_d^i(t_{i,j})$, $\dot{\mathbf{y}}_d^i(t_{i,j})$, $\ddot{\mathbf{y}}_d^i(t_{i,j})$ represent the measured positions, velocities, and accelerations along the i-th trajectory, M is the number of example trajectories, and $T_i + 1$ is the number of sampling points on each trajectory. The parameters $\mathbf{q}_i \in \mathbb{R}^m$ describe the task in each example scenario and are typically related to the goal of the action. They serve as query points into a database of example trajectories. This type of data can be obtained by any of the data collection methodologies described in Section 1.4.

To explain the concept of query points, let's consider tasks such as reaching, ball throwing, and drumming. In the case of reaching movements, the goal of an action (or query point) is described by the final reaching destination in the Cartesian space, whereas the trajectories and consequently the attractor points of the DMPs can be defined in joint space. Thus the query and attractor points are connected through the robot kinematics. In other cases, the queries are less directly associated with the parameters of the DMP. For example, in the case of ball throwing, the goal is characterized by the position of the basket into which the ball should be thrown. This position is not directly encoded in the parameters of the discrete DMP as given by Eqs. (3.10) – (3.11). As example of periodic movements, we consider drumming, where the height at which the drum is mounted can vary. In this case the robot needs to adapt its drumming motion to the varying height

of the drum, which is used as query point not directly encoded in the periodic DMP Eqs. (3.15) – (3.16). In summary, the query points normally originate from an intuitive characterization of the task, but the functional relationship between them and the DMP parameters may not be straightforward.

Thus the challenge is to generate new DMPs that specify optimal movements for any new query point $\mathbf{q}$, which typically does not coincide with one of the example queries $\mathbf{q}_k$. As explained in Section 3.1, DMPs are generally defined by the parameters $\mathbf{w}$, τ (or r and $\Omega = 1/\tau$ in the case of periodic movements), and g. Therefore, it is necessary to learn a function that maps query points to these parameters

$$\mathbf{G}\left(\mathcal{Z}\right) : \mathbf{q} \longmapsto [[\mathbf{w}_1^{\mathrm{T}}, \ldots, \mathbf{w}_n^{\mathrm{T}}], \mathbf{g}^{\mathrm{T}}, \tau]^T \tag{3.110}$$

or

$$\mathbf{G}\left(\mathcal{Z}\right) : \mathbf{q} \longmapsto [[\mathbf{w}_1^{\mathrm{T}}, \ldots, \mathbf{w}_n^{\mathrm{T}}], \mathbf{g}^{\mathrm{T}}, r, \Omega]^T \tag{3.111}$$

Examples $\mathcal{Z}$ specified in (3.109) are the data that can be used to learn this function.

Note that $\mathbf{G}(\mathcal{Z})$ becomes a function only by constraining the solution trajectory to be as similar as possible to the example trajectories. For example, there are many different ways of how to throw a ball into a basket at a certain location. The relationship between the basket positions (query points) and DMP parameters as given in Eq. (3.110) becomes a function only by requesting that the generalized throwing movements are similar to the example throws. This similarity criterion should in practice be embedded into the optimization process.

In general, the functional relationship between $\mathbf{q}$ and DMP parameters is unknown. It is often not feasible to identify a global model that provides a good approximation for the function $\mathbf{G}(\mathcal{Z})$. To avoid global model identification, *weighted averaging* and *Locally Weighted Regression* (LWR) can be applied. These methods are motivated by the observation that some data points are more relevant than the others. Control policies that solve the task in situations that are very different from the current one do not carry much information about the optimal policy in the current situation. LWR is a regression method that fits local models to nearby data [1] and has a lower computational complexity than many other nonparametric regression methods such as Gaussian process regression (GPR) [23]. With LWR there is no global optimization step; instead, training data is stored and utilized when local models need to be computed. Such approaches are also denoted as lazy learning because processing of training data is deferred until a query needs to be answered.

To fully specify a DMP that defines the motion at a new query point, we need to estimate the shape parameters $\mathbf{w}$, goal $\mathbf{g}$, and time constant τ in case of discrete movements and the shape parameters $\mathbf{w}$, center point $\mathbf{g}$, amplitude r and frequency of motion Ω in case of periodic movements. In Section 3.2 we have already explained the generation of DMPs using only one training trajectory. Here we consider the estimation of these parameters from multiple demonstrations. The rest of the DMP parameters (α_z, β_z, and

α_x) remain fixed and are determined so that the convergence of the underlying dynamic system is guaranteed.

Generalization of goal, timing, amplitude, and frequency

The goal parameter $\mathbf{g}$, time constant τ, amplitude r, and frequency Ω are either directly measured or estimated from the data. They have a clear physical interpretation. From the data (3.109), we thus obtain a sequence of measurements for discrete and periodic movements, respectively

$$\{\mathbf{g}_i, \tau_i\}_{i=1}^M, \text{ and } \{\mathbf{g}_i, r_i, \Omega_i\}_{i=1}^M. \tag{3.112}$$

For every new query point $\mathbf{q}$, the new parameters $\mathbf{g}$, τ, r, and Ω can be estimated using locally weighted distance function by putting higher weight on example queries $\mathbf{q}_i$ that are close to the current query $\mathbf{q}$. The distance weighting error criterion is in this case given by

$$C(\mathbf{q}) = \sum_{i=1}^{M} (z - z_i)^2 K(d(\mathbf{q}, \mathbf{q}_i)), \tag{3.113}$$

where z is one of $g_k,\ k = 1, \ldots, n$, τ, r, and Ω. The solutions are given by

$$\mathbf{g} = \frac{1}{\sum_{i=1}^M K(d(\mathbf{q}, \mathbf{q}_i))} \sum_{i=1}^{M} K(d(\mathbf{q}, \mathbf{q}_i))\mathbf{g}_i, \tag{3.114}$$

$$\mathbf{y}_0 = \frac{1}{\sum_{i=1}^M K(d(\mathbf{q}, \mathbf{q}_i))} \sum_{i=1}^{M} K(d(\mathbf{q}, \mathbf{q}_i))\mathbf{y}_d^i(t_{i,0}), \tag{3.115}$$

$$\tau = \frac{1}{\sum_{i=1}^M K(d(\mathbf{q}, \mathbf{q}_i))} \sum_{i=1}^{M} K(d(q, \mathbf{q}_i))\tau_i, \tag{3.116}$$

$$r = \frac{1}{\sum_{i=1}^M K(d(\mathbf{q}, \mathbf{q}_i))} \sum_{i=1}^{M} K(d(\mathbf{q}, \mathbf{q}_i))r_i, \tag{3.117}$$

$$\Omega = \frac{1}{\sum_{i=1}^M K(d(\mathbf{q}, \mathbf{q}_i))} \sum_{i=1}^{M} K(d(\mathbf{q}, \mathbf{q}_i))\Omega_i. \tag{3.118}$$

There are many possibilities to select the weighting kernel K and distance d [1]. One possible choice is the tricube kernel

$$K(d) = \begin{cases} (1 - |d|^3)^3 & \text{if } |d| < 1 \\ 0 & \text{otherwise} \end{cases}, \tag{3.119}$$

and the weighted Euclidean distance

$$d(\mathbf{q}, \mathbf{q}_i) = \|\mathbf{D}(\mathbf{q} - \mathbf{q}_i)\|,\ \mathbf{D} = \text{diag}(1/a_l) \in \mathbb{R}^{m \times m},\ a_l > 0. \tag{3.120}$$

The tricube kernel K has finite extent and a continuous first and second derivative, which means that the first two derivatives of the prediction (as a function of query points) are also continuous. The finite support of K is useful because it can help reducing the computational complexity of different optimization problems. As discussed in [1], the choice of weighting function is rarely critical for the performance of locally weighted regression.

Generalization of weights by locally weighted regression

The synthesis of shape parameters $\mathbf{w}$ is more complex because these parameters are not directly measurable and do not have any physical interpretation. This makes their computation using a simple weighting approach problematic.

It is therefore better to estimate them using locally weighted regression. Based on Eq. (3.32), we can formulate a locally weighted regression problem to compute the optimal parameters $\mathbf{w}_i$ for the given query point $\mathbf{q}$. We first compute the target values for all example trajectories

$$F^i_{k,j} = \tau^2 \ddot{y}^i_{\mathrm{d},k}(t_{i,j}) + \alpha_z \tau \dot{y}^i_{\mathrm{d},k}(t_{i,j}) - \alpha_z \beta_z (g_k - y^i_{\mathrm{d},k}(t_{i,j})), \tag{3.121}$$

where $i = 1, \ldots, M$, $j = 0, \ldots, T_i$ and $k = 1, \ldots, M$. Writing

$$\mathbf{f}^i_k = [F^i_{k,0}, F^i_{k,1}, \ldots, F^i_{k,N}]^{\mathrm{T}} \tag{3.122}$$

we can formulate the following optimization problems (for all degrees of freedom)

$$\min_{\mathbf{w}_k \in \mathbb{R}^N} \sum_{i=1}^{M} \left\| \mathbf{X}_i \mathbf{w}_k - \frac{1}{g_k - y_{k,0}} \mathbf{f}^i_k \right\|^2 K(d(\mathbf{q}, \mathbf{q}_i)), \tag{3.123}$$

where $1 \leq k \leq n$ is the k-th degree of freedom and $\mathbf{w}_k \in \mathbb{R}^N$ are the DMP weights associated with the k-th degree of freedom. Note that the above criterion includes the goal parameter g_k, the initial configuration $y_{k,0}$ on the trajectory, and the timing constant τ. The generalized $\mathbf{g}$, $\mathbf{y}_0$ and τ as respectively computed in (3.114), (3.115), and (3.116) should be used here.

K and distance d in the space of query points determine how much influence each of the example movements has on the final estimate of the control policy. The influence of each example movement should diminish with the distance of the query point $\mathbf{q}$ from the data point $\mathbf{q}_k$. Ideally, $\mathbf{D}$ should be determined so that the error in the execution of the task is as small as possible. To reduce the resulting problem to a one dimensional optimization problem while still accounting for the possibly varying spacing across dimensions of the query point space, we can define

$$a_i = c * \max_{j=1,\ldots,M} \min_{k=1,\ldots,M} \{|q_{j,i} - q_{k,i}|\}. \tag{3.124}$$

In this way, $\mathbf{D}$ is fully specified by a parameter $c \in \mathbb{R}$. For regularly spaced data, $c \approx 2.2$ often gives good results. With such c the number of example trajectories with nonzero $K(d(\mathbf{q}, \mathbf{q}_k))$ is about 4^m, where m is the dimensionality of the query point. Automatic methods for determining c are also possible. Atkeson et al. [1] suggested to minimize validation set error, which evaluates the difference between the predicted output and the observed value in the validation set, where the data in the validation set are not used for training.

This is a linear least squares problems that is further simplified because the example trajectories for which K vanishes do not influence generalization. Thus the size of the system matrix associated with the objective function (3.123).

The computational complexity of solving the least squares system (3.123) is $\mathcal{O}(N^2T)$, $T = \sum_{k=1}^{M} T_k$, and thus increases linearly with the number of data points considered by LWR and quadratically with the number of radial basis functions used in (3.5) and (3.14), respectively. Due to our choice of weighting kernel K, we normally have $K(d(\mathbf{q}, \mathbf{q}_k)) = 0$ for many k. Moreover, by cutting the support of basis functions (3.5) and (3.14) once their value falls below a certain threshold, matrices $\mathbf{X}_k$ become sparse as well. The quadratic dependence on the number of basis functions is not a problem because this number is generally much lower than the number of data points. In most practical examples, one can expect around $10000 - 50000$ data points and $25 - 50$ basis functions for DMPs. These facts make computational complexity sufficiently low to resolve the least-squares problem (3.123) on the fly using standard methods from sparse matrix algebra.

The calculation of weights for periodic movements is very similar. Instead of Eq. (3.32), the derivation of locally weighted regression problem should be based on Eq. (3.43). To generalize the learned movements to new situations, we store the estimated amplitudes and frequencies and the sampled movements from the last few movement periods, e.g. five, which produced a good movement on the robot as judged by the human teacher. We can now compute the target values for all trajectories

$$F_{k,j}^i = \frac{1}{\Omega^2} \ddot{y}_{\mathrm{d},k}^i(t_{i,j}) + \frac{\alpha_z}{\Omega} \dot{y}_{\mathrm{d},k}^i(t_{i,j}) - \alpha_z \beta_z (g_k - y_{\mathrm{d},k}^i(t_{i,j})), \tag{3.125}$$

where $i = 1, \ldots, M$, $j = 0, \ldots, T_i$ and $k = 1, \ldots, M$. Writing

$$\mathbf{f}_k^i = [F_{k,0}^i, F_{k,1}^i, \ldots, F_{k,N}^i]^{\mathrm{T}} \tag{3.126}$$

we can formulate the optimization problems (for all degrees of freedom)

$$\min_{\mathbf{w}_k \in \mathbb{R}^N} \sum_{i=1}^{M} \left\| \mathbf{X}_i \mathbf{w}_k - \mathbf{f}_k^i \right\|^2 K(d(\mathbf{q}, \mathbf{q}_i)), \tag{3.127}$$

where $1 \leq k \leq n$ is the k-th degree of freedom and $\mathbf{w}_k \in \mathbb{R}^N$ are the DMP weights associated with the k-th degree of freedom. Similar as for discrete DMPs, the above criterion includes the goal parameter g_k, the amplitude r and the frequency Ω. The

procedure CollectTrainingData
acquire trajectory points $\{y_{\mathrm{d}}^k(t_{k,j}), \dot{y}_{\mathrm{d}}^k(t_{k,j}), \ddot{y}_{\mathrm{d}}^k(t_{k,j}) | k = 1, \ldots, M,\ j = 1, \ldots, T_k\}$, e. g. by kinesthetic guiding or direct imitation;
extract the attractor points $\{\mathbf{g}_k\}$, starting points $\{\mathbf{y}_{0,k}\}$ and time constants $\{\tau_k\}$ or amplitudes $\{r_k\}$ and frequencies $\{\Omega_k\}$, $k = 1, \ldots, M$;
associate the acquired trajectories with query points $\{\mathbf{q}_k\}$;

procedure GeneralizeTrajectory
for a given query point $\mathbf{q}^*$
estimate the attractor point $\mathbf{g}^*$, starting point $\mathbf{y}_0^*$, and time constant τ^* or amplitude r^* and frequency Ω^* using (3.114), (3.115) and (3.116) or (3.117) and (3.118), respectively;
minimize (3.123) or (3.127) to estimate the parameters $\mathbf{w}^* \in \mathbb{R}^N$;
output the generalized DMP $\{\mathbf{g}^*, \mathbf{y}_o^*, \tau^*, \{\mathbf{w}_k^*\}\}$ or $\{\mathbf{g}^*, r^*, \Omega^*, \{\mathbf{w}_k^*\}\}$ for query $\mathbf{q}^*$;

Figure 3.14: Training and generalization of goal-directed actions

generalized $\mathbf{g}$, r and Ω as respectively computed by (3.114), (3.117), and (3.118) should be used.

