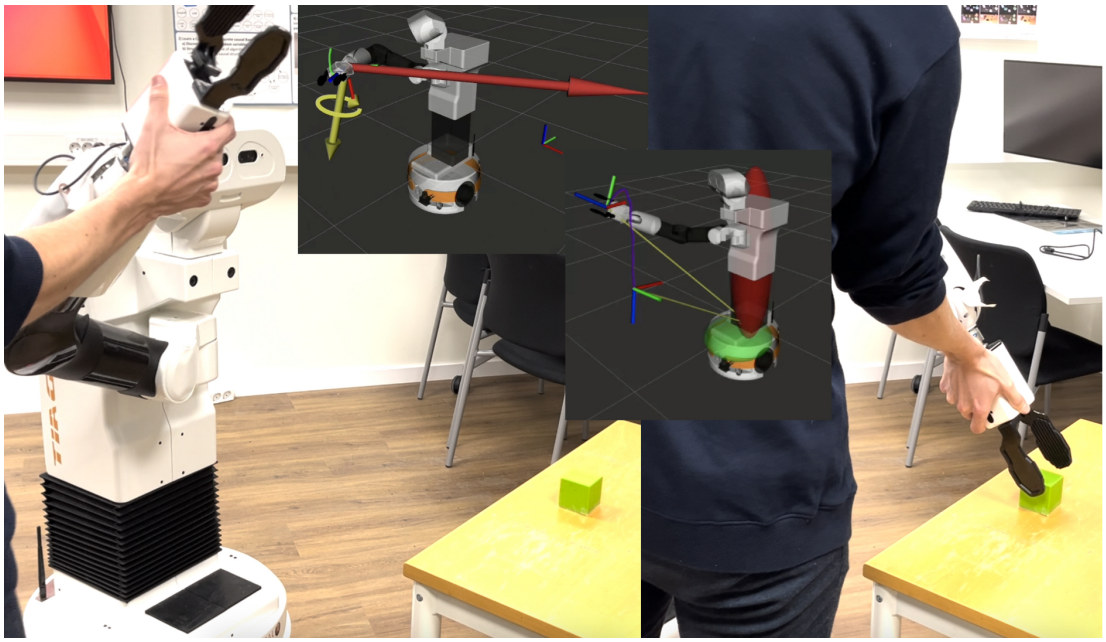
CHALMERS
UNIVERSITY OF TECHNOLOGY



From Learning to Execution: A General Interface for Kinesthetic Teaching in Robotics

Hierarchical Task Specification, Control, and Parallelizing Dynamical Movement Primitives

Master's thesis in Systems, Control and Mechatronics

HAMPUS AHLEBRAND

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

From Learning to Execution: A General Interface for Kinesthetic Teaching in Robotics

Hierarchical Task Specification, Control, and Parallelizing Dynamical Movement Primitives

HAMPUS AHLEBRAND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2024

From Learning to Execution: A General Interface for Kinesthetic Teaching in Robotics
HAMPUS AHLEBRAND

Supervisor: Dean, Emmanuel Department of Electrical Engineering
Examiner: Dean, Emmanuel, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

From Learning to Execution: A General Interface for Kinesthetic Teaching in Robotics
HAMPUS AHLEBRAND
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Robotics has the potential to solve many problems, such as relieving humans from repetitive and or hazardous work. Thus, robotics receives much research attention. Ideally, humans and robots seamlessly coexist and collaborate in different environments such as the workplace. However, programming robots with decision-making and control algorithms is difficult. One approach to this problem, when repetitive tasks are involved, is to design intuitive programming interfaces to allow fast reprogramming and thereby utilize human cognitive abilities to effectively increase the robot's adaptability.

This thesis aims to provide a general programming interface in the Learning from Demonstration paradigm. The main contribution of this work is the formulation of a general approach for allocating hierarchical tasks based on specific task requirements. These tasks can be defined in various spaces, including force, joint and Cartesian position, velocity, and others. The developed interface allows an operator to kinesthetically teach a task that is encoded and generalized using a set of Dynamical Movement Primitives (DMPs). Although DMPs are used in this work to encode learning, it is not required, in principle, any learning algorithm that can generate appropriate references can be used. To execute tasks, provide a hierarchical task specification from which a controller is automatically synthesized and DMPs prioritized. The method is experimentally validated.

The control algorithm uses a kinematic model of the robot which is shown to be sufficient for the teaching and execution of kinematic tasks. During kinesthetic teaching, reference signals are extracted from the robot's Force/Torque sensor which is noisy and contains effects due to gravity and end-effector motion. A filter was developed to filter out those effects, and it's compared with a custom-made filter. The filters were comparable and useable, future improvements to ours are discussed. An admittance algorithm (wrench to velocity) was developed to transform filtered wrench values to velocity references, it's suggested to implement it for acceleration level instead to improve Kinesthetic Teaching. Task execution was experimentally validated by comparing different task hierarchies for taught position and orientation tasks. An attempt to execute a dynamic task (wiping) in the simulation was performed and discussed. The thesis is accompanied by a video found at `https://youtu.be/t0Wgd-3bS7c`.

Keywords: Collaborative Robot, Dynamical Movement Primitive, Hierarchical Control, Nullspace Projection, Quadratic Programming

Acknowledgements

First and foremost, I would like to extend my gratitude to my supervisor, Emmanuel Dean, for his insights, encouragement, and patience throughout my research journey. His guidance has been important in shaping both the direction and success of my work.

I also wish to express my thanks to my fellow students. The years spent together have been not only a journey of academic growth but also one of mutual support and camaraderie. The discussions, collaborations, and shared experiences have been a vital part of my academic life.

Last, but not least, thanks to my friends and family for their love, support, and encouragement. They have been a source of strength and motivation, especially during the most challenging times.

To everyone who has been a part of this journey – thank you.

Hampus Ahlebrand, Gothenburg, December 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

APF	Artificial Potential Fields
Cobot	Collaborative Robot
CoM	Center of Mass
DoF	Degree Of Freedom
DMP	Dynamical Movement Primitive
FIR	Finite Impulse Response
F/T	Force/Torque
GMM	Gaussian Mixture Mode
GPM	Gradient Projection Method
HQP	Hierarchical Quadratic Programming
HRC	Human-Robot Collaboration
IIR	Infinite Impulse Response
KMP	Kernelized Movement Primitive
KT	Kinesthetic Teaching
LfD	Learning From Demonstration
MPC	Model Predictive Control
ProMP	Probabilistic Movement Primitive
QP	Quadratic Program
RQ	Research Question
ROS	Robot Operating System
TTF	Task Frame Formalism
VR	Virtual Reality

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i,j	Indices for vector and matrix elements
k	Index for time step
n_t	Number of tasks

Variables and Vectors

Unless specified differently, we represent vectors using bold lowercase letters or symbols, and matrices with bold uppercase letters or symbols. Scalars, in contrast, are denoted in a regular, non-bold typeface.

Δt	Time delta
$\mathbf{F}$	Force
$\mathbf{H}$	Quadratic term Weighting Matrix in a QP
$\mathbf{J}$	Jacobian Matrix
$\mathbf{P}$	Generalized Projection Matrix
$\mathbf{Q}$	Unit quaternion
$\mathbf{R}$	Rotation Matrix
$\mathbf{T}$	Torque
$\mathbf{W}$	Wrench
$\dot{\mathbf{q}}$	Joint state velocity
$\dot{\mathbf{x}}$	Cartesian Pose velocity
$\mathcal{F}$	Task Frame
$\mathbf{f}$	Linear Weighting Vector Term in a QP
$\mathbf{p}$	Cartesian position

$\mathbf{q}$	Joint state
$\mathbf{q}_p$	Physical Joint state
$\mathbf{q}_v$	Virtual Joint state
$\mathbf{q}'$	Augmented Joint state
$\mathbf{x}$	Cartesian Pose
$\mathbf{x_p}$	Physical Cartesian Pose
$\mathbf{x}_v$	Virtual Cartesian Pose
$\boldsymbol{\eta}$	Orientation State in the Quaternion Logarithm Space
$\boldsymbol{\xi}$	Task State
$\boldsymbol{\xi}_{ref}$	Reference Task State
$\overline{\mathbf{p}}$	Upper Boundary for the vector $\mathbf{p}$
$\underline{\mathbf{p}}$	Lower Boundary for the vector $\mathbf{p}$
g	Goal state
t	Time
τ	Desired Time duration
θ	Angle

Contents

List of Figures

List of Tables

1

Introduction

In professional and daily settings, many tasks are physically demanding, repetitive, or hazardous. Advancements in Robotics aim to mitigate the workload of such tasks. Robots have been successful in automating many tasks, particularly in manufacturing. According to the World Robotics 2023 report [1], there was a 5% annual increase in industrial robot installations globally in 2022, amounting to 553,052 units, with an estimated 590,000 units projected for 2023. The global count of operational robots reached a record high of around 3.5 million units, with the installations valued at approximately $15.7 billion. However, these are mostly traditional robots, that is, they typically function in structured settings within confined safety zones, separate from human interaction.

A robot designed for unstructured environments and human interaction is often referred to as a collaborative robot, or Cobot [2]. Cobots have different software and sensor requirements than traditional robots, for instance, they must be capable of reacting to, and sometimes predict the environment, such as human movements. Excessive forces must not be applied and collisions must be avoided. Increasingly, Cobots are employed in factories for tasks like assembly, pick and place, quality inspection, material handling, and logistics. They have been shown to enhance both flexibility and productivity, as well as reduce boredom and physical strain by assuming repetitive tasks [3]. As these Cobots become more sophisticated, it may become feasible for manufacturers to shift production lines from mass production to mass customization. Small and medium-sized enterprises might also find these robots beneficial [4, 5, 6].

Even though many Cobots are impressive, in general, humans surpass robots in flexibility and adaptability, while robots excel in precision, repeatability, and speed. Here, flexibility indicates the variety of tasks that can be performed without needing changes to hardware or source code. Adaptability corresponds to the space of environmental configurations in which a task can be executed, as well as the speed at which this space can expand (i.e., the learning rate). The challenge in enabling robots to work safely and efficiently alongside humans in unstructured environments arises from robots' limitations in aspects of flexibility and adaptability [7, 8].

Adaptability in robots can be enhanced by incorporating additional degrees of freedom. This enables the execution of tasks in multiple ways and the simultaneous tracking of several, potentially conflicting, tasks. However, this introduces the challenge of task prioritization and the translation of these priorities into control com-

mands. Humans and other animals excel in what may be termed hierarchical or multi-objective control, a concept that is receiving significant attention in research [9, 10, 11, 12].

The field of Human-Robot Collaboration (HRC) investigates approaches to leverage the strengths of both humans and robots in environments where they share workspace and objectives [13]. A prevalent strategy in HRC involves creating intuitive programming interfaces that enable humans to rapidly reprogram robots for new tasks. That is, flexibility and adaptability are sought to be improved by efficient utilization of human cognition. One such programming paradigm is Learning From Demonstration (LfD) [14], which is particularly popular as it requires no programming expertise [2]. LfD can be executed either directly or indirectly. In direct demonstrations, data is directly mapped to the robot's structure; for example, an operator might physically guide the robot. An indirect demonstration might involve the robot visually observing the performance of a task. One performance metric of an LfD interface is the generality of the learning. Comparing two employed learning algorithms, the less general one would need new demonstrations for smaller changes in conditions. Another metric is the number of demonstrations needed to teach a task, and yet another is the intuitiveness of the interface. Generally, there are two levels of learning in this context: low and high. Low-level learning involves directly encoding kinematic and/or dynamic features, such as a path or forces. High-level learning is more abstract, encompassing tasks like understanding the instruction 'When the human is thirsty, fetch a cup'.

In this study, we developed a Learning From Demonstration (LfD) interface. Our primary contribution is the formulation of a general approach for allocating hierarchical tasks based on specific task requirements. It is general in the sense that tasks can be defined in various spaces, including force, joint and Cartesian position, velocity, and others without changing the control algorithm. For example, when teaching the robot, one provides the teaching task specification. The interface is also agnostic to the utilized learning algorithm as long as it can generate references for specified tasks. However, Dynamical Movement Primitives were used for task learning (and to generate references).

The remainder of this chapter is organized into three sections. First in Sec. 1.1, an overview of related work is provided as well as the formulation of the research question for this thesis. How the research question is addressed, i.e. the methodology is detailed in Sec. 1.2. Finally, Sec. 1.3 outlines the structure of the thesis, serving as a roadmap for its organization and content.

1.1 Related Work and Research Questions

The related work section is divided into three parts. Research relating to 'Task Specifications' is presented in Sec. 1.1.1 where the aim is to find a task formalism that supports many types of low-level tasks. In Sec. 1.1.2, an overview of methods relating to hierarchical control is provided. Works on the use of Dynamical Move-

ment Primitives for learning and execution are reviewed in Sec. 1.1.3. Research questions are detailed in Sec. 1.1.4.

1.1.1 Task Specification

A *task*, in this context, refers to a specific objective assigned to a robot. Tasks can be defined at varying levels of abstraction. For example, "pick up the cup" represents a human-level (or high-level) task, "move to position A" denotes an instruction-level task, and "set joint to angle B" characterizes a low-level task. Regardless of level, each task needs to be converted into the control level supported by the robot, a joint-level representation in our case. That is, joint position (angles for revolute joints and distances for prismatic joints) must result from our controller, and be provided to the robot's control interface. Other joint-level representations could include joint velocity or torque (forces).

A task specification interface should, for flexibility, uniformly represent different task types. For example, a task could be directly defined in joint space, kinematic (position, velocity. etc.), or dynamic (torque), the same applies in Cartesian Space. A uniform representation has the same structure for all supported task types. Tasks are typically formulated as equality constraints. However, certain tasks are more suitably represented as inequality constraints, such as joint limit and obstacle avoidance. By supporting both equality and inequality constraints, the interface becomes more versatile.

The authors in [15] present an early example of a task specification that handles a single Cartesian impedance, admittance, or force-control task. However, this specification requires controller parameters, which necessitates the programmer to have an in-depth understanding of the controller's architecture.

Mason [16] introduces a specification for Cartesian hybrid force/position tasks. This specification demands the task frame's pose, the directions controlled by force, and their corresponding goals. This method is commonly known as Task Frame Formalism (TTF). A clear mathematical definition of TTF is given in [17]. This work was further expanded in [18]. The extension allows for a variable relationship between the task frame and the end-effector frame. It incorporates elements like feedforward velocities, tracking directions (to correct for orientation errors caused by contact), and motion constraints for both the task frame and end-effector frame. These constraints are integrated as redundancy (Sec. 2.1.3) resolution strategies, where the programmer decides on the constraints for the unconstrained dimensions of the task (minimize velocity, for example) such that the controller can find a unique solution.

The *Generalized task specification matrix* was introduced in [19]. This matrix can incorporate both motion and force tasks in any reference frame, whether constant, configuration-dependent, or time-varying. However, joint-level tasks are not included in this framework. An automatic determination of viable position and force-controlled directions is achieved by utilizing their orthogonality, calculated based

on the sensed end-effector wrench. These matrices transform position/force vectors to a task frame. In this frame, where position/force directions are easily specified, position/force components are extracted from the vectors. Subsequently, they are transformed back to the reference frame used by the controller.

The works mentioned above focus on task specifications for the end-effector, a reasonable approach since the goal is to be robot-independent. However, certain tasks are more conveniently defined at the joint level, such as joint limits and actuator torques. Additionally, in general, tasks cannot be encoded as inequality constraints, which is natural for managing joint or workspace limits. A task specification approach that accommodates these considerations is presented in [20]. This method allows for the representation of any task related to joint velocities, including those defined by inequality constraints, under the assumption that the controller utilizes numerical optimization.

1.1.2 Multi-objective Low-level Control

Robots often function in complex environments, requiring the simultaneous management of multiple objectives. Consider a drone: it must not only reach its destination quickly but also minimize energy consumption and avoid obstacles. Multi-objective control facilitates the concurrent consideration of these performance metrics. In the context of robotics, multi-objective control is inherently connected to redundancy resolution. Redundancy occurs when a robot has more degrees of freedom (DoFs) than are necessary for a specific task. A typical example is a seven-jointed robotic arm operating in 3D space, where only 6 DoFs are needed for the end-effector pose, 3 DoFs for position, and 3 DoFs for orientation. The additional DoFs can be utilized to address additional tasks like energy minimization. Furthermore, a task, orientation say, may be considered more important than a position (don't spill the cup) and therefore needs to be prioritized.

1.1.2.1 Analytical Methods

Analytical methods, in this context, leverage null-space projection and Jacobian pseudo-inverses for redundancy resolution. An early method addressing a single Cartesian space task is discussed in [21]. This work emphasizes the minimization of the joint velocity norm, demonstrating that inverse differential kinematics via the pseudo-inverse provides an optimal solution.

The introduction of null-space projections can be attributed to [22, 23], where secondary tasks are projected into the primary task's null space. A comprehensive framework for managing an arbitrary number of tasks is later introduced in [24]. Two primary approaches to null-space projection have been identified: *successive* and *augmented* projections [12]. While the former is computationally efficient, it doesn't guarantee strict priority adherence. The latter, although computationally demanding, ensures strict priority adherence [24].

Inequality constraints in robotics can be addressed in various ways, with popu-

lar methods including Gradient Projection (GPM) [25], Artificial Potential Fields (APF) [25], Clamping [26], and Task Scaling [27]. Gradient Projection involves defining a differentiable cost function linked to control variables and constraints. For instance, proximity to an obstacle or joint limits might increase this cost [23], and the gradient, a vector in the control space, is then appropriately projected. Artificial potential fields operate similarly, transforming constraints into equality constraints, with task objectives acting as either attractive or repulsive fields [28, 29]. Task scaling modifies the task scale to prevent constraint violations, maintaining the task direction at the expense of execution time [27]. Clamping addresses joint-level constraints by immobilizing joints at risk of violation, treating them as rigid links [30, 26, 31, 32, 33]. Both GPM and APF may not guarantee optimality or adherence to inequality constraints. Task scaling and clamping are also likely suboptimal. Additionally, the effectiveness of these techniques depends on parameter tuning.

Tasks can be assigned either strict or soft (non-strict) priorities relative to other tasks. When Task A has a strictly higher priority than Task B, the execution of Task B should not affect Task A's execution. Conversely, with soft priorities, task B may influence task A, if Task B cannot be executed otherwise. Strict priorities are crucial for tasks that must not be compromised, such as preventing self-collision. However, adhering strictly to priorities can lead to slow convergence or even the inability to execute secondary tasks. For instance, consider a primary positioning task and a secondary orientation task. Once the robot reaches the position goal, it might be in a configuration that prevents achieving the orientation goal without compromising the primary task, even though both tasks are simultaneously feasible. The null-space projector developed in [11, 34] handles both soft and strict priorities.

1.1.2.2 Numerical Methods

Here, numerical methods refer to methods using numerical optimization. To address the limitations (handling task constraints) of the analytical methods, Hierarchical Quadratic Programming (HQP) can be utilized. The foundational concept of HQP is to solve a series of quadratic programs (QPs), each addressing a different task. The first level handles the highest priority task and its constraints, while the subsequent level deals with a secondary task and its constraints, including the constraints imposed on it by higher-priority tasks [35, 36, 37]. Although this cascade approach can be computationally intensive, [38] and [11] demonstrate how to solve it using a single QP. As discussed, [11] also introduces a generalized null-space projector and a priority matrix, enabling flexible task prioritization, including task insertion and deletion. Other HQP algorithms typically support only strict priorities. Soft priorities might be managed using a single QP with weighted task cost functions [39, 40], but this does not assure performance for all tasks.