3.7 Summary

Dynamic Movement Primitives (DMPs) represent a powerful and flexible framework for the specification and execution of robot control policies. This chapter introduced the theoretical foundations of DMPs and demonstrated their utility in representing and learning robot motions through a comprehensive approach grounded in dynamic systems theory. By relying on autonomous, nonlinear differential equations, DMPs ensure smooth kinematic trajectories and allow for robust performance across a wide range of tasks and conditions.

DMPs offer a unified framework to model both discrete (point-to-point) and periodic (repetitive) movements, making them suitable for different applications in robotics. They incorporate adjustable parameters that enable the encoding of diverse trajectories while maintaining the stability of the underlying system. This ensures that robots can adapt their movements dynamically without compromising the smoothness or reliability of motion execution.

A distinguishing feature of DMPs is their modular design, which separates the temporal progression of movements (handled by a canonical system) from the spatial shaping of trajectories (handled by transformation systems). This separation allows for time-independent motion representation and facilitates adaptation to varying task durations. Additionally, DMPs integrate nonlinear forcing terms that can be learned from demonstrations using techniques such as locally weighted regression. This capability enables robots to generalize learned trajectories to new scenarios and goals, even in unstructured environments.

The chapter also explored advanced capabilities of DMPs, such as modulation and adaptation. Modulation techniques allow for dynamic adjustments of motion parameters like goal position, trajectory amplitude, and movement frequency. Furthermore, DMPs can incorporate coupling terms to account for sensory feedback, enabling real-time responses to environmental changes. For example, mechanisms like phase stopping and trajectory coupling enhance the robustness of DMPs against perturbations and ensure smooth recovery from disturbances.

Another significant advantage of DMPs is their compatibility with learning from demonstration (LfD) methodologies. DMPs can be initialized from a single demonstration or synthesized from multiple examples to generalize across task variations. This makes them particularly suitable for tasks that require both precision and adaptability, such as object manipulation, assembly, and human-robot collaboration.

The chapter also introduced extensions of DMPs to Cartesian space, where both position and orientation trajectories are represented seamlessly. Quaternion-based formulations provide a singularity-free representation of rotational trajectories, ensuring smooth and robust orientation control. Similarly, periodic DMPs enable the encoding of rhythmic motions, with applications ranging from locomotion to repetitive industrial tasks.

Evaluations of DMPs demonstrated their effectiveness in learning and reproducing complex motion patterns while ensuring stability and robustness. The ability to adapt trajectories in real time based on sensory feedback further underscores their suitability for dynamic and uncertain environments. The integration of DMPs with online learning methods, such as reinforcement learning and statistical modeling, holds promise for extending their capabilities in real-world applications.

In conclusion, DMPs provide a comprehensive and practical framework for robotic motion generation and control. Their blend of stability, flexibility, and adaptability makes them a valuable tool for addressing the challenges of modern robotics.

Chapter 4

Dynamic Systems and Gaussian Mixture Regression

4.1 Robot Motion Representation and Dynamical Systems

DMPs are not the only type of dynamical system that can be used for robot control. An alternative approach also based on dynamical systems has been developed in [9] and [15]. This approach models robot motion as a dynamical system whose evolution encodes the desired robot motion in each robot state. Let's denote the robot's state by $\mathbf{y} \in \mathbb{R}^n$, where $\mathbf{y}$ typically represents the robot's position (and sometimes velocity) and n is the number of degrees of freedom. With this approach, the motion is modeled as a first-order autonomous dynamical system:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \tag{4.1}$$

where $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a continuous and differentiable vector field that defines the rate of change of the state.

To represent the desired motions, functions $\mathbf{f}(\mathbf{y})$ should be designed such that the system converges to a desired attractor point. For a point-to-point motion, the attractor is typically a fixed point $\mathbf{y}^*$, and the goal is to ensure that:

$$\lim_{t\to\infty} \mathbf{y}(t) = \mathbf{y}^* \quad \text{for all} \quad \mathbf{y}(0) \in \mathbb{R}^n. \tag{4.2}$$

The stability of dynamical systems is a critical property in ensuring the desired behavior of robotic motion. For a fixed point $\mathbf{y}^*$ to act as an attractor, the vector field $\mathbf{f}(\mathbf{y})$ must be designed such that the equilibrium point $\mathbf{y}^*$ is globally asymptotically stable. This means that for any initial state $\mathbf{y}(0)$, the state trajectories $\mathbf{y}(t)$ converge to $\mathbf{y}^*$ over time, and small perturbations do not lead to diverging trajectories. The SEDS framework ensures this property by embedding stability guarantees into its design. It leverages Lyapunov theory to ensure that the system's energy-like function decreases monotoni-

cally as the state evolves, enabling robust recovery from perturbations and changes in the environment.

While DMPs are a widely used and powerful approach for motion representation in robotics, the SEDS approach provides a complementary framework with unique strengths. DMPs excel at replicating specific demonstrated trajectories using second-order dynamics with constant coefficients, ensuring inherent stability as the phase variable approaches zero. This stability, combined with their computational efficiency and the ability to adapt to different start and goal configurations, makes DMPs particularly effective for tasks requiring fast, reliable, and precise trajectory reproduction. Moreover, DMPs often require only a single demonstration to encode a trajectory, making them well-suited for applications where simplicity and speed are paramount.

The SEDS approach, however, offers significant advantages in scenarios requiring adaptability and generalization across diverse conditions. Unlike DMPs, which are focused on a single trajectory, SEDS represents motion as a continuous vector field over the state space. This enables SEDS to capture an entire class of trajectories, providing robust responses to perturbations, varying initial conditions, and dynamic environmental changes. By directly encoding attractor dynamics within the representation, SEDS eliminates the need for replanning when deviations occur—a feature shared with DMPs.

In the rest of this chapter, we explain how to design functions $\mathbf{f}$ with stable attractor dynamics and how to learn such functions from human demonstrations using the SEDS framework.

4.2 Lyapunov Stability Theory and Conditions

To design functions $\mathbf{f}$ with stable attractor dynamics, we first introduce the Lyapunov stability theory. The Lyapunov theory provides a framework to analyze the stability of dynamical systems. A system described by $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ is said to be stable if small perturbations in the initial state $\mathbf{y}(0)$ result in bounded deviations of $\mathbf{y}(t)$ for all future times. The stability conditions are assessed using a Lyapunov function $V(\mathbf{y}) : \mathbb{R}^n \to \mathbb{R}$, which acts as an energy-like measure of the system's state.

The key concepts are as follows:

1. **Stability**: The attractor point $\mathbf{y}^*$ is stable if, for every $\epsilon > 0$, there exists a $\delta > 0$ such that $\|\mathbf{y}(0) - \mathbf{y}^*\| < \delta$ implies $\|\mathbf{y}(t) - \mathbf{y}^*\| < \epsilon$ for all $t \geq 0$.

2. **Asymptotic Stability**: The attractor point $\mathbf{y}^*$ is asymptotically stable if it is stable and, in addition, $\lim_{t\to\infty} \mathbf{y}(t) = \mathbf{y}^*$ for all $\mathbf{y}(0)$ in a neighborhood of $\mathbf{y}^*$.

3. **Global Asymptotic Stability**: The equilibrium point $\mathbf{y}^*$ is globally asymptotically stable if $\lim_{t\to\infty} \mathbf{y}(t) = \mathbf{y}^*$ for all $\mathbf{y}(0) \in \mathbb{R}^n$.

The Lyapunov function $V(\mathbf{y})$ must satisfy the following conditions to prove stability:

- $V(\mathbf{y}) > 0$ for all $\mathbf{y} \neq \mathbf{y}^*$ and $V(\mathbf{y}^*) = 0$ (positive definiteness),
- $\dot{V}(\mathbf{y}) = \nabla V(\mathbf{y})^\top \mathbf{f}(\mathbf{y}) \leq 0$ (negative semi-definiteness),
- For asymptotic stability, $\dot{V}(\mathbf{y}) < 0$ for all $\mathbf{y} \neq \mathbf{y}^*$,
- For global asymptotic stability, $V(\mathbf{y}) \to \infty$ as $\|\mathbf{y}\| \to \infty$ (radially unboundedness).

An example of a candidate Lyapunov function is the quadratic function:

$$V(\mathbf{y}) = \frac{1}{2}\|\mathbf{y} - \mathbf{y}^*\|^2, \tag{4.3}$$

which ensures that the system's trajectories converge to $\mathbf{y}^*$ if $\dot{V}(\mathbf{y}) < 0$.

By analyzing the above conditions, we can prove the stability of an attractor without computing an explicit solution to the system's differential equations. This is in contrast to indirect methods, which rely on analyzing the solutions of the system. The direct method evaluates stability by constructing a Lyapunov function and examining its behavior along the system's trajectories. This makes it particularly powerful for analyzing nonlinear systems where explicit solutions are often infeasible. Stability of the attractor is guaranteed if a Lyapunov function $V(\mathbf{y})$ exist such that:

1. $V(\mathbf{y}) > 0$ for all $\mathbf{y} \neq \mathbf{y}^*$ and $V(\mathbf{y}^*) = 0$,
2. $\dot{V}(\mathbf{y}) = \nabla V(\mathbf{y})^\top \mathbf{f}(\mathbf{y}) < 0$ for all $\mathbf{y} \neq \mathbf{y}^*$.

Its time derivative along the trajectories of the system is:

$$\dot{V}(\mathbf{y}) = (\mathbf{y} - \mathbf{y}^*)^\top \mathbf{f}(\mathbf{y}). \tag{4.4}$$

To ensure stability, $\mathbf{f}(\mathbf{y})$ must be designed such that $\dot{V}(\mathbf{y}) < 0$ for $\mathbf{y} \neq \mathbf{y}^*$.

4.3 Design of Dynamic Systems by PBD

The desired motion is often learned from user demonstrations. Given a dataset of trajectories $\mathcal{D} = \{(\mathbf{y}_i, \dot{\mathbf{y}}_i)\}_{i=1}^N$, the goal is to fit a function $\mathbf{f}(\mathbf{y})$ that replicates the observed behavior and fulfills the Lyapunov stability criteria. This can be achieved using probabilistic models, e.g. Gaussian Mixture Models (GMM) combined with Gaussian Mixture Regression (GMR).

4.3.1 Data Collection

A human demonstrates the desired behavior multiple times. From each trajectory we sample the recorded states and their corresponding velocities:

$$\mathcal{D} = \{\{(\mathbf{y}_{n,t}, \dot{\mathbf{y}}_{n,t})\}_{t=0}^{T_n}\}_{n=1}^N, \tag{4.5}$$

where n indexes the user demonstrations and t the recorded positions and velocities on the demonstrated motion. The number of demonstrations needed depends on the complexity of the task and the variability of the trajectories. For tasks with low variability, a few demonstrations (e.g., 3-5) may suffice to capture the key characteristics of the motion. For tasks with higher variability or complex dynamics, more demonstrations are required to accurately compute the dynamical system. It is crucial to ensure that the demonstrations cover the entire state space relevant to the task to prevent poor generalization.

4.3.2 Gaussian Mixture Model (GMM)

A Gaussian Mixture Model (GMM) is a probabilistic model used to represent data as a mixture of multiple Gaussian distributions. It is particularly effective for modeling complex data distributions and is widely used in tasks such as clustering, density estimation, and trajectory modeling in robotics. It is defined as follows:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{4.6}$$

where:

- K is the number of Gaussian components,
- π_k are the mixing coefficients, with $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$,
- $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the k-th Gaussian component with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$.

Each Gaussian component is expressed as:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right), \tag{4.7}$$

where d is the dimensionality of $\mathbf{x}$ and $|\boldsymbol{\Sigma}_k|$ is the determinant of $\boldsymbol{\Sigma}_k$. The parameters of a GMM include:

- mixing coefficients π_k,
- means $\boldsymbol{\mu}_k$,
- covariance matrices $\boldsymbol{\Sigma}_k$.