1.1.3 LfD with Motion Primitives

There is a wide range of Learning From Demonstration (LfD) methods available [2], and as mentioned earlier, these methods focus on teaching robots low-level variables. In this section, we will explore various versions of Dynamic Movement Primitives

(DMP) and discuss some works that prioritize DMPs.

The theory of DMPs derives from *motion primitives theory. Motion primitives theory* revolves around the question: "How do biological systems, such as humans and other animals, achieve the execution of intricate movements in a versatile and imaginative manner?". This theory aims to understand the mechanisms through which biological systems accomplish complex motions by employing the concepts of sequencing and adapting units of actions known as motion primitives [41].

Foundational works on DMPs are presented in [42, 43, 44]. A DMP models a learned variable ($y \in \mathbb{R}$) as a weakly non-linear dynamical system Eq. (1.1). That is, the acceleration of the variable ($\ddot{y} \in \mathbb{R}$) is given by an autonomous linear spring damper system plus a learned non-autonomous non-linear term ($f(x) \in \mathbb{R}$) Eq. (1.2). Note, f forces the system to behave a certain way and the autonomous system (set $f = 0$) converges to g, hence, f can be learned such that a desired behavior is generated. In Eq. (1.1), there are several parameters, $\tau \in \mathbb{R}$ is a positive temporal scaling factor, $\alpha_z, \alpha_z \in \mathbb{R}$ are non-negative and determines the dynamics of the spring-damping system. The variables $y_0 \in \mathbb{R}$ and $g \in \mathbb{R}$ are, respectively, the initial and target states, and $x \in \mathbb{R}$ is a phase variable which determines the temporal evolution of the system.

$$\tau\ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + (g - y_0)f(x) \tag{1.1}$$

$$f(x) = \frac{\sum_{i=1}^{N} \psi_i(x) w_i}{\sum_{i=1}^{N} \psi_i(x)} x \tag{1.2}$$

$$\psi_i(x) = exp(-h_i(x - c_i)^2), \ h_i, c_i \in \mathbb{R} \tag{1.3}$$

In [44], many proposed properties are verified in both simulation and physical experiments. These properties include:

- **Robustness to perturbation**: Convergence to the goal state is guaranteed.
- **Topological equivalence**. The learned trajectory remains qualitatively the same even if initial and final states are changed.
- **Coupling terms**: Terms for obstacle avoidance, contact forces, and others can easily be added.

The classic DMP formulation [42] does not support the encoding of Cartesian space orientation. To address this, a unit quaternion version of the DMP was introduced in [45]. However, this version does not have a part that is a linear spring-damping autonomous system, and therefore it has different properties from the positional DMP [42] and has been noted for its slow convergence [45]. An improved, though still non-linear, version is presented in [46], offering faster convergence and becoming the standard for orientation encoding DMP. Despite this improvement, its non-linear autonomous dynamics can result in undesired oscillatory behavior. This issue is rectified in the formulation provided in [47], which does not exhibit such oscillations. Additionally, [47] is mathematically equivalent to the positional DMP, hence, it has the same desirable properties.

Topological equivalence is particularly beneficial for tasks like painting and writing. However, as the trajectory scales, violations of kinematic limits and damage

to the robot or its environment can result. Therefore, imposing constraints may be necessary. In the classic DMP formulation (Eq. (1.1)), the non-linear learned term $f(x)$ (Eq. (1.2)) is scaled with $(g - y_0)$, thus, $(g - y_0)f$ scales the trajectory. Three main issues arise:

- No motion is generated if $(g - y_0) = 0$.
- Small changes in g can lead to large accelerations when $(g - y_0) \approx 0$, due to the non-linearity of f.
- The trajectory is mirrored if $(g - y_0)$ changes sign.

To address these issues, [48] offers a formulation that separates f and $(g - y_0)$. However, the global scaling property is lost when another term dominates f, especially if the relative position between the initial and goal states differs significantly from the demonstration. The authors in [49] suggest multiplying f in the [48] formulation with a rotation and dilation matrix $\mathbf{S}$. A similar matrix is proposed for the original formulation in [50], but these don't resolve the first issue listed. Another scaling approach is presented in [51], linking DMP to trajectory optimization and showing equivalence to optimizing trajectories over a specific norm. This norm determines how the DMP adapts the trajectory to new goals, and an optimal norm can be approximated by giving several demonstrations of tasks with changing goals. However, none of these works guarantee adherence to kinematic limits.

To enforce kinematic limits while ensuring a smooth trajectory, optimization is necessary. In the approach detailed in [52], the forcing term is redefined to provide a positional reference directly, instead of acceleration. This reference is generated through an analytical expression of smooth functions, allowing for the derivation of analytical expressions for velocity and acceleration references as well. Hence, the trajectory is analytically described, eliminating the need for integration and enabling an online optimization scheme where kinematic and environmental constraints can be incorporated [53]. We'll refer to the DMP formulated as in [52] and use the optimization scheme in [53] as DMP-MPC (Model Predictive Control (MPC)). Additionally, the method in [52] is reversible, which is beneficial for tasks that are more easily learned backward, for exception handling, and this reformulation can also be applied to the quaternion DMP presented in [47].

Probabilistic Movement Primitives (ProMPs) are another widely used approach [54]. ProMPs necessitate multiple demonstrations to learn a motion, maintaining a distribution over trajectories, essentially a blend or weighted sum of classic movement primitives. When provided with an initial state and goal, ProMPs select the most probable trajectory. These primitives are stochastic; thus, with the same initial state and goal, the trajectories will vary. The adaptability of ProMPs to new goals is contingent on the range of demonstrations provided; significantly different new goals can lead to unpredictable behavior.

The Kernelized Movement Primitive (KMP) represents an alternative probabilistic approach [55]. While DMPs and ProMPs require manual tuning of numerous parameters, such as the definition and quantity of basis functions and related parameters, KMP offers a non-parametric solution. Additionally, KMP is indifferent

to input-output variable types. Unlike DMP and ProMP, which are driven by time or phase (parameterization of time), KMP encompasses a broader range of mappings, such as position to velocity, with time/phase-driven scenarios being just one example of its capabilities.

The Gaussian Mixture Model (GMM) [56] is another method that, similar to the KMP, does not require specific input-output models or the specification of basis functions. This approach enables a robot to discern what to imitate without explicit instructions on features like Cartesian position. Instead, the robot identifies relevant features by analyzing the spatio-temporal variations and correlations observed during the demonstration.

Consider the prioritization of Dynamic Movement Primitives (DMPs) in a demonstrated manipulation task, where the end-effector position, orientation, and contact wrenches are recorded. During execution, these DMPs must be prioritized. The highest priority might be assigned to a wrench DMP that generates a reference along the normal to the task surface, while a secondary task could involve a position DMP along the task plane. In [57], priorities are learned from demonstrations using Probabilistic Movement Primitives (ProMPs). Each ProMP represents a task, and its priority is determined by the task's accuracy (variance) across demonstrations. If two ProMPs exhibit varying accuracies at different times, their relative priorities shift during execution. Hierarchical Quadratic Programming (HQP) is integrated with classic DMPs in [58], but only a single position task is encoded using the DMP.

1.1.4 Research Questions

Based on the current state of the art, the following open challenges have been identified:

- What combination of task specification and control methods leads to intuitive task programming?
- How can optimal prioritization of Dynamic Movement Primitives (DMPs) be learned from demonstrations?
- What method allows for the automatic selection of an optimal combination and prioritization of DMPs from a library of DMPs?

This study pursues the following research question:

- Can effective performance be achieved by allowing a user to define the hierarchy of DMPs?

1.2 Methodology

An overview of the system design is seen in Fig. 1.1.

The procedure of task specification (including priorities) will be defined by the high-level semantic module, depicted by the connection of the 'Semantic Task Specification Priorities' and 'Hierarchical Task Specification' block in Fig. 1.1.

The 'DMP Library' holds a set of DMPs trained through kinesthetic teaching. The DMP formulation used in this project will be based on [52]. During execution, the DMP-MPC [53] will be used for position tasks which enable the reference generation to account for environmental- and self-collision avoidance, work-space limits, velocity, and acceleration constraints. Such inequality tasks can also be accounted for by imposing inequality constraints in the 'Hierarchical Controller' block, however, the hierarchical controller is local and doesn't utilize model predictive control. To reduce the complexity of the problem, Aruco markers [59] can be used to represent goals and obstacles. The 'Hierarchical Task Specification' will define which DMPs should be loaded during execution and how to order them hierarchically. Both equality and inequality tasks can be specified, and defined as the reference source. Hence, the DMPs will not be the only source for the references. For example, during Kinesthetic Teaching (KT) the Force/Torque (F/T) sensor is the source to define contact tasks. The task specification generated in the block 'Semantic Task Specification Priorities' is a combination of the hierarchical specification model used in our previous work [20], [11]. The priority matrix from [11] is used instead of the simple strict priorities in [20].

The 'Tasks' block holds a set of hierarchically prioritized task definitions. A single task definition, has state variables $\boldsymbol{\xi}$, $\dot{\boldsymbol{\xi}}$, and task references $\boldsymbol{\xi}_{ref}$, $\dot{\boldsymbol{\xi}}_{ref}$ and $\ddot{\boldsymbol{\xi}}_{ref}$. The task also has the source of the task reference (DMP, sensor, etc.), the task Jacobian, and the task reference frame.

The 'Hierarchical Controller' block will use the null-space projection and optimization method presented in our work [11]. However, adapted to velocity-controlled manipulators. The controller will be synthesized from the specification. This means, the priority matrix will be defined, and the size of the projector and the optimization matrices will be initialized. This block will process the task objects, calculate desired task velocities, retrieve Jacobians, calculate the null-space projector, set optimization matrices, and solve the optimization problem to calculate the joint velocities $\dot{\mathbf{q}}$ that satisfies the hierarchical tasks. OSQP [60] can be used as the QP solver.

The physical robot is controlled via the virtual robot as in [20]. The state of the virtual robot ('vRobot' block), a kinematic model in our work, is integrated from a velocity command to give position commands to the physical robot.

The block 'pRobot', is the physical robot. Tiago from PAL robotics [61] will be used as the experimental platform. This robot is equipped with a F/T sensor, a mobile base, and a redundant robot arm. The code developed in this project will be based on ROS, and the C++ API.

The feedback from the physical robot is dynamically defined during execution, it will depend on the task specification. For example, joint positions/velocities, visual feedback, or/and F/T sensor values.

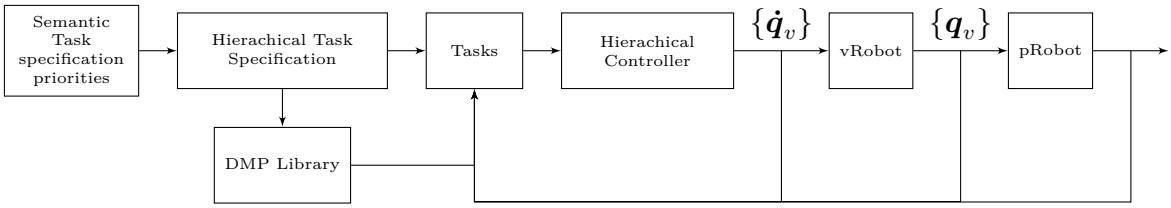


Figure 1.1: System design. The user provides a task specification. This specification is used to create an ordered list of task objects. A task object has a state, references, Jacobian, a compensation. The hierarchical controller processes the task objects to set up a generalized null space projector and solve a QP. This results in a joint velocity command sent to the virtual robot. Integrating the virtual robot joint state gives the position command for the physical robot.

1.3 Thesis outline

In Sec. 2.1 the necessary kinematics is presented, this is relevant for the understanding of most of the blocks in Fig. 1.1. Then we give mathematical details of the DMPs used in this thesis in Sec. 2.2 this relates to the 'DMP Library' block. In Sec. 2.3 we motivate the task specification format. The hierarchical controller is presented in Sec. 2.4, first the basics of null-space projection then the algorithms for the generalized projector used are outlined, then the QP optimization scheme follows. Basics for admittance control is provided in Sec. 2.5 and Chapter 2 ends with Sec. 2.6 which provides a short introduction to the lowpass Butterworth filter. In Chapter 3, the software implementation is presented, and data from different experiments are analyzed. Results, future directions, sustainability, and ethics are discussed in Chapter 4.

2
Theory

2.1 Kinematics

Kinematics is the study of motion without considering the forces (including torques) that cause this motion. Understanding kinematics is vital for developing robots capable of performing complex tasks with precision and accuracy [62]. This section briefly explains key concepts such as pose (Sec. 2.1.1), forward (Sec. 2.1.2), inverse (Sec. 2.1.3), and velocity kinematics (Sec. 2.1.4). Additionally, the section delves into the application of unit quaternions for representing orientation (Sec. 2.1.5). Readers who want more in-depth explanations of these concepts are referred to the textbooks [25, 62].

2.1.1 Pose Representation

Pose describes the position and orientation of an object. Position is in general an n-dimensional point p, it can be described using n coordinates relative to a coordinate reference frame. A frame consists of an origin o, a specific point in space, and a set of n orthonormal vectors. Orientation describes the directions of the orthonormal vectors relative to the corresponding set in the reference frame. In other words, pose describes a frame in terms of another. Here, we focus on the case where $n = 3$. For example, to control the pose of the robot's end-effector, a frame is attached to the end-effector and a reference frame is placed in the robot base. Let the desired pose also be described w.r.t. the base frame, then, a control command can be calculated by comparing the desired and the end-effector frames.

Consider two three-dimensional frames, named A and B. Then, a point $p \in \mathbb{R}^3$ can be expressed in these frames as follows:

$$p^A = \begin{bmatrix} p_x^A \\ p_y^A \\ p_z^A \end{bmatrix},$$
$$p^B = \begin{bmatrix} p_x^B \\ p_y^B \\ p_z^B \end{bmatrix}.$$

In this representation, the scalars $p_j^i \in \mathbb{R}, i \in \{A, B\}, j \in \{x, y, z\}$, are called coordinates. Note the distinction between points and vectors. Vectors have magnitude and direction and can originate from any point in space. Hence, the point p and the

vector $\mathbf{p}$ can share the same components but the latter can appear anywhere in space.

As stated, orientation describes the directions of the orthonormal vectors of a frame. As with position, a reference frame is needed to parameterize orientation. The rotation matrix is a common parameterization of orientation. Each column of the matrix describes a basis vector of the frame relative to the reference frame. Using superscript notation, $\mathbf{R}_B^A$ represents the orientation of frame B relative to frame A. Fig. 2.1 shows an example in two-dimensional space.

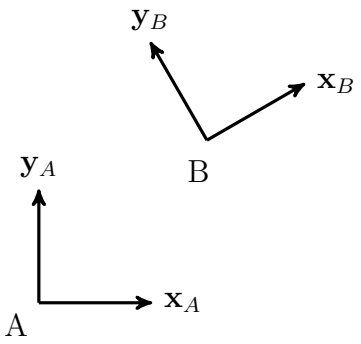


Figure 2.1: Two 2D frames, A and B.

Consider the two two-dimensional frames A and B in Fig. 2.1. Note, the orthonormal vectors $\mathbf{x}_i \in \mathbb{R}^2$ and $\mathbf{y}_i \in \mathbb{R}^2$ for $i \in \{A, B\}$ are the set of basis vectors of the corresponding frame. We choose A as the reference and describe the orientation of frame B. Specifically, we will describe the coordinates of $\mathbf{x}_B \in \mathbb{R}^2$ and $\mathbf{y}_B \in \mathbb{R}^2$ as linear combinations of $\mathbf{x}_A \in \mathbb{R}^2$ and $\mathbf{y}_A \in \mathbb{R}^2$:

$$\mathbf{x}_B^A = (\mathbf{x}_B \cdot \mathbf{x}_A)\mathbf{x}_A + (\mathbf{x}_B \cdot \mathbf{y}_A)\mathbf{y}_A,$$
$$\mathbf{y}_B^A = (\mathbf{y}_B \cdot \mathbf{x}_A)\mathbf{x}_A + (\mathbf{y}_B \cdot \mathbf{y}_A)\mathbf{y}_A.$$

Hence, the matrix

$$\mathbf{R}_B^A = \begin{bmatrix} (\mathbf{x}_B \cdot \mathbf{x}_A) & (\mathbf{x}_B \cdot \mathbf{y}_A) \\ (\mathbf{y}_B \cdot \mathbf{x}_A) & (\mathbf{y}_B \cdot \mathbf{y}_A) \end{bmatrix} \in \mathbb{R}^{2\times 2} \tag{2.1}$$

describes the orientation of frame B w.r.t. frame A. However, this representation requires n^2 parameters, which is redundant since only $n(n-1)/2$ parameters are necessary for a complete description of orientation in n-dimensional space. For $n = 3$, a commonly used minimal representation is Euler angles. However, Euler angles are limited by *representation singularities* (Sec. 2.1.4). In these singular configurations, the rank of the representation (rotation) matrix decreases, causing a loss of one or more DoFs; two or more of these angles describe rotation about the same axis. Hence, if the physical system rotates with a component in the singular direction(s), the Euler angles fail to model the system. In this work, we utilize a non-minimal parameterization, the unit quaternion (Sec. 2.1.5) for representing end-effector orientation because they're singularity-free, and rotation matrices for handling coordinate transformations since they're more intuitive.

For representing poses in 3D space, we use homogeneous transformation matrices, which are a combination of rotation and translation. This is expressed as:

$$H = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{2.2}$$

where $\mathbf{R} \in \mathbb{R}^{3\times 3}$ is the rotation matrix and $\mathbf{p} \in \mathbb{R}^3$ is the position vector. This representation is particularly beneficial in the study of forward kinematics (Sec. 2.1.2).

2.1.2 Forward Kinematics

Forward kinematics relates joint values $\mathbf{q} \in \mathbb{R}^n$ to the end-effector pose. A robot manipulator is composed of a series of links and joints, together forming a kinematic chain. For the sake of simplicity and without loss of generality, assume that each joint is either revolute or prismatic, as more complex joint types can often be modeled as combinations of these two. Each type of joint offers one degree of freedom (DoF) and is therefore described by a scalar variable. Revolute joints allow for rotational movement of one link relative to another, while a prismatic joint enables linear displacement between two connected links. Given the values of the revolute and prismatic joints as well as link geometries, forward kinematics gives the end-effector pose. Now we will introduce the concepts of *Configuration Space* and *Workspace*.

The *Configuration Space* (also known as Reachable Space) is the set of all possible joint configurations of the manipulator. In contrast, the *Workspace* contains viable poses of the robot's end-effector. Thus, the configuration space is characterized by the set of all feasible values of $\mathbf{q} \in \mathbb{R}^n$, with n as the number of DoFs, while the workspace contains all possible end-effector (ee) poses (Sec. 2.1.1), denoted as $\mathbf{x}_{ee}^w \in \mathbb{R}^6$. By attaching a frame (Sec. 2.1.1) to each link in the kinematic chain and considering the links as rigid bodies with 1-DoF joints, we can determine the position of every point on the manipulator if the joint positions $\mathbf{q} \in \mathbb{R}^n$ and link geometries are known. Forward kinematics is the mapping from the configuration space to the workspace, described by the function:

$$\mathbf{x}_{ee}^w = \mathbf{g}(\mathbf{q}) \tag{2.3}$$

Determining the function $\mathbf{g} : \mathbb{R}^n \longrightarrow \mathbb{R}^6$ is straightforward given the geometry of the robot manipulator, though it can be a tedious task. The relative pose of two successive frames in the kinematic chain depends only on the corresponding joint value and constant parameters (assuming rigid links) defined by the frames' placement. Therefore, the homogeneous transformation matrix (Sec. 2.1.1) representing the ee pose w.r.t. the world frame as a function of the joint states $\mathbf{q}$ is given by the product of individual transformations:

$$\mathbf{H}_{ee}^w = \mathbf{H}_1^w(q_1)\mathbf{H}_2^1(q_2)\dots\mathbf{H}_{ee}^{n-1}(q_n) \tag{2.4}$$

where $q_i \in \mathbb{R}, i \in \{1, 2, \dots, n\}$ are the coordinates of $\mathbf{q}$.