These parameters are typically estimated using the Expectation-Maximization (EM) algorithm, which iteratively maximizes the likelihood of the data under the model.

In robotics, GMMs are often used to model trajectory distributions. Given a set of demonstrated trajectories, GMMs can encode the variability and correlations in the data. This way we provide a compact representation for learning and reproducing motion.

To derive the velocity field function $\mathbf{f}$ from (4.1), we model the dataset $\mathcal{D}$ defined in Eq. (4.5) using a Gaussian Mixture Model (GMM):

$$p(\mathbf{y}, \dot{\mathbf{y}}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{y}, \dot{\mathbf{y}}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{4.8}$$

where $p(\mathbf{y}, \dot{\mathbf{y}})$ is the joint probability of position $\mathbf{y}$ and velocity $\dot{\mathbf{y}}$, π_k are the mixing coefficients, $\sum_{k=1}^{K} \pi_k = 1$ and $\boldsymbol{\mu}_k \in \mathbb{R}^{2n}$, $\boldsymbol{\Sigma}_k \in \mathbb{R}^{2n \times 2n}$ are the mean and covariance of the k-th component

$$\mathcal{N}(\mathbf{y}, \dot{\mathbf{y}}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^n |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2} \left(\begin{bmatrix} \mathbf{y} \\ \dot{\mathbf{y}} \end{bmatrix} - \boldsymbol{\mu}_k \right)^\top \boldsymbol{\Sigma}_k^{-1} \left(\begin{bmatrix} \mathbf{y} \\ \dot{\mathbf{y}} \end{bmatrix} - \boldsymbol{\mu}_k \right) \right). \tag{4.9}$$

Model (4.8) can also be interpreted as

$$p(\mathbf{y}, \dot{\mathbf{y}}) = \sum_{k=1}^{K} P(k) p(\mathbf{y}, \dot{\mathbf{y}} | k), \tag{4.10}$$

where $P(k) = \pi_k$ denotes the probability of picking the k-th component and $p(\mathbf{y}, \dot{\mathbf{y}}|k) = \mathcal{N}(\mathbf{y}, \dot{\mathbf{y}}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the conditional probability that the datapoint $[\mathbf{y}^\top, \dot{\mathbf{y}}^\top]^\top$ belongs to the k-th component.

The likelihood of the data under the GMM is given by

$$\mathcal{L} = \prod_{n=1}^{N} \prod_{t=0}^{T_n} \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\begin{bmatrix} \mathbf{y}_{n,t} \\ \dot{\mathbf{y}}_{n,t} \end{bmatrix}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \right), \tag{4.11}$$

and the log-likelihood, which is typically maximized, is given by:

$$\ln(\mathcal{L}) = \sum_{n=1}^{N} \sum_{t=0}^{T_n} \ln\left(\sum_{k=1}^{K} \pi_k \mathcal{N}\left(\begin{bmatrix} \mathbf{y}_{n,t} \\ \dot{\mathbf{y}}_{n,t} \end{bmatrix}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \right) \right). \tag{4.12}$$

The number of mixture components K is a critical parameter that affects the model's ability to represent the underlying data. In the context of threjectory representation by DS, K determines the resolution at which the trajectory dynamics are modeled. Each Gaussian component corresponds to a region of the state space, capturing the local behavior of the system. Thus, selecting K has direct implications for the fidelity of the learned motion. A concrete procedure for determining K is provided in Section 4.3.5.

4.3.3 Gaussian Mixture Regression (GMR)

Gaussian Mixture Regression (GMR) is a technique derived from a Gaussian Mixture Model (GMM) to model the conditional distribution of output variables given input variables. We start by assuming that the joint probability distribution of $[\mathbf{y}^\top, \dot{\mathbf{y}}^\top]^\top$ is modeled

by a GMM as in (4.8). Let's write

$$\boldsymbol{\mu}_k = \begin{bmatrix} \boldsymbol{\mu}_{k,\mathbf{y}} \\ \boldsymbol{\mu}_{k,\dot{\mathbf{y}}} \end{bmatrix}, \tag{4.13}$$

$$\boldsymbol{\Sigma}_k = \begin{bmatrix} \boldsymbol{\Sigma}_{k,\mathbf{y}} & \boldsymbol{\Sigma}_{k,\mathbf{y},\dot{\mathbf{y}}} \\ \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}},\mathbf{y}} & \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}}} \end{bmatrix}. \tag{4.14}$$

The conditional distribution $p(\dot{\mathbf{y}}|\mathbf{y})$ for each component is then given as:

$$p(\dot{\mathbf{y}}|\mathbf{y}, k) = \mathcal{N}(\mathbf{y}; \widehat{\boldsymbol{\mu}}_{k,\mathbf{y}}, \widehat{\boldsymbol{\Sigma}}_{k,\mathbf{y}}), \tag{4.15}$$

where:

$$\widehat{\boldsymbol{\mu}}_k = \boldsymbol{\mu}_{k,\dot{\mathbf{y}}} + \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}},\mathbf{y}}\boldsymbol{\Sigma}_{k,\mathbf{y}}^{-1}(\mathbf{y} - \boldsymbol{\mu}_{k,\mathbf{y}}), \tag{4.16}$$

$$\widehat{\boldsymbol{\Sigma}}_k = \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}}} - \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}},\mathbf{y}}\boldsymbol{\Sigma}_{k,\mathbf{y}}^{-1}\boldsymbol{\Sigma}_{k,\mathbf{y},\dot{\mathbf{y}}}. \tag{4.17}$$

The overall conditional distribution $p(\dot{\mathbf{y}}|\mathbf{y})$ is a mixture of the conditional distributions for all components:

$$p(\dot{\mathbf{y}}|\mathbf{y}) = \sum_{k=1}^{K} h_k(\mathbf{y})\mathcal{N}(\mathbf{y}; \widehat{\boldsymbol{\mu}}_k, \widehat{\boldsymbol{\Sigma}}_k), \tag{4.18}$$

where $h_k(\mathbf{y})$ are the responsibilities:

$$h_k(\mathbf{y}) = \frac{P(k)p(\mathbf{y}|k)}{\sum_{i=1}^{K} P(i)p(\mathbf{y}|i)} = \frac{\pi_k\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_{k,\mathbf{y}}, \boldsymbol{\Sigma}_{k,\mathbf{y}})}{\sum_{i=1}^{K} \pi_i\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_{i,\mathbf{y}}, \boldsymbol{\Sigma}_{i,\mathbf{y}})}, \tag{4.19}$$

The expected value of $\dot{\mathbf{y}}$ is then computed as follows:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) = \mathbb{E}[\dot{\mathbf{y}}|\mathbf{y}] = \sum_{k=1}^{K} h_k(\mathbf{y})\hat{\boldsymbol{\mu}}_k. \tag{4.20}$$

Writing

$$\mathbf{A}_k = \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}},\mathbf{y}}\boldsymbol{\Sigma}_{k,\mathbf{y}}^{-1}, \tag{4.21}$$

$$\mathbf{b}_k = \boldsymbol{\mu}_{k,\dot{\mathbf{y}}} - \boldsymbol{\Sigma}_{k,\dot{\mathbf{y}},\mathbf{y}}\boldsymbol{\Sigma}_{k,\mathbf{y}}^{-1}\boldsymbol{\mu}_{k,\mathbf{y}} = \boldsymbol{\mu}_{k,\dot{\mathbf{y}}} - \mathbf{A}_k\boldsymbol{\mu}_{k,\mathbf{y}}, \tag{4.22}$$

we can rewrite Eq. (4.16) as

$$\widehat{\boldsymbol{\mu}}_k = \boldsymbol{\mu}_{k,\dot{\mathbf{y}}} + \mathbf{A}_k(\mathbf{y} - \boldsymbol{\mu}_{k,\mathbf{y}}) = \mathbf{A}_k\mathbf{y} + \mathbf{b}_k. \tag{4.23}$$

We obtain the following expression for the velocity field $\dot{\mathbf{y}}$

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) = \sum_{k=1}^{K} h_k(\mathbf{y})(\mathbf{A}_k\mathbf{y} + \mathbf{b}_k). \tag{4.24}$$

GMR provides a smooth and probabilistic mapping from $\mathbf{y}$ to $\dot{\mathbf{y}}$, making it ideal for tasks like trajectory learning and motion planning in robotics.

4.3.4 Stable Estimator of Dynamical Systems (SEDS)

The SEDS framework is a powerful approach for learning stable motion dynamics from demonstrations. By leveraging principles from Lyapunov stability theory, SEDS ensures that the resulting dynamical system converges to a desired attractor point while reproducing the demonstrated motion. This makes it particularly suitable for robotics applications where robustness and adaptability are critical, such as trajectory planning, manipulation, and navigation tasks.

SEDS models the motion as a mixture of Gaussians, where each component captures a segment of the observed behavior. The framework enforces global asymptotic stability by constraining the parameters of the learned dynamical system. These stability guarantees are achieved through a Lyapunov function, which ensures that the system's trajectories converge to the desired attractor from any initial condition within the state space.

A key strength of SEDS lies in its ability to balance accurate reproduction of demonstrated motions with mathematical stability guarantees. This dual objective is formulated as an optimization problem, where the parameters of the Gaussian Mixture Model (GMM) are estimated while satisfying stability constraints. The following sections present the mathematical formulation of this optimization problem and its variants.

Next we explain how the unknown parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ forming the Gaussian mixture model (4.8) are computed in SEDS. The standard approach of applying an expectation-maximization algorithm to estimate these parameters is insufficient because with a classic EM-algorithm it is not possible to guarantee that the resulting system (4.24) is asymptotically stable. Thus in the following we derive sufficient conditions for the estimation of parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ that guarantee the asymptotic stability of (4.24).

To prove stability we utilize Lyapunov function

$$V(\mathbf{y}) = \frac{1}{2}(\mathbf{y} - \mathbf{y}^*)^\top(\mathbf{y} - \mathbf{y}^*). \tag{4.25}$$

Its derivative is given by

$$\begin{aligned}
\dot{V}(\mathbf{y}) &= \frac{dV}{d\mathbf{y}}\frac{d\mathbf{y}}{dt} = \frac{1}{2}\left(\frac{d}{d\mathbf{y}}((\mathbf{y}-\mathbf{y}^*)^\top(\mathbf{y}-\mathbf{y}^*))\right)\dot{\mathbf{y}} = (\mathbf{y}-\mathbf{y}^*)^\top\dot{\mathbf{y}} \\
&= (\mathbf{y}-\mathbf{y}^*)^\top\sum_{k=1}^{K} h_k(\mathbf{y})(\mathbf{A}_k\mathbf{y}+\mathbf{b}_k) \\
&= (\mathbf{y}-\mathbf{y}^*)^\top\sum_{k=1}^{K} h_k(\mathbf{y})(\mathbf{A}_k(\mathbf{y}-\mathbf{y}^*)+\mathbf{A}_k\mathbf{y}^*+\mathbf{b}_k) \\
&= \sum_{k=1}^{K} h_k(\mathbf{y})\left((\mathbf{y}-\mathbf{y}^*)^\top\mathbf{A}_k(\mathbf{y}-\mathbf{y}^*)+(\mathbf{y}-\mathbf{y}^*)^\top(\mathbf{A}_k\mathbf{y}^*+\mathbf{b}_k)\right) \qquad (4.26)
\end{aligned}$$

The definition of responsibilities (4.19) guarantees that $h_k(\mathbf{y}) > 0$. Note also that

$$\begin{aligned}
(\mathbf{y}-\mathbf{y}^*)^\top\mathbf{A}_k(\mathbf{y}-\mathbf{y}^*) &= (\mathbf{y}-\mathbf{y}^*)^\top\left(\frac{1}{2}(\mathbf{A}_k+\mathbf{A}_k^\top)+\frac{1}{2}(\mathbf{A}_k-\mathbf{A}_k^\top)\right)(\mathbf{y}-\mathbf{y}^*) \\
&= \frac{1}{2}(\mathbf{y}-\mathbf{y}^*)^\top(\mathbf{A}_k+\mathbf{A}_k^\top)(\mathbf{y}-\mathbf{y}^*). \qquad (4.27)
\end{aligned}$$

We can thus guarantee that $\dot{V}(\mathbf{y}) < 0\ \forall\mathbf{y}$ if symmetric matrices $\mathbf{A}_k + \mathbf{A}_k^\top$ are negative-definite, i.e. $\mathbf{A}_k + \mathbf{A}_k^\top \prec 0$, and if $\mathbf{A}_k\mathbf{y}^* + \mathbf{b}_k = 0$. Note that these are sufficient, not necessary conditions. As explained in Section 4.2, it follows from the Lyapunov theory that an arbitrary differential equation system (4.24) with negative definite matrices $\mathbf{A}_k + \mathbf{A}_k^\top$ and fulfilling $\mathbf{A}_k\mathbf{y}^* + \mathbf{b}_k = 0$ is asymptotically stable at target $\mathbf{y}^*$.