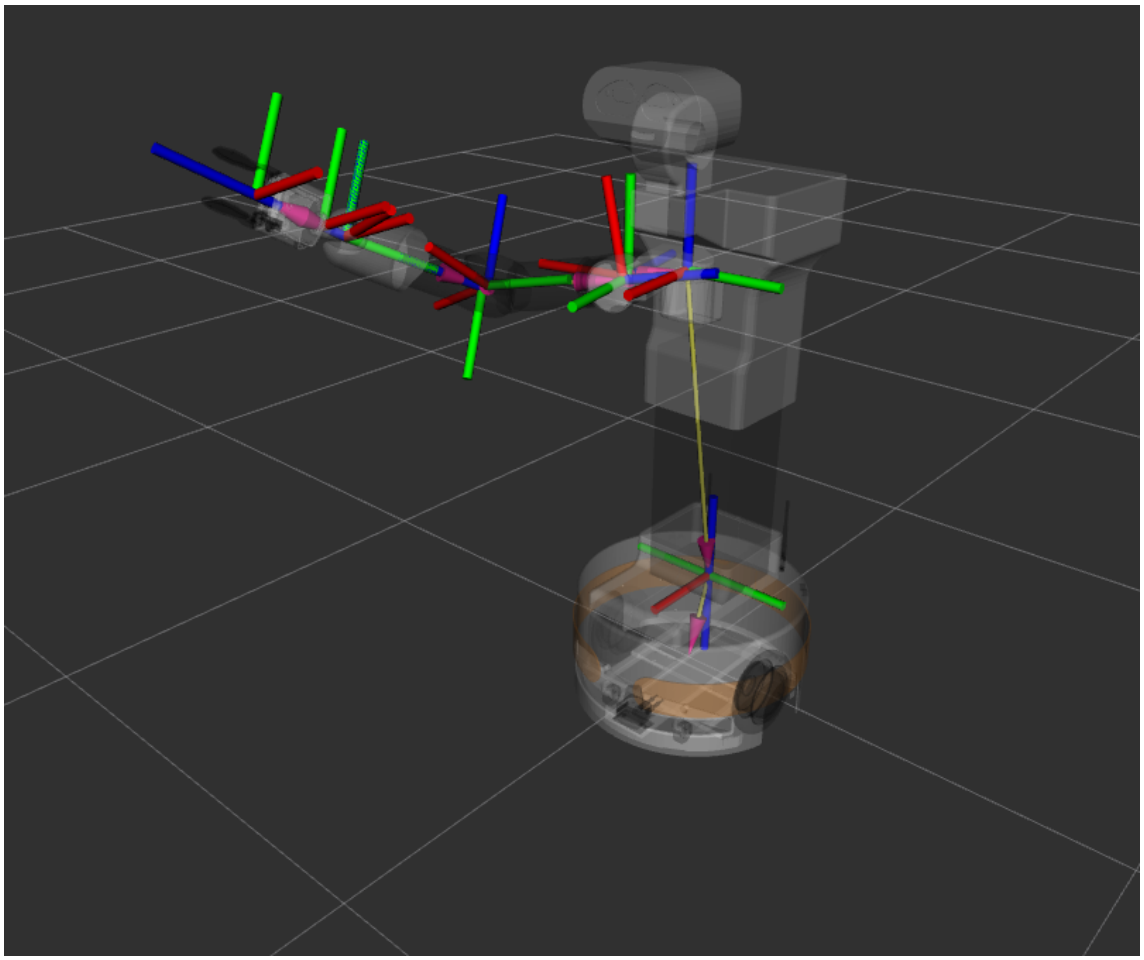


Figure 2.2: Frames (Sec. 2.1.1) used in the forward kinematic calulations for Tiago.

2.1.3 Inverse Kinematics

Inverse kinematics is the process of determining the joint configuration $\mathbf{q} \in \mathbb{R}^n$ required for a end-effector pose $\mathbf{x}_{ee}^w \in \mathbb{R}^m$ (Sec. 2.1.1). This relationship is mathematically expressed as $\mathbf{q} = \mathbf{g}^{-1}(\mathbf{x}_{ee}^w)$ (Sec. 2.1.2). The difficulty of solving this equation depends on the dimensions n and m. Therefore, it is useful to introduce two concepts: *Task Space* and the notion of *Redundancy*.

The *Task Space* is defined by the specific objectives that the robot's end-effector needs to achieve. For instance, if the objective is to move the end-effector along the global x-axis, the task space is one-dimensional ($m = 1$), focusing only on this axis. Conversely, if the task involves controlling the full Cartesian pose of the end-effector, the task space becomes six-dimensional (three for position and three for orientation) ($m = 6$).

A robot is considered redundant when it possesses more degrees of freedom than are strictly necessary to perform a given task, i.e., $n > m$. In such cases, the equation $\mathbf{q} = g^{-1}(\mathbf{x})$ can have multiple, or even infinite, solutions. Addressing this multiplicity of solutions often involves utilizing constraints and or optimization techniques to

identify the most suitable joint configuration for the given task. However, a detailed exploration of these optimization strategies is outside the scope of this thesis. The strategy employed in this thesis is explained in Sec. 2.4.

2.1.4 Velocity Kinematics

Velocity kinematics, also known as differential kinematics, describes the relationship between the joint velocities $\dot{\mathbf{q}} \in \mathbb{R}^n$ and the end-effector velocity $\dot{\mathbf{x}}_{ee}^w \in \mathbb{R}^6$. This relationship is described by Eq. (2.5).

$$\dot{\mathbf{x}}_{ee}^w = \frac{d}{dt}\mathbf{g}(\mathbf{q}) = \frac{\partial}{\partial \mathbf{q}}(\mathbf{g}(\mathbf{q}))\dot{\mathbf{q}} = \mathbf{J}\dot{\mathbf{q}} \tag{2.5}$$

In Eq. (2.5) $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{6\times n}$ is the Jacobian matrix. Note, Eq. 2.5 is linear, in contrast to the nonlinear nature of Eq. 2.3. Since inverse velocity kinematics is a linear mapping it is generally simpler than solving inverse kinematics (Sec. 2.1.3). However, in the case of redundant robots, the inverse velocity kinematics problem can still present an infinite number of solutions. As stated in Sec. 2.1.3, the redundancy resolution method used in this thesis is explained in Sec. 2.4. Another significant consideration in velocity kinematics is the occurrence of singularities.

A singularity arises when the Jacobian matrix (Eq. (2.5)) loses rank, leading to a loss of the robot's ability to instantaneously translate/rotate in certain directions. That is, in singular joint configurations ($\mathbf{q}$), specific end-effector velocities ($\dot{\mathbf{x}}_{ee}^w$) cannot be generated by any joint velocities ($\dot{\mathbf{q}}$). Avoiding these situations is important. There are three types of singularities to consider:

- *Boundary Singularities*: These occur at the mechanical limits of the robot's workspace. At a boundary singularity, one or more joints reach their physical limits. These singularities are generally easier to address as they are linked to the well-defined physical constraints of the robot. In this work, they're expressed as inequality constraints in an optimization problem (Sec. 2.4).
- *Internal Singularities*: These singularities happen within the robot's workspace, away from the boundaries. An internal singularity is characterized by a loss of rank in the Jacobian matrix, i.e. $\det(\mathbf{J}(\mathbf{q})) = 0$, resulting in the robot's inability to move instantaneously in certain directions. The control algorithm (Sec. 3.1.2) used in this work doesn't avoid these but chooses joint velocity $\dot{\mathbf{q}}$ that minimizes the difference between the possible motion and the desired one (Sec. 2.4.2).
- *Algorithmic Singularities*: These arise not from physical limitations, but due to algorithmic constraints, where the algorithm fails to find a solution despite the existence of a feasible physical solution. The control algorithm (Sec. 3.1.2) is hierarchical (Sec. 2.4) and locally optimized in the Configuration Space (Sec. 2.1.2), hence, it can reach local optimum w.r.t. to the entire task specification (Sec. 1.1.1). That is, the highest priority task has been accomplished but not the lower priority tasks even though they (or some) could be.

2.1.5 The Unit Quaternion

The special orthogonal group of degree 3, denoted as $SO(3)$, is the set of all 3×3 Euclidean matrices that represent right-handed orientations ($\mathbf{R} \in \mathbb{R}^{3\times3}$), with a determinant of 1 ($\det(\mathbf{R}) = 1$).