We can now finally define an optimization problem to compute the unknown parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$. Two different optimization problems have been proposed in [15]. The Stable Estimator of Dynamical Systems (SEDS) Likelihood framework optimizes a log-likelihood function while ensuring global asymptotic stability. This results in the following optimization problem:

$$\begin{aligned}
\min_{\pi_k,\,\boldsymbol{\mu}_k,\,\boldsymbol{\Sigma_k}} \quad & \mathcal{L}(\theta) = \frac{1}{T}\sum_{n=1}^{N}\sum_{t=0}^{T_n}\log\left(\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{y}_{n,t},\dot{\mathbf{y}}_{n,t};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right), \\
\text{subject to:} \quad & \mathbf{A}_k\mathbf{y}^* + \mathbf{b}_k = 0, \forall k \\
& \mathbf{A}_k + \mathbf{A}_k^\top \prec 0,\ \forall k \\
& \boldsymbol{\Sigma}_k \succ 0,\ \forall k \\
& 0 < \pi_k < 1,\ \forall k \\
& \sum_{k=1}^{K}\pi_k = 1
\end{aligned} \qquad (4.28)$$

Here T is the overall number of datapoints in (4.5). Alternatively, SEDS Mean Squared

Error framework optimizes

$$\begin{aligned}
\min_{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma_k}} \quad & \mathcal{J}(\theta) = \frac{1}{2T} \sum_{n=1}^{N} \sum_{t=0}^{T_n} \|\dot{\mathbf{y}}_{n,t} - \mathbf{f}(\mathbf{y}_{n,t})\|^2, \\
\text{subject to:} \quad & \mathbf{A}_k \mathbf{y}^* + \mathbf{b}_k = 0, \forall k \\
& \mathbf{A}_k + \mathbf{A}_k^\top \prec 0, \ \forall k \\
& \boldsymbol{\Sigma}_k \succ 0, \ \forall k \\
& 0 < \pi_k < 1, \ \forall k \\
& \sum_{k=1}^{K} \pi_k = 1
\end{aligned} \tag{4.29}$$

SEDS-Likelihood and SEDS-MSE optimization problems can be solved using standard libraries for nonlinear programming.

The SEDS-likelihood approach optimizes the probability of the observed data under the Gaussian Mixture Model (GMM) used to represent the dynamics. It seeks to find the parameters that maximize the likelihood of the data, leveraging the probabilistic nature of GMM. It provides a robust probabilistic framework, making it well-suited for tasks involving uncertainty or noisy demonstrations. By capturing the underlying data distribution, it can effectively handle complex or multimodal trajectory patterns. It is particularly valuable in applications where uncertainty quantification is important, as the probabilistic nature allows for modeling the variance in trajectories. However, the resulting optimization process can be computationally expensive due to the need to calculate the likelihood for multiple components and enforce stability constraints. It may be less intuitive for users focused on direct trajectory reproduction, as the emphasis is on probability rather than error minimization.

The SEDS-MSE approach, on the other hand, directly minimizes the error between the velocities predicted by the learned model and those observed in the demonstrations. It focuses on accurate trajectory reproduction by reducing the Euclidean distance between predicted and observed values. The objective is straightforward and intuitive, as it directly minimizes the mismatch between the demonstrated and reproduced velocities.It is computationally efficient as it avoids the probabilistic calculations required by the SEDS-likelihood approach. This approach is easier to interpret and validate in terms of achieving precise trajectory reproduction. However, unlike the likelihood approach, it does not explicitly account for uncertainty or variability in the data, which can be a limitation in tasks with noisy or multimodal data. It may be less robust to outliers or variations in the demonstrations, as it focuses solely on minimizing the deterministic error.

In summary, the SEDS-likelihood approach is probabilistic, focusing on capturing the underlying distribution of the data and ensuring robustness to noise. It is computationally intensive but better suited for tasks that involve significant variability or require modeling uncertainty. In contrast, the SEDS-MSE approach prioritizes accuracy and

simplicity, directly targeting the precise reproduction of demonstrated trajectories. It is computationally lighter and more intuitive but less equipped to handle variability and noise.

4.3.5 Selecting Optimal Number of Mixture Components for SEDS

In the SEDS framework, the choice of K must also account for the stability constraints imposed during parameter estimation. A larger K provides greater flexibility to model complex dynamics but may introduce challenges in satisfying the Lyapunov stability conditions. Conversely, a smaller K simplifies stability enforcement but may lack the resolution to accurately capture the demonstrated trajectories.

To ensure a balance, the designer must ensure that K is sufficient to capture the variability in the demonstration data, including transitions between different regions of the state space. On the other hand, K should not be too large to such that the optimization problem remains computationally tractable while satisfying the stability constraints. A smaller K often results in a more interpretable model, which can be beneficial for understanding the learned dynamics.

In practice, K is usually determined experimentally, starting with a smaller number and increasing it until the model adequately captures the variability in the demonstrated trajectories. The selection process should also account for the trade-offs between accuracy, stability, and computational complexity inherent to the SEDS framework:

Initial Selection Based on Data Complexity Start with a small value of K (e.g., 2-3) to capture the basic structure of the data. Increase K iteratively, adding components as necessary to capture finer details in the trajectories.

Evaluation Metrics Use model selection criteria such as the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) to balance model fit and complexity. These criteria are defined as:

$$\text{AIC} = 2m - 2\ln(\widehat{\mathcal{L}}), \tag{4.30}$$

$$\text{BIC} = m\ln(T) - 2\ln(\widehat{\mathcal{L}}), \tag{4.31}$$

where m is the number of estimated parameters $(\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, T is the number of data points, and $\ln(\widehat{\mathcal{L}})$ is the maximized log-likelihood of the model as specified in (4.12). Lower AIC or BIC scores indicate better trade-offs between model complexity and data fit.

Cross-Validation : Divide the data into training and validation sets. Fit the GMM with different values of K and evaluate the performance on the validation set. Select the K that minimizes error on the validation set while avoiding overfitting.

Stability Analysis : Ensure that the chosen K allows the model to satisfy the Lyapunov stability constraints in the SEDS framework. Test the model's ability to converge to the attractor for a range of initial conditions.

4.3.6 Accuracy of Motion Reproduction with SEDS

To evaluate the performance of Stable Estimator of Dynamical Systems (SEDS) framework, we analyze its performance in reproducing demonstrated trajectories and its ability to handle perturbations. While the accuracy of motion reproduction with SEDS is usually not as high as what can be achieved with Dynamic Movement Primitives (DMPs), the system reliably converges to the desired attractor. This convergence ensures robustness and adaptability in dynamic environments, a key advantage of the SEDS framework over methods focusing solely on trajectory accuracy.

Increasing the number of mixture components K in the Gaussian Mixture Model (GMM) used by SEDS could improve the accuracy of motion reproduction. A larger K allows the system to capture finer details in the demonstrated trajectories. However, this improvement in accuracy comes at the cost of increased complexity in the optimization problem. As K grows, ensuring that the stability constraints are satisfied becomes more computationally challenging, and the resulting system may be harder to interpret.

Figure 4.2 illustrates how Gaussian Mixture Models (GMM) within the SEDS framework approximate demonstrated 2-D movements by displaying the mean values and covariance matrices of the Gaussians alongside the reproduced trajectories. Each Gaussian is represented as an ellipse, where the center denotes the mean and the shape and size reflect the covariance matrix, indicating the variance and orientation of the Gaussian. The trajectories pass through these ellipses, demonstrating how the means act as attractors that guide the motion while the covariance influences the spread and variability around the mean. The number of components determines the level of detail captured: with more Gaussians, finer details of the motion are preserved, while fewer Gaussians yield a smoother but less precise approximation. As the trajectory progresses, it transitions smoothly between regions dominated by different Gaussians, blending their influences based on proximity and variance. This ensures a continuous and stable motion. The visual representation in both position and phase planes highlights how each Gaussian shapes specific segments of the trajectory, creating a dynamic interplay that accurately and stably replicates the demonstrated movement.

Figure 4.3 illustrates the global asymptotic stability of the SEDS framework through streamlines that visualize the system's behavior. Each streamline represents the trajectory of the system as it evolves from a different initial position within the state space. The graphs highlight that, regardless of the initial starting position, all trajectories converge to the desired attractor point. This behavior demonstrates the stability of the learned dynamical system and validates its ability to generalize across diverse initial conditions.

The streamlines underscore the core strength of SEDS in embedding stability directly

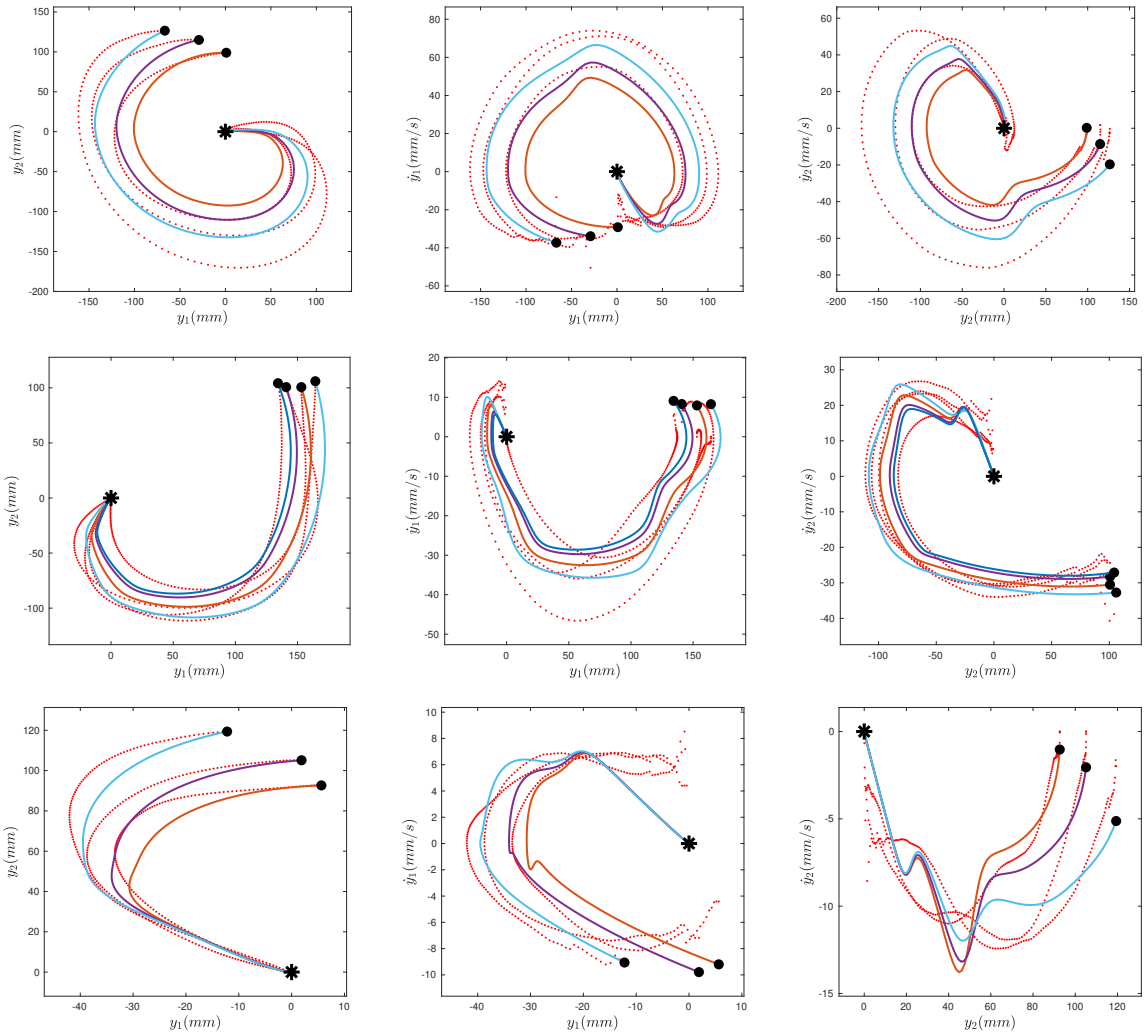

Figure 4.1: Each row in the figure shows the reproduction of demonstrated 2-D movements with a single dynamic system. The red dotted lines show the demonstrated trajectories and the full colored lines the trajectories reproduced by the computed dynamic system. The figures show trajectories in (y_1, y_2) position plane and $(y_1, \dot{y}_1)$ and $(y_2, \dot{y}_2)$ phase planes.

within the learned model. Unlike approaches that rely on external control mechanisms to recover from perturbations, the SEDS framework inherently returns to the desired trajectory due to its attractor dynamics. This feature makes it particularly suitable for applications requiring consistent convergence to target points in dynamic or uncertain environments.

Beyond accuracy, a key strength of SEDS is its inherent robustness to perturbations. The dynamic system, governed by Lyapunov stability principles, ensures that the system trajectories naturally return to the desired motion even when subjected to disturbances, as demonstrated in Fig. 4.4.

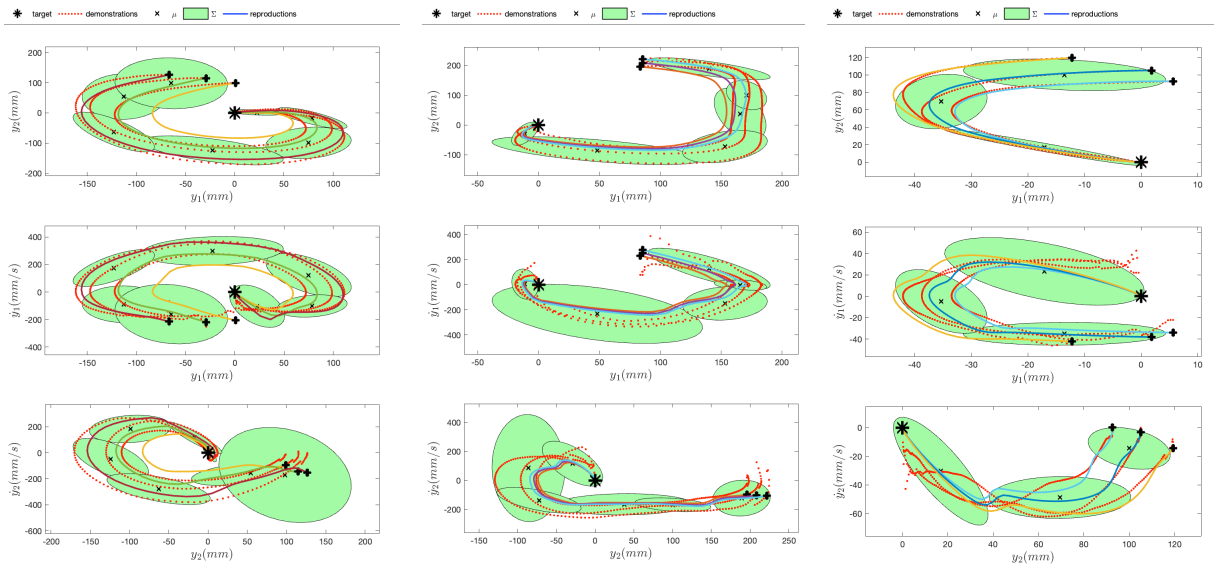

Figure 4.2: Each column in the figure shows the Gaussian mixture models used to represent the demonstrated movements. The means and covariance matrices in (y_1, y_2) position plane and phase spaces $(y_1, \dot{y}_1)$ and $(y_2, \dot{y}_2)$ are shown. The same data as in Fig. 4.1 was used to train the models.

4.4 Summary

The Stable Estimator of Dynamical Systems (SEDS) framework offers a robust, flexible, and mathematically grounded approach to learning stable dynamical systems for robot motion representation. This chapter has explored the foundational principles, design, and evaluation of SEDS, emphasizing its unique advantages and limitations when compared to other methods such as Dynamic Movement Primitives (DMPs).

SEDS leverages Gaussian Mixture Models (GMM) and Gaussian Mixture Regression (GMR) to encode the variability in demonstrated motions while ensuring stability through Lyapunov theory [15, 16]. This guarantees that the system converges to the desired attractor point under all initial conditions. This built-in stability is a significant advantage, allowing SEDS to respond adaptively to perturbations and environmental variations without requiring external stabilization mechanisms.

A key strength of SEDS lies in its ability to balance motion reproduction accuracy

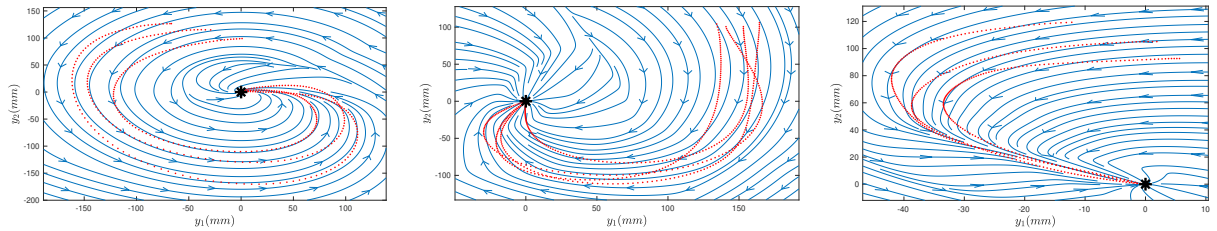

Figure 4.3: Convergence towards the attractor point for different starting positions. The same data as in Fig. 4.1 was used to train the models.

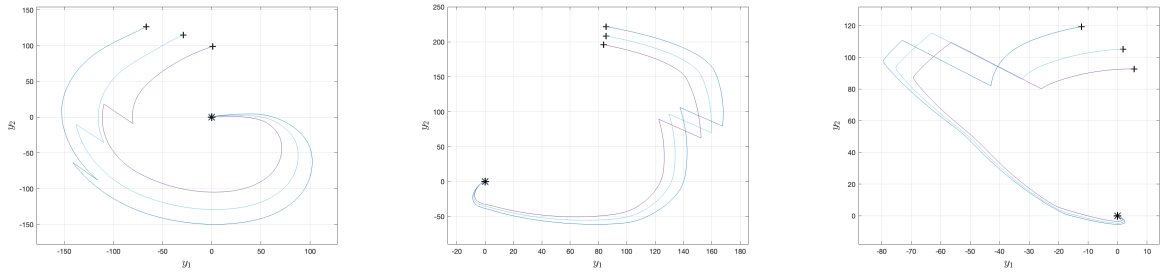

Figure 4.4: Robustness of the learned models to perturbations. The same data as in Fig. 4.1 was used to train the models. During motion reproduction, the robot's end-effector at $t = 1$ second with the displacement vector $[-30; 30]$ mm.

with stability guarantees. While its accuracy in reproducing individual trajectories may not match that of DMPs, SEDS excels in tasks requiring generalization and adaptability. Its ability to recover from perturbations and return to the desired trajectory makes it particularly suitable for dynamic and uncertain environments [9]. However, the choice of the number of mixture components, K, in the GMM significantly impacts performance. A larger K can improve the fidelity of motion reproduction by capturing finer details in trajectories but adds complexity to the optimization process, potentially affecting computational efficiency.

Unlike DMPs, which prioritize simplicity and efficient trajectory reproduction, SEDS's strength lies in its broader applicability to tasks that demand adaptability and inherent stability. While well-suited for continuous state spaces, one limitation of SEDS is that ensuring stability for complex trajectories with many attractors may require careful tuning or domain-specific heuristics, which can increase implementation complexity. Recent advancements, such as learning Lyapunov functions alongside policies using neural networks, further extend the capabilities of SEDS [4].

Compared to other trajectory planning techniques, such as polynomials and splines, SEDS benefits from its dynamic nature. Splines, while optimized for smoothness and precision in predefined paths, are static representations that require external intervention to handle perturbations. In contrast, SEDS inherently adapts through its vector field dynamics, making it more robust to disturbances and better suited for complex, changing environments [30].

In summary, SEDS provides a powerful framework for designing dynamic systems that are accurate in reproducing demonstrated motions while being inherently stable and robust. Its applicability extends to various robotics tasks, including manipulation, navigation, and interaction, where adaptability and stability are paramount. Despite requiring careful parameter selection and computational resources for complex tasks, SEDS remains a cornerstone method for learning and representing stable dynamical systems in robotics.

Chapter 5

Probabilistic Movement Primitives (ProMPs)

Probabilistic Movement Primitives (ProMPs) provide a powerful approach to learning and reproducing motion in robotics, extending the versatility and adaptability of previous frameworks like Dynamic Movement Primitives (DMPs) and Stable Estimator of Dynamical Systems (SEDS) with some unique features. Unlike DMPs, which focus on deterministic trajectory generation, or SEDS, which emphasizes stability through Lyapunov functions, ProMPs employ a probabilistic formulation to capture the variability and uncertainty inherent in motion demonstrations. This chapter explores the foundations of ProMPs, their mathematical formulation, and their comparative advantages over DMPs and SEDS.

5.1 Mathematical Formulation of ProMPs

At the core of ProMPs lies the assumption that trajectories can be represented as a linear combination of basis functions with weights drawn from a multivariate Gaussian distribution. Formally, for a single degree of freedom (DoF), a trajectory $\mathbf{y}(t)$, which now includes both position $y(t)$ and velocity $\dot{y}(t)$, is expressed as:

$$\mathbf{y}(t) = \mathbf{\Phi}(t)\mathbf{w} + \boldsymbol{\epsilon}, \tag{5.1}$$

where $\mathbf{\Phi}(t)$ is a $2\times n$ matrix of time-dependent basis functions, $\mathbf{w} \in \mathbb{R}^n$ is an n-dimensional vector of weights, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_\epsilon)$ is Gaussian noise capturing trajectory variability, with $\mathbf{\Sigma}_\epsilon = \begin{bmatrix} \sigma_y^2 & 0 \\ 0 & \sigma_{\dot{y}}^2 \end{bmatrix}$ to account for different noise levels in position and velocity.

The state vector $\mathbf{y}(t)$ is given by:

$$\mathbf{y}(t) = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix}, \tag{5.2}$$

which ensures that both position and velocity are included in the representation.

Each row of the basis function matrix $\mathbf{\Phi}(t)$ is defined as:

$$\mathbf{\Phi}(t) = \begin{bmatrix} \boldsymbol{\phi}(t) \\ \dot{\boldsymbol{\phi}}(t) \end{bmatrix}, \tag{5.3}$$

where $\boldsymbol{\phi}(t)$ is an $n \times 1$ vector of basis functions for position, and $\dot{\boldsymbol{\phi}}(t)$ is its derivative with respect to time, representing the basis functions for velocity. Each element of $\boldsymbol{\phi}(t)$ is defined as:

$$\phi_i(t) = \exp\left(-\frac{(t - c_i)^2}{2h_i^2}\right), \tag{5.4}$$

where c_i and h_i are the center and width of the i-th basis function, respectively. These parameters determine the temporal placement and scale of the basis functions, enabling flexible representation of trajectories. The complete vector of basis functions is then given by:

$$\boldsymbol{\phi}(t) = \begin{bmatrix} \phi_1(t) & \phi_2(t) & \dots & \phi_n(t) \end{bmatrix}, \tag{5.5}$$

where n is the number of Gaussian basis functions.

The coefficients of the matrix $\mathbf{\Phi}(t)$ are defined as:

$$\mathbf{\Phi}(t) = \begin{bmatrix} \phi_1(t) & \phi_2(t) & \dots & \phi_n(t) \\ \dot{\phi}_1(t) & \dot{\phi}_2(t) & \dots & \dot{\phi}_n(t) \end{bmatrix}, \tag{5.6}$$

where $\phi_i(t)$ is the i-th basis function evaluated at time t, and $\dot{\phi}_i(t)$ is its time derivative. This formulation ensures that $\mathbf{\Phi}(t)$ encapsulates both position and velocity contributions for all basis functions.

The weights $\mathbf{w}$ are modeled probabilistically:

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_w, \mathbf{\Sigma}_w), \tag{5.7}$$

where $\boldsymbol{\mu}_w \in \mathbb{R}^n$ and $\mathbf{\Sigma}_w \in \mathbb{R}^{n \times n}$ are the mean and covariance matrix of the weight distribution, respectively. This probabilistic modeling enables ProMPs to capture both the mean behavior and the variability of motion across multiple demonstrations.

5.1.1 Using Phase Variables Instead of Time

While the primary formulation of ProMPs uses time t to index the basis functions, it is often advantageous to use a phase variable s instead. The phase variable provides a normalized representation of progress through the trajectory, independent of absolute time. The phase variable $s(t)$ is defined as:

$$s(t) = \frac{t}{T}, \tag{5.8}$$

where T is the total duration of the trajectory. This normalization maps $t \in [0, T]$ to $s \in [0, 1]$.

The basis functions are then parameterized by s rather than t:

$$\phi_i(s) = \exp\left(-\frac{(s - c_i)^2}{2h_i^2}\right), \tag{5.9}$$

$$\dot{\phi}_i(s) = \dot{s}(t)\left(-\frac{s - c_i}{h_i^2}\phi_i(s)\right), \tag{5.10}$$

where $\dot{s}(t) = \frac{1}{T}$ is the time derivative of the phase variable.

Using the phase variable allows for time-scaling invariance, which is particularly useful when trajectories of varying durations need to be compared or reproduced. The trajectory representation is then expressed as:

$$\mathbf{y}(s) = \mathbf{\Phi}(s)\mathbf{w} + \boldsymbol{\epsilon}, \tag{5.11}$$

where $\mathbf{\Phi}(s)$ replaces $\mathbf{\Phi}(t)$, and the structure of $\mathbf{\Phi}(s)$ remains consistent with the definition above.

5.1.2 Marginalizing Over Weights to Compute Likelihoods

Let's assume now that we observe trajectory $\mathbf{y}(t)$. Assuming model (5.7), its likelihood at each time t is computed by marginalizing over the weights:

$$p(\mathbf{y}(t)) = \int p(\mathbf{y}(t)|\mathbf{w})p(\mathbf{w})d\mathbf{w}. \tag{5.12}$$

This integral can be solved efficiently due to the Gaussian assumptions, yielding:

$$p(\mathbf{y}(t)) = \mathcal{N}(\mathbf{y}(t); \mathbf{\Phi}(t)\boldsymbol{\mu}_w, \mathbf{\Phi}(t)\mathbf{\Sigma}_w\mathbf{\Phi}(t)^\top + \mathbf{\Sigma}_\epsilon), \tag{5.13}$$

where $\mathbf{\Phi}(t)$ is a matrix of basis functions evaluated at all time steps. To transition from Eq. (5.12) to Eq. (5.13), we exploit the fact that both $p(\mathbf{y}(t)|\mathbf{w})$ and $p(\mathbf{w})$ are Gaussian distributions, which makes the integral tractable. The likelihood of observing the trajectory $\mathbf{y}(t)$ given the weights $\mathbf{w}$ is Gaussian, expressed as:

$$p(\mathbf{y}(t)|\mathbf{w}) = \mathcal{N}(\mathbf{y}(t); \mathbf{\Phi}(t)\mathbf{w}, \mathbf{\Sigma}_\epsilon), \tag{5.14}$$

where $\mathbf{\Phi}(t)$ is the matrix of basis functions evaluated at all time steps, $\mathbf{\Sigma}_\epsilon$ is the covariance matrix representing independent Gaussian noise, and $\mathbf{\Phi}(t)\mathbf{w}$ represents the mean of this conditional Gaussian. Similarly, the prior distribution over weights is also Gaussian:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \mathbf{\Sigma}_w), \tag{5.15}$$

where $\boldsymbol{\mu}_w$ and $\boldsymbol{\Sigma}_w$ are the mean and covariance of the weights, respectively.