- **Quaternion Representation**:
  - An orientation (Sec. 2.1.1) in $SO(3)$ can be expressed as a unit quaternion $\mathbf{Q}$, given by:
    $$\mathbf{Q} = [w \ \mathbf{v}^\top] = [\cos(\theta) \ \sin(\theta)\mathbf{k}^\top]$$
    where $\mathbf{k} \in \mathbb{R}^3$ and 2θ represents the axis-angle equivalent. Here, $\mathbf{k}$ indicates the rotation direction and 2θ is the rotation angle relative to a frame. Note that $\|\mathbf{Q}\| = 1$, thus the term *unit* quaternion.
- **Quaternion Operations**:
  - Quaternion Product:
    $$\mathbf{Q}_1 * \mathbf{Q}_2 = \Big[w_1w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, \quad w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2\Big] \tag{2.6}$$
    If $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are unit quaternions with the equivalent rotation matrix representations $\mathbf{R}_1$ and $\mathbf{R}_2$, $\mathbf{Q}_1 * \mathbf{Q}_2$ is equivalent to $\mathbf{R}_1\mathbf{R}_2$.
  - The inverse of a unit quaternion is its conjugate, denoted as $\mathbf{Q}^{-1} = \bar{\mathbf{Q}} = [w \ -\mathbf{v}^\top]$.
- **Logarithmic and Exponential Mappings**:
  - The logarithmic and exponential mappings are defined as:
    $$\log(\mathbf{Q}) = \begin{cases} \frac{\mathbf{v}}{2\cos^{-1}(w) \ \|\mathbf{v}\|}, & \text{if } |w| \neq 1 \\ [0,0,0] & \text{otherwise} \end{cases}$$
    $$\exp(\boldsymbol{\eta}) = \begin{cases} [\cos(\|\boldsymbol{\eta}/2\|), \frac{\sin(\|\boldsymbol{\eta}/2\|)}{\|\boldsymbol{\eta}\|}\boldsymbol{\eta}^\top], & \text{if } \|\boldsymbol{\eta}\| \neq 0 \\ [1,0,0,0] & \text{otherwise} \end{cases}$$
    The logarithm corresponds to an axis and angle, interpreted as the unit time angular velocity needed to rotate from the reference frame to $\mathbf{Q}$. It's useful for measuring orientation errors (Sec. 2.2.2). The exponential mapping is its inverse, providing the quaternion for a given angular velocity. Furthermore, note that $SO(3)$ is not a vector space, e.g., let $\mathbf{Q}_1 = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^\top \in SO(3)$ and $\mathbf{Q}_2 = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^\top \in SO(3)$, then $\mathbf{Q}_1 + \mathbf{Q}_2 \notin SO(3)$ since $\|\mathbf{Q}_1 + \mathbf{Q}_2\| \neq 1$. However, the space described by $\log(\mathbf{Q})$ is a vector space and this allows to formulate orientation control (Sec. 2.2.2) in an analogous way to position control (Sec. 2.2.1).
- **Relation to Angular Velocity and Angular Acceleration**:
  - Denoting the angular velocity of $\mathbf{Q}$ as $\boldsymbol{\Omega} = [0 \ \boldsymbol{\omega}^\top]$, $\boldsymbol{\omega} \in \mathbb{R}^3$ the time derivative of $\boldsymbol{\eta} = \log(\mathbf{Q})$ relates $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ as:
    $$\boldsymbol{\Omega} = 2(\mathbf{J}_\eta\dot{\boldsymbol{\eta}}) * \bar{\mathbf{Q}}$$
  - The Jacobian $\mathbf{J}_\eta$ is:
    $$\mathbf{J}_\eta = \frac{1}{2}\begin{bmatrix} -sin(\theta)\mathbf{k}^\top \\ \frac{\sin(\theta)}{\theta}(\mathbf{I}_3 - \mathbf{k}\mathbf{k}^\top) + \cos(\theta)\mathbf{k}\mathbf{k}^\top \end{bmatrix}$$

where $\mathbf{I}_3$ is the 3×3 identity matrix.

– The angular acceleration $\dot{\boldsymbol{\omega}}$ is given by:

$$\dot{\boldsymbol{\Omega}} = 2(\dot{\mathbf{J}}_\eta \dot{\boldsymbol{\eta}} + \mathbf{J}_\eta \ddot{\boldsymbol{\eta}}) * \bar{\mathbf{Q}} - \frac{1}{2}\begin{bmatrix} \|\boldsymbol{\omega}\|^2 & \mathbf{0}_{1\times 3} \end{bmatrix}^\top$$

– Then we have

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_Q \dot{\mathbf{Q}}$$

where

$$\mathbf{J}_Q = \begin{bmatrix} \frac{-sin(\theta)+\theta cos(\theta)}{sin^2(\theta)}\mathbf{k} & \frac{\theta}{sin(\theta)}\mathbf{I}_3 \end{bmatrix}$$

These relations are used in the control loop (Sec. 3.1.2) to set velocities and accelerations.

2.2 Dynamical Movement Primitives

The theory presented here closely follows the presentation in [52] and [53]. As highlighted in the introduction (Sec. 1.1.3), a Dynamic Movement Primitive (DMP) comprises two main parts: a transformation system and a canonical system.

The transformation system models a task variable through a linear second-order differential equation, supplemented by a non-linear forcing term f, where f is a learned component.

The canonical system, on the other hand, is employed to synchronize multiple transformation systems. It also serves to decouple the transformation systems from direct time dependency. This decoupling is useful as it allows for manipulation of the temporal evolution of the tasks.

There are various formulations of DMPs in existing literature, this thesis primarily discusses the theory as laid out in [52]. For a more extensive review of different DMP formulations and their applications, refer to [41].

2.2.1 Positional DMP

Let $y \in \mathbb{R}$ represent a 1-DoF (Degree Of Freedom) variable, which could be a joint position, a Cartesian position coordinate, a force component, etc. The evolution of the phase variable $x \in \mathbb{R}$ is governed by a canonical system. Eq. (2.7) describes the transformation system for a 1-DoF DMP and the associated canonical system, Eq. (2.8).

$$\ddot{y} = \ddot{y}_x + D(\dot{y}_x - \dot{y}) + K(y_x - y), \tag{2.7}$$

$$\dot{x} = \frac{h(x)}{\tau}, \quad x\left(\frac{t_0}{\tau}\right) = x_0, \quad x\left(\frac{t_f}{\tau}\right) = x_f. \tag{2.8}$$

In Eq. (2.7), $K \in \mathbb{R}$ and $D \in \mathbb{R}$ are non-negative scalars, representing the spring and damping constants, respectively. The function $h(x) \in \mathbb{R}$ in the canonical system evolves monotonically from $x_0 \in \mathbb{R}$ to $x_f \in \mathbb{R}$ and remains constant outside this interval to ensure convergence.

The desired trajectory $y_x \in \mathbb{R}$, $\dot{y}_x \in \mathbb{R}$, and $\ddot{y}_x \in \mathbb{R}$ is scaled both spatially and temporally, see Eqs. 2.9 to 2.11. Spatial scaling, governed by $k_s \in \mathbb{R}$, adjusts the reference trajectory due to changes in the initial state $y_0 \in \mathbb{R}$ or target state $g \in \mathbb{R}$. Temporal scaling is accomplished through the phase variable x, its derivatives $\dot{x}$, $\ddot{x}$, and the duration variable $\tau \in \mathbb{R}$ which describes the desired time duration of the trajectory.

$$y_x = k_s(f_p(x) - f_p(x_0)) + y_0, \tag{2.9}$$
$$\dot{y}_x = k_s \dot{f}_p(x), \tag{2.10}$$
$$\ddot{y}_x = k_s \ddot{f}_p(x), \tag{2.11}$$
$$k_s = \frac{g - y_0}{f_p(x_f) - f_p(x_0)}, \tag{2.12}$$
$$f_p(x) = \frac{\sum_{i=1}^{N} \psi_i(x) w_i}{\sum_{i=1}^{N} \psi_i(x)}, \; w_i \in \mathbb{R} \tag{2.13}$$
$$\psi_i(x) = \exp(-\sigma_i(x - c_i)^2). \tag{2.14}$$

The Gaussian kernel widths $\sigma_i \in \mathbb{R}$ and centers $c_i \in \mathbb{R}$ are typically chosen as follows to ensure equal spacing in the time domain:

$$c_i = \exp\left(\alpha_x \frac{i-1}{N-1}\right), \quad \alpha_x \in \mathbb{R} \tag{2.15}$$
$$\sigma_i = \frac{1}{(c_{i+1} - c_i)^2}, \quad \sigma_N = \sigma_{N-1}. \tag{2.16}$$

Note that k_s and the forcing term f_p do not necessarily need to be formulated as described above. Alternative scaling terms may be necessary, particularly if g and y_0 are close, to avoid numerical instability as both the numerator and denominator approach zero. Indeed, different scaling approaches are documented in the literature (See Sec. 1.1.3).

Let us define $\boldsymbol{\phi} \in \mathbb{R}^N$ and $\mathbf{w} \in \mathbb{R}^N$ as

$$\boldsymbol{\phi} = \frac{1}{\sum_{i=1}^{N} \psi_i(x)} \begin{bmatrix} \psi_1(x) & \cdots & \psi_N(x) \end{bmatrix}, \tag{2.17}$$
$$\mathbf{w} = \begin{bmatrix} w_1 & \cdots & w_N \end{bmatrix}. \tag{2.18}$$

Then, the forcing term and its derivatives can be expressed as

$$f_p(x) = \boldsymbol{\phi}^\top \mathbf{w}, \tag{2.19}$$

$$\dot{f}_p(x) = \left(\frac{\partial \boldsymbol{\phi}}{\partial x}\dot{x}\right)^\top \mathbf{w}, \tag{2.20}$$

$$\ddot{f}_p(x) = \left(\frac{\partial^2 \boldsymbol{\phi}}{\partial x^2}\dot{x}^2 + \frac{\partial \boldsymbol{\phi}}{\partial x}\ddot{x}\right)^\top \mathbf{w}. \tag{2.21}$$

The resulting trajectory is a linear combination of the basis functions weighted by $\mathbf{w}$, effectively capturing the underlying structure of the learned movement (Sec. 2.2.3).