Substituting these into Eq. (5.12), the marginalization integral becomes:

$$p(\mathbf{y}(t)) = \int \mathcal{N}(\mathbf{y}(t); \boldsymbol{\Phi}(t)\mathbf{w}, \boldsymbol{\Sigma}_\epsilon)\mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)d\mathbf{w}. \tag{5.16}$$

The product of these two Gaussian distributions yields another Gaussian distribution. As explained below (see Section 5.1.3), the marginal distribution of $\mathbf{y}(t)$ has a mean given by $\boldsymbol{\Phi}(t)\boldsymbol{\mu}_w$ and a covariance given by $\boldsymbol{\Phi}(t)\boldsymbol{\Sigma}_w\boldsymbol{\Phi}(t)^\top + \boldsymbol{\Sigma}_\epsilon$. Consequently, the integral simplifies directly to:

$$p(\mathbf{y}(t)) = \mathcal{N}(\mathbf{y}(t); \boldsymbol{\Phi}(t)\boldsymbol{\mu}_w, \boldsymbol{\Phi}(t)\boldsymbol{\Sigma}_w\boldsymbol{\Phi}(t)^\top + \boldsymbol{\Sigma}_\epsilon). \tag{5.17}$$

This result leverages the properties of Gaussian distributions, specifically their closed-form solutions for convolution (marginalization in this case) and combination. By carefully handling the mean and covariance terms, we arrive at the desired probabilistic description of $\mathbf{y}(t)$ in Eq. (5.17).

The above formulation highlights the elegance of ProMPs in combining probabilistic reasoning with basis function representation. Each basis function contributes to the overall trajectory generation, while the covariance $\boldsymbol{\Sigma}_w$ captures the uncertainty and variability in motion.

5.1.3 Product of Gaussian Distributions

The integral in Eq. (5.12) involves the product of two Gaussian distributions:

$$p(\mathbf{y}(t)) = \int \mathcal{N}(\mathbf{y}(t); \boldsymbol{\Phi}(t)\mathbf{w}, \boldsymbol{\Sigma}_\epsilon)\mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)d\mathbf{w}. \tag{5.18}$$

To compute this integral, we use the fact that the product of two Gaussian distributions yields another Gaussian distribution, scaled by a normalization factor. This enables the integral to be solved in closed form.

The conditional likelihood $p(\mathbf{y}(t)|\mathbf{w})$ is given as:

$$p(\mathbf{y}(t)|\mathbf{w}) = \mathcal{N}(\mathbf{y}(t); \boldsymbol{\Phi}(t)\mathbf{w}, \boldsymbol{\Sigma}_\epsilon), \tag{5.19}$$

while the prior on weights is given by:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w). \tag{5.20}$$

The product of these distributions is:

$$p(\mathbf{y}(t), \mathbf{w}) = \mathcal{N}(\mathbf{y}(t); \boldsymbol{\Phi}(t)\mathbf{w}, \boldsymbol{\Sigma}_\epsilon)\mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w). \tag{5.21}$$

To compute $p(\mathbf{y}(t))$, we marginalize over $\mathbf{w}$:

$$p(\mathbf{y}(t)) = \int p(\mathbf{y}(t), \mathbf{w}) d\mathbf{w}. \tag{5.22}$$

Substituting the product form:

$$p(\mathbf{y}(t)) = \int \mathcal{N}(\mathbf{y}(t); \mathbf{\Phi}(t)\mathbf{w}, \mathbf{\Sigma}_\epsilon)\mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \mathbf{\Sigma}_w) d\mathbf{w}. \tag{5.23}$$

This integral can be evaluated directly because the product of Gaussians is Gaussian. Specifically, the marginal distribution of $\mathbf{y}(t)$ is another Gaussian with:

- Mean: $\mathbf{\Phi}(t)\boldsymbol{\mu}_w$,
- Covariance: $\mathbf{\Phi}(t)\mathbf{\Sigma}_w\mathbf{\Phi}(t)^\top + \mathbf{\Sigma}_\epsilon$.

Eq. (5.17) is then obtained using this mean and covariance.

This result captures both the mean and variance of the observed trajectory $\mathbf{y}(t)$, combining contributions from the weight distribution and the noise model. The term $\mathbf{\Phi}(t)\boldsymbol{\mu}_w$ represents the mean trajectory, while $\mathbf{\Phi}(t)\mathbf{\Sigma}_w\mathbf{\Phi}(t)^\top + \mathbf{\Sigma}_\epsilon$ encapsulates the variability due to weight uncertainty and measurement noise.

5.2 Learning the Parameters of the Weight Distribution

In general, ProMPs are learned from multiple demonstrations. The diversity of demonstrations should reveal the variance of the task and the uncertainty of the execution.

The parameters of the weight distribution, $\boldsymbol{\mu}_w$ and $\mathbf{\Sigma}_w$, are learned from a set of N demonstrated trajectories $\{\mathbf{y}_i\}_{i=1}^N$. Each trajectory $\mathbf{y}_i(t)$, including both position and velocity, is represented as a set of observations $\{\mathbf{y}_i(t_j)\}_{j=1}^{T_i}$ over discrete time steps $t_1, t_2, \ldots, t_{T_i}$. Each observation $\mathbf{y}_i(t_j)$ is defined as:

$$\mathbf{y}_i(t_j) = \begin{bmatrix} y_i(t_j) \\ \dot{y}_i(t_j) \end{bmatrix}, \tag{5.24}$$

where $y_i(t_j)$ and $\dot{y}_i(t_j)$ are the position and velocity at time t_j, respectively. Note that the durations of the demonstrations, T_i, are not necessarily constant.

For each demonstration, $\mathbf{y}_i(t)$ is vectorized into $\mathbf{Y}_i$:

$$\mathbf{Y}_i = \begin{bmatrix} \mathbf{y}_i(t_1) \\ \mathbf{y}_i(t_2) \\ \vdots \\ \mathbf{y}_i(t_{T_i}) \end{bmatrix}, \tag{5.25}$$

where $\mathbf{Y}_i \in \mathbb{R}^{2T_i}$. This vectorization enables a consistent formulation for parameter estimation, incorporating both position and velocity at each time step.

The basis function matrix $\mathbf{\Phi}(t)$ incorporates both position and velocity contributions. Its $2 \times n$ structure is defined as:

$$\mathbf{\Phi}(t) = \begin{bmatrix} \phi_1(t) & \phi_2(t) & \dots & \phi_n(t) \\ \dot{\phi}_1(t) & \dot{\phi}_2(t) & \dots & \dot{\phi}_n(t) \end{bmatrix}, \tag{5.26}$$

where $\phi_i(t)$ is the i-th Gaussian basis function evaluated at time t and $\dot{\phi}_i(t)$ is its time derivative. To concatenate these basis functions over all time steps of a demonstration, the full concatenated matrix $\mathbf{\Phi}$ is constructed as:

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{\Phi}(t_1) \\ \mathbf{\Phi}(t_2) \\ \vdots \\ \mathbf{\Phi}(t_{T_i}) \end{bmatrix}, \tag{5.27}$$

where $\mathbf{\Phi} \in \mathbb{R}^{2T_i \times n}$ represents the basis function contributions for all positions and velocities across the entire demonstration.

To estimate $\boldsymbol{\mu}_w$ and $\mathbf{\Sigma}_w$, we first compute the weights for each demonstration by solving the regularized least-squares problem:

$$\mathbf{w}_i = \arg\min_{\mathbf{w}} \sum_{j=1}^{T_i} \|\mathbf{y}_i(t_j) - \mathbf{\Phi}(t_j)\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2, \tag{5.28}$$

where λ is a regularization parameter. The closed-form solution is given by:

$$\mathbf{w}_i = (\mathbf{\Phi}^\top \mathbf{\Phi} + \lambda \mathbf{I})^{-1} \mathbf{\Phi}^\top \mathbf{Y}_i, \tag{5.29}$$

where $\mathbf{\Phi}$ is the concatenated matrix of basis functions evaluated at all time steps, and $\mathbf{Y}_i$ is the vectorized trajectory for the i-th demonstration, including both position and velocity.

During this process, it is often necessary to normalize the trajectories to ensure a consistent time scale across demonstrations. This normalization adjusts each demonstration to a common duration T_{norm} using a mapping from the original time scale $[0, T_i]$ to the normalized time scale $[0, T_{\text{norm}}]$. The normalized time parameter $\tau(t)$ is defined as:

$$\tau(t) = \frac{t}{T_i} T_{\text{norm}}, \tag{5.30}$$

where $t \in [0, T_i]$ represents the original time scale. The normalized basis functions and their derivatives are then computed using $\tau(t)$ to ensure compatibility across varying durations.

Once the weights $\{\mathbf{w}_i\}_{i=1}^N$ are computed, the mean and covariance are estimated as:

$$\boldsymbol{\mu}_w = \frac{1}{N}\sum_{i=1}^{N}\mathbf{w}_i, \tag{5.31}$$

$$\boldsymbol{\Sigma}_w = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}_i - \boldsymbol{\mu}_w)(\mathbf{w}_i - \boldsymbol{\mu}_w)^\top. \tag{5.32}$$

5.2.1 Motion Reproduction

During reproduction, a new trajectory is generated by sampling weights from the learned distribution:

$$\mathbf{w}_{\text{sampled}} \sim \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w). \tag{5.33}$$

The trajectory is then computed as:

$$\mathbf{y}(t) = \boldsymbol{\Phi}(t)\mathbf{w}_{\text{sampled}}, \tag{5.34}$$

where $\boldsymbol{\Phi}(t)$ accounts for both position and velocity contributions.

Fig. 5.1 shows how the ProMP model captures and reproduces motion trajectories based on the training data. The training data highlights the variability across multiple demonstrations, showing the inherent differences in the motion. The reproduced ProMPs illustrate the model's ability to learn the central tendency of the motion, depicted as the mean trajectory, while simultaneously encoding the variability through the 1-sigma interval. This probabilistic representation not only captures the typical motion pattern

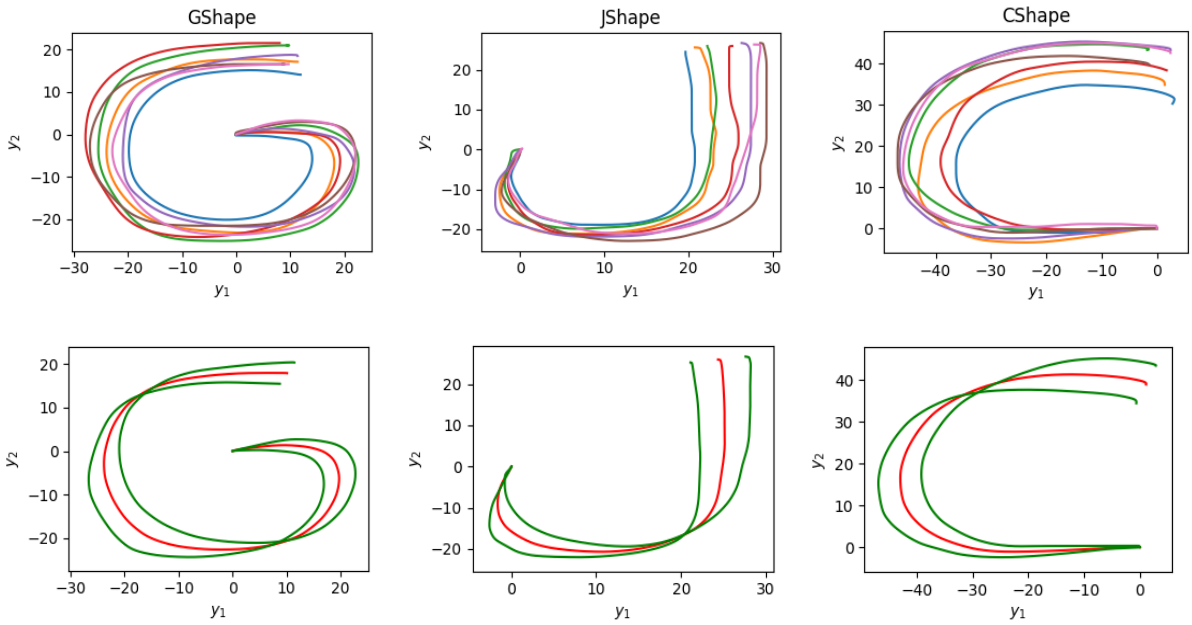

Figure 5.1: The upper row graphs show the data for training a ProMP and the lower row the reproduced ProMPs (red curves) with 1-sigma interval (green curves).