2.2.2 Quaternion DMP

Let $\mathbf{Q}$ be a unit quaternion (Sec. 2.1.5) representing a Cartesian space pose, with $\mathbf{Q}_0$ and $\mathbf{Q}_g$ denoting the initial and goal orientations, respectively. Define the orientation state variable as $\boldsymbol{\eta} = \log(\mathbf{Q} * \mathbf{Q}_0^{-1}) \in \mathbb{R}^3$, which maps the quaternion into a vector space. This mapping facilitates the application of a linear state space model, analogous to Eq. (2.7), and is described by Eq. (2.22).

$$\ddot{\boldsymbol{\eta}} = \ddot{\boldsymbol{\eta}}_x + \mathbf{D}(\dot{\boldsymbol{\eta}}_x - \dot{\boldsymbol{\eta}}) + \mathbf{K}(\boldsymbol{\eta}_x - \boldsymbol{\eta}) \tag{2.22}$$

Here, $\mathbf{D} \in \mathbb{R}^{3\times3}$ and $\mathbf{K} \in \mathbb{R}^{3\times3}$ are positive definite diagonal matrices representing damping and stiffness coefficients, respectively. The desired trajectory $\boldsymbol{\eta}_x \in \mathbb{R}^3$ and its scaling are defined as follows:

$$\boldsymbol{\eta}_x = \mathbf{K}_s \mathbf{f}_q(x), \tag{2.23}$$

$$\mathbf{K}_s = \operatorname{diag}\left(\boldsymbol{\eta}_g ./ \mathbf{f}_q(x_f)\right), \tag{2.24}$$

$$\mathbf{f}_q(x) = \mathbf{W}\boldsymbol{\phi}(x). \tag{2.25}$$

Here, $\boldsymbol{\eta}_g = \log(\mathbf{Q}_g * \mathbf{Q}_0^{-1})$, and the matrix $\mathbf{W}$ is defined as

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_x \\ \mathbf{w}_y \\ \mathbf{w}_z \end{bmatrix}^\top, \tag{2.26}$$

where $\mathbf{W} \in \mathbb{R}^{3\times N}$, each row corresponds to the weights that model the trajectories of the $\boldsymbol{\eta}$ components. The Gaussian kernels $\boldsymbol{\phi}$, as used in Sec. 2.2.1, are employed here as well.

2.2.3 Learning the Weights

Considering the mathematical equivalence between the positional DMP Eq. (2.7) and the quaternion DMP Eq. (2.22), the same learning algorithm can be applied to both for determining the weights. Let the collected data for 1-DoF be denoted as $y_d \in \mathbb{R}$. The optimization problem for learning the weights can be formulated as follows:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \left(\int_{x_0}^{x_f} J(f_p(x), y_d(x))\, dx \right) \tag{2.27}$$

where J represents a cost function. Common choices for J include the squared difference between the demonstration and the prediction, reflecting a Least Squares approach used in this thesis. Note that several parameters need to be set for the implementation: the number of Gaussian kernels N, kernel centers (c_i) and widths (σ_i), and the standard deviation scaling factor α_x, See Sec. 2.2.1.

2.2.4 DMP with an MPC extension

A property of the DMP reference formulation (Sec. 2.2.1) is that it's affine w.r.t. the weights (2.19), hence, no need for numerical integration to retrieve the trajectory for either position, velocity, or acceleration. This allows for a fast online optimization scheme.

The unconstrained DMP Eqs. (2.9) to (2.11) provides the reference in the optimization scheme. The weights of a movement primitive Eq. (2.28) are updated such that it minimizes the difference between the optimized movement primitive and the reference. This is done over a horizon of $N \in \mathbb{N}$ steps using a time step $T_s >> \Delta t$ where Δt is the control cycle of the control loop (Sec. 3.1.2). The optimization is formulated as a Quadratic Program (QP); due to the affine nature, various constraints are easily added, for instance, initial and final state constraints, position, velocity, acceleration, and obstacles.

Let $j \in \mathbb{Z}$ be an index denoting the current time, $T_{f,j} \in \mathbb{R}$, the current desired duration time, $\mathbf{g}_j \in \mathbb{R}^n$ current goal (or final state), the lower kinematic limits are denoted by underlines and the upper limits by over lines. That is, $\underline{\mathbf{y}} \in \mathbb{R}^n$ denotes the lower positional limits of $\mathbf{y} \in \mathbb{R}^n$ and $\overline{\mathbf{y}} \in \mathbb{R}^n$ gives upper limits and so forth. Let $\mathbf{y}_d \in \mathbb{R}^n$, $\dot{\mathbf{y}}_d \in \mathbb{R}^n$ and $\ddot{\mathbf{y}}_d \in \mathbb{R}^n$ denote the DMP generated reference over a horizon of N data points, note, this corresponds to N phase values. The model movement primitive is

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w}_1 \tag{2.28}$$

where $\mathbf{w}_1 \in \mathbb{R}^{3K}$ is the vector of weights to be optimized, the number of Gaussian kernels $K \in \mathbb{Z}$, and

$$\mathbf{\Phi} = \mathbf{I}_n \otimes \overline{\boldsymbol{\phi}}^\top \tag{2.29}$$

where $\otimes$ is the Kronecker product and $\overline{\boldsymbol{\phi}}$ are the truncated Gaussian kernels (Eq. (2.17)). The truncation makes the computation less expensive as only a subset of weights influences the output for a certain phase (time). Lower and upper kinematic soft limits are encoded in $\mathbf{l}_i \in \mathbf{R}^n$ and $\mathbf{u}_i \in \mathbb{R}^n$; desired limits but the optimizer is penalized but allowed to violate them. For each DoF and derivative order, there are slack variables collected in $\mathbf{s} = \begin{bmatrix} \mathbf{s}_p^\top & \mathbf{s}_v^\top & \mathbf{s}_a^\top \end{bmatrix}^\top \in \mathbb{R}^9$. The slack vector denotes maximum soft limit violations. To ensure that hard limits are respected, the norm of the kth slack variable is constrained: $\mathbf{l}_{i,hard}(k) \leq \|\mathbf{l}_i(k) - \mathbf{s}(k)\| \wedge \|\mathbf{u}_i(k) - \mathbf{s}(k)\| \leq \mathbf{u}_{i,hard}(k)$, in vector form, this relation is described as $\|\mathbf{s}\|_1 \leq \bar{\mathbf{s}}$. The optimization problem is stated in (2.30).

$$min_{\mathbf{w_1},\mathbf{s}} \sum_{i=j+1}^{j+N} (\boldsymbol{\psi}_i \mathbf{w_1} - \mathbf{z}_{d,i})^\top \mathbf{H}_i (\boldsymbol{\psi}_i \mathbf{w_1} - \mathbf{z}_{d,i}) + \mathbf{s}^\top \mathbf{H}_s \mathbf{s} \tag{2.30}$$

$$s.t.\ \mathbf{l}_i \leq \mathbf{A}_i \mathbf{w}_1 + \mathbf{I}_{3n}\mathbf{s} \leq \mathbf{u}_i,\ i = j+1, .., j+N \tag{2.31}$$

$$\|\mathbf{s}\|_1 \leq \bar{\mathbf{s}} \text{ Enforce hard limits} \tag{2.32}$$

$$\mathbf{A}_j \mathbf{w}_1 = \mathbf{b}_j \text{ Initial state constraint} \tag{2.33}$$

$$\mathbf{A}_L \mathbf{w}_1 = \mathbf{b}_l \text{ Final state constraint} \tag{2.34}$$

Where $\boldsymbol{\psi} = \begin{bmatrix} \boldsymbol{\Phi}_i^\top & \dot{\boldsymbol{\Phi}}_i^\top \end{bmatrix}^\top$, $\mathbf{z}_{d,i} = \begin{bmatrix} \mathbf{y}_{d,i}^\top & \dot{\mathbf{y}}_{d,i}^\top \end{bmatrix}^\top$, $\mathbf{H}_i = diag((1-\lambda)\mathbf{I}_n, \lambda \mathbf{I}_n)$, and $\lambda \in [0,1]$ prioritize optimization over position or velocity. Soft limit violations are penalized by $\mathbf{H}_s$. Finally, $\mathbf{A}_i = \begin{bmatrix} \boldsymbol{\Phi}_i^\top & \dot{\boldsymbol{\Phi}}_i^\top & \ddot{\boldsymbol{\Phi}}_i^\top \end{bmatrix}^\top$ and $\mathbf{b}_j = \begin{bmatrix} \mathbf{y}_j^\top & \dot{\mathbf{y}}_j^\top & \ddot{\mathbf{y}}_j^\top \end{bmatrix}^\top$ and correspondingly for L.

2.3 Task Specification

In this work, we control a virtual robot through velocity control (Sec. 1.2), hence, tasks need to be related to the virtual joint velocity $\mathbf{q}_v \in \mathbb{R}^n$. Let $\boldsymbol{\xi} = \mathbf{h}(\mathbf{q}_v) \in \mathbb{R}^m$ denote an arbitrary task type, e.g. joint position or a Cartesian pose, and $\mathbf{h} : \mathbb{R} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ describes its relation to the joint space. We then have

$$\dot{\boldsymbol{\xi}} = \frac{\partial \mathbf{h}(\mathbf{q}_v)}{\partial \mathbf{q}_v} \dot{\mathbf{q}}_v \tag{2.35}$$

where we'll refer to $\frac{\partial \mathbf{h}(\mathbf{q}_v)}{\partial \mathbf{q}_v}$ as the *task Jacobian* which provides the desired joint velocity mapping. Furthermore, a task can have

- A frame $\mathcal{F}$ attached to the robot (Sec. 2.3.1).
- Local task controller $\mathbf{r}$: $\dot{\boldsymbol{\xi}}^* = \mathbf{r}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}, \boldsymbol{\xi}_{ref}, \dot{\boldsymbol{\xi}}_{ref}, \ddot{\boldsymbol{\xi}}_{ref}) \in \mathbb{R}^m$. Where $\dot{\boldsymbol{\xi}}^*$ is the desired task velocity, $\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}$ the task state, and $\boldsymbol{\xi}_{ref}, \dot{\boldsymbol{\xi}}_{ref}, \ddot{\boldsymbol{\xi}}_{ref}$ the task reference. This controller can have some parameters and retrieve references from different sources like the F/T sensor or a DMP. In this work, the same control structure is used for all tasks:

  $$\dot{\boldsymbol{\xi}}_i^* = \dot{\boldsymbol{\xi}}_i + (M\ddot{\boldsymbol{\xi}}_{i,ref} + D(\dot{\boldsymbol{\xi}}_{i,ref} - \dot{\boldsymbol{\xi}}_i) + K(\boldsymbol{\xi}_{i,ref} - \boldsymbol{\xi}_i))\Delta t \tag{2.36}$$

  where Δt is the control period (Sec. 3.1.2) and $M, K, D \in \mathbb{R}$ are non-negative control parameters.
- Priority matrix $\mathbf{A}_i = diag(\mathbf{A}_{i,1} I_{m1}, .., \mathbf{A}_{i,n_t} I_{n_t})$ that encode the hierarchies of the other tasks relative to this task (Sec. 2.4.1).
- A task velocity compensation: $\dot{\boldsymbol{\xi}}_{comp}$
- Inequality constraints.