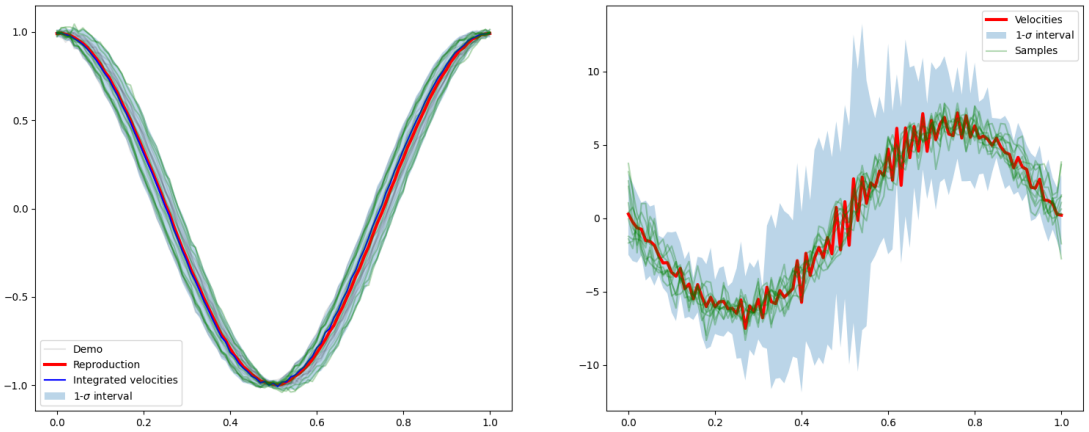

Figure 5.2: The left graph shows the position obtained by integrating mean velocity of a ProMP and the variance of training data. The right graph shows the mean velocity and its variance computed by the ProMP.

but also provides a clear understanding of the uncertainty, showcasing the model's strength in generalizing to similar trajectories while maintaining robustness to variability in the data. The graphs confirm that the ProMP implementation accurately reflects the desired probabilistic properties of the motion.

ProMPs are designed to encode and reproduce not only position trajectories but also velocity profiles, ensuring a comprehensive representation of motion dynamics, as shown in Figure 5.2. By capturing both the mean velocity and its variability, ProMPs enable robots to replicate the nuances of demonstrated motions with greater accuracy. This ability to model velocity alongside position is crucial for tasks that require precise dynamic execution, such as manipulation or interaction in variable environments, where both spatial and temporal aspects of motion are essential for success. This and the previous example were generated using the code from [7].

To ensure that the reproduced trajectory conforms to the desired duration $T_{\text{reproduction}}$, we align the time scaling with the normalized duration T_{norm}. The time parameter t for reproduction is scaled as:

$$t = \tau^{-1}(\tau(t)), \tag{5.35}$$

where $\tau(t) = \frac{t}{T_{\text{reproduction}}} T_{\text{norm}}$ is the normalized time scale. The basis functions and their derivatives are then evaluated as:

$$\phi_i(\tau(t)) = \exp\left(-\frac{(\tau(t) - c_i)^2}{2h_i^2}\right), \tag{5.36}$$

$$\dot{\phi}_i(\tau(t)) = \dot{\tau}(t)\left(-\frac{\tau(t) - c_i}{h_i^2}\phi_i(\tau(t))\right), \tag{5.37}$$

where $\dot{\tau}(t) = \frac{T_{\text{norm}}}{T_{\text{reproduction}}}$ is the scaling factor for the time derivative. This ensures that the reproduced trajectory adheres to the desired temporal constraints while retaining the learned variability and central tendencies of the demonstrations.

5.3 Ensuring Specific Configuration at a Given Time

In robotic applications, it is often required for the trajectory to achieve a specific configuration $\mathbf{y}_g$ (e.g., a goal or intermediate state) at a given time t_g. ProMPs enable this by conditioning the weight distribution on the desired configuration.

5.3.1 Conditioning on a Specific Configuration

To enforce a specific configuration $\mathbf{y}_g$ at time t_g, we impose a probabilistic constraint within the ProMP framework. The conditioned distribution over the weights $\mathbf{w}$ is computed as (see also Section 5.3.4):

$$p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g) \propto p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w})p(\mathbf{w}). \tag{5.38}$$

The likelihood of the desired configuration given the weights is Gaussian:

$$p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w}) = \mathcal{N}(\mathbf{y}_g; \mathbf{\Phi}(t_g)\mathbf{w}, \mathbf{\Sigma}_\epsilon), \tag{5.39}$$

where $\mathbf{\Phi}(t_g)$ is the basis function matrix evaluated at t_g, and $\mathbf{\Sigma}_\epsilon$ represents the observation noise.

Combining this with the prior distribution over weights, $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \mathbf{\Sigma}_w)$, the conditioned distribution $p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g)$ is Gaussian with:

$$\boldsymbol{\mu}_w^{\text{cond}} = \boldsymbol{\mu}_w + \mathbf{G}(\mathbf{y}_g - \mathbf{\Phi}(t_g)\boldsymbol{\mu}_w), \tag{5.40}$$

$$\mathbf{\Sigma}_w^{\text{cond}} = \mathbf{\Sigma}_w - \mathbf{G}\mathbf{\Phi}(t_g)\mathbf{\Sigma}_w, \tag{5.41}$$

where the gain matrix $\mathbf{G}$ is defined as:

$$\mathbf{G} = \mathbf{\Sigma}_w\mathbf{\Phi}(t_g)^\top(\mathbf{\Phi}(t_g)\mathbf{\Sigma}_w\mathbf{\Phi}(t_g)^\top + \mathbf{\Sigma}_\epsilon)^{-1}. \tag{5.42}$$

5.3.2 Detailed Derivation of Conditioned Mean and Covariance

The conditioning process in ProMPs leverages properties of multivariate Gaussian distributions. Given a prior Gaussian distribution over weights:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \mathbf{\Sigma}_w), \tag{5.43}$$

and a probabilistic observation model:

$$p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w}) = \mathcal{N}(\mathbf{y}_g; \mathbf{\Phi}(t_g)\mathbf{w}, \mathbf{\Sigma}_\epsilon), \tag{5.44}$$

the posterior distribution $p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g)$ can be derived using the Gaussian conditioning formula.

To compute the posterior distribution $p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g)$, we use the fact that the product of two Gaussian distributions is another Gaussian, up to a normalization factor. Specifically:

$$p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g) \propto p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w})p(\mathbf{w}). \tag{5.45}$$

Substituting the Gaussian forms:

$$p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w}) = \mathcal{N}(\mathbf{y}_g; \mathbf{\Phi}(t_g)\mathbf{w}, \mathbf{\Sigma}_\epsilon), \tag{5.46}$$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w, \mathbf{\Sigma}_w), \tag{5.47}$$

the product of these distributions yields:

$$p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_w^{\text{cond}}, \mathbf{\Sigma}_w^{\text{cond}}), \tag{5.48}$$

where the conditioned mean and covariance are given by:

$$\boldsymbol{\mu}_w^{\text{cond}} = \boldsymbol{\mu}_w + \mathbf{G}(\mathbf{y}_g - \mathbf{\Phi}(t_g)\boldsymbol{\mu}_w), \tag{5.49}$$

$$\mathbf{\Sigma}_w^{\text{cond}} = \mathbf{\Sigma}_w - \mathbf{G}\mathbf{\Phi}(t_g)\mathbf{\Sigma}_w. \tag{5.50}$$

The conditioned mean $\boldsymbol{\mu}_w^{\text{cond}}$ is a combination of the prior mean $\boldsymbol{\mu}_w$ and a correction term. This correction term adjusts the prior mean based on the discrepancy between the desired configuration $\mathbf{y}_g$ and the predicted configuration $\mathbf{\Phi}(t_g)\boldsymbol{\mu}_w$, weighted by the uncertainties in the prior and observation models. The conditioned covariance $\mathbf{\Sigma}_w^{\text{cond}}$ is the prior covariance $\mathbf{\Sigma}_w$ reduced by a term proportional to the information gain from the observation at t_g. This reflects the fact that the uncertainty in $\mathbf{w}$ decreases as more constraints are added.

5.3.3 Reproducing the Conditioned Trajectory

Once the conditioned distribution $p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g)$ is computed, a new trajectory can be generated. A sample of weights $\mathbf{w}_{\text{sampled}}$ is drawn from the conditioned distribution:

$$\mathbf{w}_{\text{sampled}} \sim \mathcal{N}(\boldsymbol{\mu}_w^{\text{cond}}, \mathbf{\Sigma}_w^{\text{cond}}). \tag{5.51}$$

The trajectory is then computed as:

$$\mathbf{y}(t) = \mathbf{\Phi}(t)\mathbf{w}_{\text{sampled}}, \tag{5.52}$$

where the reproduced trajectory respects the specified configuration $\mathbf{y}_g$ at time t_g while maintaining the natural variability of the learned ProMP.

Fig. 5.3 shows trajectory generation with ProMPs given different initial conditions for the beginning of motion. On the left, the training data consists of multiple trajectory demonstrations highlighting the variability in execution. The demonstrations exhibit

significant variance, particularly in the middle and end segments of the trajectories, illustrating the inherent variability in human or robot motion. On the right, the generated trajectories represent the mean trajectories computed by conditioning ProMPs on the specified initial points.

ProMPs enable such conditioning by leveraging their probabilistic nature, where the underlying distribution of trajectory weights is updated based on the given initial conditions. This process ensures that the generated trajectories not only start from the specified points but also conform to the variability and patterns observed in the training data. The conditioned trajectories smoothly transition from the specified initial states, reflecting the probabilistic correlations encoded in the learned model.

The ability to condition trajectories makes ProMPs particularly useful for tasks requiring adaptability to changes in the starting configuration. For example, in applications like pick-and-place operations, where the robot's initial position might vary due to dynamic workspace constraints, ProMPs can generate appropriate motion plans that adhere to the statistical properties of the demonstrated trajectories. This adaptability is achieved without the need for re-learning or extensive recalibration, which is a significant advantage over deterministic approaches.

Moreover, the right-hand plot demonstrates the smoothness and consistency of the conditioned trajectories, ensuring that the robot's motion remains natural and feasible. This smooth adaptation is crucial in collaborative scenarios, such as human-robot interaction, where the robot must seamlessly adjust its actions based on the human partner's behavior. The probabilistic framework of ProMPs enables such adaptability while preserving the natural variability and robustness of the learned motion.

Overall, Fig. 5.3 illustrates the core strengths of ProMPs: their ability to capture

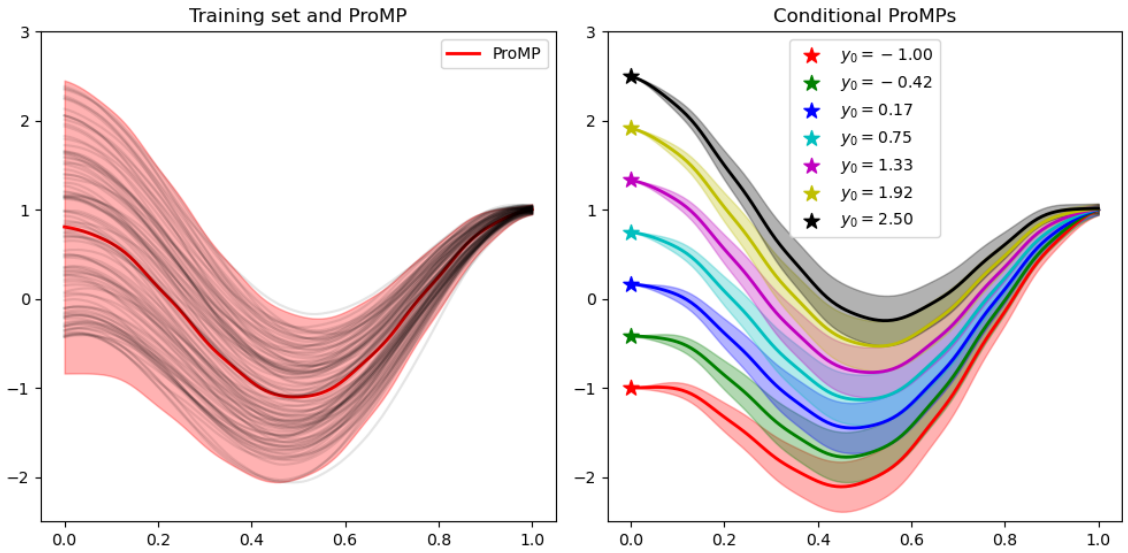

Figure 5.3: The left graph shows the training data, the mean trajectory and the variance in the training data. The right graph shows the computed mean trajectories given the initial condition and the associated variances.

the variability of demonstrated motions, adapt to task-specific constraints, and generate trajectories that balance naturalness and task requirements. This capability underscores ProMPs' suitability for dynamic and uncertain environments where flexibility and robustness are paramount.

5.3.4 Bayesian Basis for Conditioning

The conditioning operation described above is rooted in Bayes' theorem. Bayes' theorem states that:

$$p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g) \propto p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w})p(\mathbf{w}). \tag{5.53}$$

Here, $p(\mathbf{w})$ is the prior distribution over the weights, representing what has been learned from demonstrations, and $p(\mathbf{y}(t_g) = \mathbf{y}_g|\mathbf{w})$ is the likelihood of the desired configuration $\mathbf{y}_g$ given these weights. The posterior distribution $p(\mathbf{w}|\mathbf{y}(t_g) = \mathbf{y}_g)$ combines these terms to yield a new distribution over weights that respects both the prior knowledge and the imposed constraint. This Bayesian formulation ensures consistency between the learned ProMP and the specified configuration, allowing the system to balance prior variability with task-specific requirements.

5.4 Summary and Discussion

Probabilistic Movement Primitives (ProMPs) offer a versatile and robust framework for learning, modeling, and reproducing motion trajectories in robotics. By employing a probabilistic representation, ProMPs capture the variability and uncertainty inherent in motion demonstrations, making them particularly suitable for tasks requiring adaptability and multimodal trajectory representations. The foundational work by Paraschos et al. [24, 25] laid the theoretical groundwork for ProMPs, introducing a probabilistic extension of traditional movement primitives that enables both deterministic reproduction of mean trajectories and stochastic sampling for diverse trajectory generation.

Compared to other frameworks such as Dynamic Movement Primitives (DMPs) and the Stable Estimator of Dynamical Systems (SEDS), ProMPs excel in capturing probabilistic variations and ensuring adaptability. DMPs combine accurate trajectory reproduction with attractor dynamics, ensuring convergence to a desired goal even in the presence of perturbations, but rely on fixed attractor points and deterministic formulations. SEDS emphasizes stability and robustness through Lyapunov functions, guaranteeing global convergence to attractors but often lacks explicit modeling of variability. ProMPs, on the other hand, prioritize capturing the probabilistic nature of motion, making them ideal for tasks requiring flexibility and the ability to encode multimodal trajectory distributions. However, ProMPs do not inherently ensure stability, which might require additional constraints or mechanisms in specific applications.

The probabilistic framework of ProMPs leverages a Gaussian weight distribution to encode variability observed in demonstrations. Using a weighted combination of basis

functions parameterized by either time or a phase variable, ProMPs provide a robust mechanism for modeling trajectories of varying durations while maintaining consistency. Learning involves estimating the mean and covariance of the weight distribution from multiple demonstrations, capturing both the central tendency and variability in the observed data. The use of Bayesian conditioning enables task-specific constraints, such as achieving specific configurations, while preserving the natural variability of motion.

ProMPs have demonstrated significant potential in human-robot interaction and collaborative tasks, as highlighted in the works of Maeda et al. [21, 20]. These extensions, referred to as interaction primitives, leverage the probabilistic framework to model and predict the motion of interacting agents. ProMPs have been successfully applied to coordinate multiple human-robot tasks, enabling robots to anticipate human actions and adjust their behavior in real-time. Similarly, they have shown promise in robot-directed imitation learning tasks [19], reproducing complex behaviors demonstrated by humans and generalizing from limited examples.

The choice between ProMPs, DMPs, and SEDS depends on the specific requirements of the task. For applications demanding deterministic and stable motion generation, DMPs or SEDS may be preferred. However, for tasks requiring adaptability, probabilistic reasoning, and the ability to model multimodal distributions, ProMPs provide a compelling solution. Furthermore, ProMPs' extension to interaction primitives highlights their versatility in multi-agent and collaborative scenarios, enabling coordinated motion and real-time adaptability.

In summary, ProMPs provide a powerful tool for modern robotic motion generation, combining flexibility, adaptability, and the ability to represent probabilistic variations. Their applicability extends to various robotics tasks, including human-robot collaboration, manipulation, and complex task planning, making them a cornerstone in robotic motion representation methodologies.

Chapter 6

Motion Representations and Robot Learning

Robot motion representation stands at the core of both control and learning in robotics. In conclusion, we reflect on the major representations discussed in this textbook – splines, Dynamic Movement Primitives (DMPs), Stable Estimator of Dynamical Systems (SEDS), and Probabilistic Movement Primitives (ProMPs) – and examine their significance in imitation learning, robot learning, especially in reinforcement learning (RL) and deep reinforcement learning (DRL).

Modern robotics demands flexible, adaptive, and robust representations that can accommodate the diverse requirements of robot motion. Polynomials and splines, as discussed, form the foundation of trajectory planning. Their ability to model smooth, continuous trajectories with explicit control over derivatives makes them an excellent tool for precision tasks like pick-and-place operations or industrial machining. However, their static nature poses challenges in dynamic and uncertain environments. While spline-based representations excel in predefined settings, they lack the adaptability required for tasks that involve external perturbations or unstructured interactions. The advancements in robot control and machine learning have necessitated motion representations that go beyond such static paradigms.

DMPs emerged to address challenges in robot motion generation by introducing a dynamic, goal-oriented framework that encodes trajectories as solutions to differential equations with attractor dynamics. This ensures robust convergence to a desired goal, making DMPs ideal for tasks requiring resilience to disturbances or variations in starting and end conditions. Their simplicity and scalability to high-dimensional tasks have solidified their role in robotic control. However, their deterministic formulation, while advantageous for precise goal-oriented applications, limits their capacity to represent and generalize the variability inherent in real-world demonstrations.

To address this limitation, extensions to DMPs have integrated statistical learning techniques, enabling them to capture trajectory variability and generalize across the training data. Using statistical learning techniques such as locally weighted regression, DMPs

can interpolate or extrapolate to generate trajectories for unseen queries, thus adapting to novel configurations or task conditions. This approach also supports probabilistic reasoning, allowing robots to account for uncertainty in trajectory execution. Such capabilities are particularly beneficial in dynamic environments, including human-robot collaboration, where DMPs enhanced by statistical learning can adaptively and robustly respond to variability in human actions or task requirements, bridging classical control with modern data-driven frameworks [2, 17].

SEDS offers an alternative by emphasizing global stability through Lyapunov functions. This framework is rooted in the mathematical rigor of stability theory. They ensure convergence to attractors across a wide range of initial conditions. SEDS is particularly well-suited for tasks where safety and reliability are paramount, such as human-robot interaction or autonomous navigation in cluttered environments. The use of Gaussian Mixture Models (GMMs) enables SEDS to capture the variability of demonstrated motions while maintaining stability. However, the trade-off between stability and motion reproduction fidelity can constrain its use in tasks requiring precise trajectory matching. Furthermore, the computational complexity associated with ensuring Lyapunov stability can be a limiting factor in high-dimensional settings.

ProMPs represent a significant evolution in motion representation, prioritizing the probabilistic nature of trajectories. By modeling motion as a distribution over trajectories, ProMPs capture both the central tendency and the variability of demonstrations. This capability makes them particularly suitable for tasks requiring adaptability, such as human-robot collaboration or manipulation under uncertainty. The use of Bayesian inference allows ProMPs to incorporate task-specific constraints seamlessly, enabling applications in dynamic and multimodal environments. Their flexibility and probabilistic foundation position them as a versatile tool in modern robotics.

The broader context of robot control and learning places these representations within the frameworks of reinforcement learning (RL) and deep reinforcement learning (DRL). RL, with its emphasis on optimizing policies through trial-and-error interactions, benefits significantly from representations like DMPs and ProMPs. DMPs provide a structured way to initialize RL policies, leveraging their dynamic attractor properties to explore goal-oriented behaviors effectively. ProMPs, on the other hand, enable RL agents to model uncertainty and adapt policies to stochastic environments. Their probabilistic nature aligns well with the exploration-exploitation trade-offs inherent in RL. For example, the structured representations provided by ProMPs can reduce the sample complexity of learning policies in tasks requiring generalization across varying conditions [28].

In DRL, where the focus shifts to high-dimensional state and action spaces, motion representations play an even more crucial role. DMPs and SEDS, with their ability to encode complex trajectories and ensure stability, respectively, serve as valuable tools for embedding domain knowledge into neural network architectures. Moreover, incorporating ProMP-based priors into DRL models can accelerate convergence by guiding policy exploration towards regions of the state space with high demonstration density. Similarly,

SEDS-inspired stability constraints can enhance the robustness of DRL policies, ensuring safe interactions in physical environments [18, 10].

Furthermore, motion representations can serve as a bridge between model-based and model-free approaches in RL. While model-free methods rely on data-driven exploration, model-based methods benefit from the structured representations offered by DMPs, SEDS, and ProMPs. These representations provide a way to encode prior knowledge about dynamics, reducing the sample complexity of learning. For example, in robot manipulation tasks, ProMPs can represent a family of feasible trajectories, allowing RL algorithms to focus on fine-tuning policies rather than discovering them from scratch.

The interplay between motion representations and learning extends to imitation learning and policy transfer. Techniques like guided policy search and behavior cloning rely on theoretically sound motion representations to encode demonstration data. The ability to transfer knowledge across tasks and environments depends on the expressiveness and adaptability of the chosen representation.

In conclusion, the trajectory representations discussed in this textbook offer a wide spectrum of tools for addressing the diverse challenges of robot motion. From the precision of splines to the adaptability of ProMPs, each framework contributes unique strengths to the broader landscape of robotics. The integration of these representations with imitation learning, reinforcement learning, and deep learning paradigms underscores their relevance in advancing robot autonomy. As robotics continues to evolve, the synergy between control, learning, and motion representation will play a pivotal role in shaping the capabilities of future robotic systems.

Bibliography

[1] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *AI Review*, 11:11–73, 1997.

[2] S. Chernova and A. Thomaz. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121, 2014.

[3] S. Chiaverini and B. Siciliano. The unit quaternion: A useful tool for inverse kinematics of robot manipulators. *System Analysis, Modelling and Simulation*, 35:45–60, 1999.

[4] A. Coulombe and H.-C. Lin. Generating stable and collision-free policies through Lyapunov function learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3037–3043, London, UK, 2023.

[5] C. de Boor. *A Practical Guide to Splines; Revised Edition.* Applied Mathematical Sciences 27. Springer, New York, 2001.

[6] R. L. Eubank. *Spline Smoothing and Nonparametric Regression.* Marcel Dekker, New York, 1988.

[7] A. Fabisch. movement_primitives: Imitation learning of cartesian motion with movement primitives. *Journal of Open Source Software*, 9(97):6695, 2024.

[8] A. Gams, A. J. Ijspeert, S. Schaal, and J. Lenarčič. On-line learning and modulation of periodic movements with nonlinear dynamical systems. *Autonomous Robots*, 27(1):3–23, 2009.

[9] E. Gribovskaya, S. M. Khansari-Zadeh, and A. Billard. Learning non-linear multivariate dynamics of motion in robotic manipulators. *The International Journal of Robotics Research*, 30(1):80–117, 2011.

[10] A. Gupta, K. Zhang, and H.-C. Lin. Stability-aware reinforcement learning for robotic systems. *IEEE Transactions on Robotics*, 39(3):567–589, 2023.

[11] C. L. Hu and L. L. Schumaker. Univariate complete smoothing. *Numerische Mathematik*, 49(1):1–10, 1986.

[12] 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.

[13] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning rhythmic movements by demonstration using nonlinear oscillators. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 958–963, Lausanne, Switzerland, 2002.

[14] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1398–1403, Washington, DC, 2002.

[15] S. M. Khansari-Zadeh and A. Billard. Learning stable non-linear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[16] S. M. Khansari-Zadeh and A. Billard. Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765, 2014.

[17] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[18] P. Kormushev, S. Calinon, and D. G. Caldwell. Learning adaptive motor skills: A survey of techniques for robotics. *Advanced Robotics*, 34(9):555–573, 2020.

[19] G. J. Maeda, M. Ewerton, M. Goldhoorn, J. Krüger, and J. Peters. Probabilistic movement primitives for robot-directed imitation learning. *Autonomous Robots*, 41(3):593–612, 2017.

[20] G. J. Maeda, M. Ewerton, D. Koert, and J. Peters. Probabilistic movement primitives for coordination of multiple human-robot collaborative tasks. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 276–284, 2017.

[21] G. J. Maeda, M. Ewerton, and J. Peters. Learning interaction for collaborative tasks with probabilistic movement primitives. In *Robotics: Science and Systems (RSS)*, 2016.

[22] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, New York, 1994.

[23] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local Gaussian process regression. *Advanced Robotics*, 23:2015–2034, 2009.

[24] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2616–2624. Curran Associates, Inc., 2013.

[25] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):529–551, 2018.

[26] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. Probabilistic movement primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 365–371, San Francisco, California, 2013.

[27] T. Petrič, A. Gams, A. J. Ijspeert, and L. Žlajpah. On-line frequency adaptation and movement imitation for rhythmic robotic tasks. *The International Journal of Robotics Research*, 30(14):1775–1788, 2011.

[28] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):297–330, 2020.

[29] L. Righetti, J. Buchli, and A. J. Ijspeert. Dynamic Hebbian learning in adaptive frequency oscillators. *Physica D*, 216:269–281, 2006.

[30] L. Rozo, J. Silvério, S. Calinon, and D. G. Caldwell. Learning controllers for reactive and proactive behaviors in human-robot collaboration. *Frontiers in Robotics and AI*, 3:30, 2016.

[31] S. Schaal, P. Mohajerian, and A. Ijspeert. Dynamics systems vs. optimal control – a unifying view. *Progress in Brain Research*, 165(6):425–445, 2007.

[32] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In P. Dario and R. Chatila, editors, *Robotics Research: The Eleventh International Symposium*, pages 561–572, Berlin, Heidelberg, 2005. Springer.

[33] S. E. Thompson and R. V. Patel. Formulation of joint trajectories for industrial robots using B-splines. *IEEE Trans. Industrial Electronics*, 34(2):192–199, May 1987.

[34] A. Ude. Filtering in a unit quaternion space for model-based object tracking. *Robotics and Autonomous Systems*, 28(2-3):163–172, 1999.

[35] A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, 26(5):800–815, 2010.

[36] A. Ude, B. Nemec, T. Petrič, and J. Morimoto. Orientation in cartesian space dynamic movement primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004, Hong Kong, 2014.

[37] G. Wahba. *Spline Models for Observational Data*. SIAM, Philadelphia, 1990.

[38] J. Yuan. Closed-loop manipulator control using quaternion feedback. *IEEE Journal of Robotics and Automation*, 4(4):434 –440, 1988.