To illustrate the necessity of a task compensation [20], consider an indirect force task (not measured from the force sensor)

$$\boldsymbol{\xi} = \mathbf{f}_{ind} = \mathbf{K}_{spring}(\mathbf{x}_v - \mathbf{x}_p)$$

where $\mathbf{x}_v$, $\mathbf{x}_p$ are the virtual and physical robot poses. The time derivative is

$$\dot{\boldsymbol{\xi}} = \mathbf{K}_{spring}(\mathbf{J}_v(\mathbf{q}_v)\dot{\mathbf{q}}_v - \dot{\mathbf{x}}_p).$$

Therefore, the compensated task velocity $\dot{\boldsymbol{\xi}} + \mathbf{K}_{spring}\dot{\mathbf{x}}_p$ can be used to calculate the corresponding joint velocity. Note that the compensation is determined by the task type.

Then, a set of tasks can be specified in the form presented in Table 2.1

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
$\mathbf{A}_1$	cart. pose	DMP-MPC	(1,80,300)	$\mathcal{F}_1$	$\mathbf{l}_1 \leq \dot{\mathbf{x}} \leq \mathbf{u}_1$
$\mathbf{A}_2$	cart. pose	init	(1,80,300)	$\mathcal{F}_2$	$\mathbf{l}_2 \leq \dot{\mathbf{x}} \leq \mathbf{u}_2$
$\mathbf{A}_3$	joint position	$\mathbf{0}$	(0,80,0)	$\mathcal{F}_3$	$\mathbf{l}_3 \leq \dot{\mathbf{q}} \leq \mathbf{u}_3$

Table 2.1: Task Specification table. The first column describes the priority matrix of the task. The second column gives the type. The second column gives the reference source. The fourth column provides control parameters to the task controller Eq. (2.36). The fifth column provides the task frame (Sec. 2.3.1). The last column gives task inequality constraints, these need not be specified.

2.3.1 Task Frame

A task frame is a frame (Sec. 2.1.1) attached to the robot or some object in the environment i.e. to be controlled. For example, the frame could be attached to the end-effector. However, we may only be interested in controlling certain dimensions of the frame, e.g., the global positional x-direction and the z-axis of the frame. Furthermore, it may be more convenient to specify tasks in certain coordinate systems. The global x-direction is easily formulated in the global frame, and the end-effector frame z-axis is easily specified in the end-effector frame. When retrieving the desired joint velocity $\dot{\mathbf{q}}^* \in \mathbb{R}$ from the desired task velocity $\dot{\xi}^*$, the task Jacobian and $\dot{\xi}^*$ have to be described in the same reference system (Sec. 1.1.1). To allow for partial task specification in different frames, following [20], Cartesian space tasks are set using

$$\mathcal{F} = \text{diag}(\mathbf{R}_1^w, \mathbf{R}_2^w)\mathbf{S}_{select} \tag{2.37}$$

and for joint space tasks

$$\mathcal{F} = \mathbf{S}_{select} \tag{2.38}$$

where $\mathcal{F} \in \mathbb{R}^{6\times m}$ is a matrix representation of the task frame expressed in the reference system (w), $\mathbf{R}_1^w, \mathbf{R}_2^w \in \mathbb{R}^{3\times 3}$ are rotation matrices, for Cartesian space tasks $\mathbf{S}_{select} \in \mathbb{R}^{6\times m}$, and for joint space tasks $\mathbf{S}_{select} \in \mathbb{R}^{n\times m}$. The selection matrix $\mathbf{S}_{select}$ selects the relevant axis. For the example case given above (global $x-$axis,

frame z-axis), $\mathcal{F}$ would be

$$\mathcal{F} = diag(\mathbf{I}_3, \mathbf{R}_{ee}^{w}) \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.39}$$

where $\mathbf{R}_{ee}^{w}$ describes the end-effector orientation w.r.t. the reference system.

The transformation $\mathcal{F}$ is used to transform the task Jacobian (Eq. (2.40)).

$$\mathbf{J}_{\mathcal{F}} = \mathcal{F}^{\top}\mathbf{J} \tag{2.40}$$

where $\mathbf{J}$ is the full Jacobian for the task type.

2.4 Hierarchical Control

Hierarchical or Multi-objective control refers to the design and implementation of control strategies that simultaneously attempt to solve multiple goals or objectives. These objectives can often be conflicting, meaning the optimal solution for one objective might not be optimal for another. Redundancy (Sec. 2.1.3) opens the possibility for multi-objective optimization. For instance, a redundant robotic arm can be controlled to both reach an object and minimize energy consumption. In this thesis, hierarchical control is achieved by combining nullspace projection and numerical optimization.

First, the basics of nullspace projection are presented, followed by the specific methods used in this thesis. Then, a section presenting the optimization scheme is discussed. For more details on the theory presented in this section, the reader is directed to [11, 34, 12].

2.4.1 Nullspace Projection

Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the nullspace of $\mathbf{A}$ is defined as the set of all vectors $\mathbf{x} \in \mathbb{R}^n$ such that:

$$\mathbf{A}\mathbf{x} = \mathbf{0}$$

The null space of $\mathbf{A}$, denoted Null($\mathbf{A}$), satisfies the property Null($\mathbf{A}$) $\perp$ Row($\mathbf{A}$), where Row($\mathbf{A}$) is the row space of $\mathbf{A}$. Furthermore, since Null($\mathbf{A}$) and Row($\mathbf{A}$) span $\mathbb{R}^n$, any vector $\mathbf{v} \in \mathbb{R}^n$ can be decomposed as $\mathbf{v} = \mathbf{v}_{\text{row}} + \mathbf{v}_{\text{null}}$, which is depicted in Fig. 2.3.
A null space projector $\mathbf{P}(\mathbf{A}) \in \mathbb{R}^{n \times n}$ has the property $\mathbf{v}_{null} = \mathbf{P}(\mathbf{A})\mathbf{v}$.

Let us derive this projector. Assume the rows of $\mathbf{B} \in \mathbb{R}^{dim(\text{Row}(A)) \times n}$ contain an

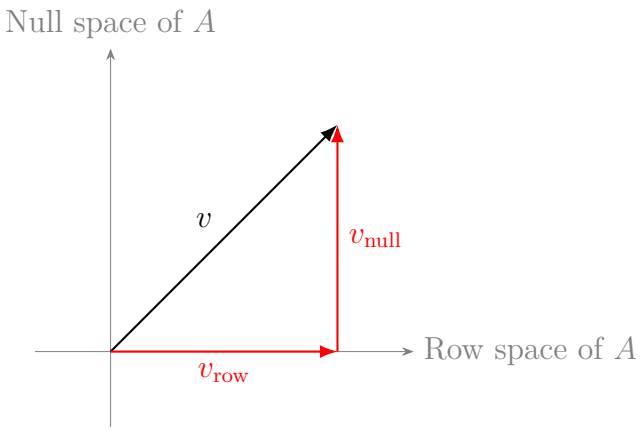


Figure 2.3: Decomposition of vector v into its components in the row space and null space of matrix $\mathbf{A}$.

orthonormal basis of Row($\mathbf{A}$), which can be achieved, e.g., by using elementary row operations. Note, $\mathbf{Bv}$ gives the scalar product of $\mathbf{v}$ with each of the orthonormal basis vectors spanning Row($\mathbf{A}$). To get $\mathbf{v}_{row}$ These scalar products must be multiplied with the corresponding basis vector, multiplied with $\mathbf{B}^\top$ i.e. We have

$$\begin{aligned}\mathbf{v}_{null} &= \mathbf{v} - \mathbf{v}_{row} = \mathbf{v} - \mathbf{B}^\top\mathbf{B}\mathbf{v} = (\mathbf{I} - \mathbf{B}^\top\mathbf{B})\mathbf{v} \\ \Rightarrow &\mathbf{P} = (\mathbf{I} - \mathbf{B}^\top\mathbf{B})\end{aligned}$$

The projector can also be found by:

$$\mathbf{P}(\mathbf{A}) = \mathbf{I} - \mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top)^{-1}\mathbf{A}$$

Where $\mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top)^{-1}$ is the Moore-Penrose pseudoinverse of $\mathbf{A}$.

Null space projection is a powerful tool in the context of redundant manipulators. Redundant manipulators can achieve a given task in multiple configurations and this redundancy can be exploited to achieve secondary objectives without affecting the primary task by projecting them into the nullspace of the primary task.

Consider a robot manipulator with a primary task ($\boldsymbol{\xi}_1 \in \mathbb{R}^{m_1}$) described by:

$$\boldsymbol{\xi}_1 = \mathbf{h}_1(\mathbf{q})$$

where $\mathbf{q} \in \mathbb{R}^n$ is the joint vector and $\mathbf{h}_1 : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is some task function, e.g., end-effector position, i.e., $m = 3$.

Now, suppose we have a secondary task ($\boldsymbol{\xi}_2 \in \mathbb{R}^{m_2}$) described by:

$$\boldsymbol{\xi}_2 = \mathbf{h}_2(\mathbf{q})$$

which could be any other objective, e.g., maintaining a certain joint posture or avoiding obstacles.

To achieve the primary task, we can define a control law:

$$\dot{\mathbf{q}} = \mathbf{J}_1^\dagger \dot{\boldsymbol{\xi}}_1^*$$
$$\mathbf{J}_1^\dagger = \mathbf{J}_1^\top (\mathbf{J}_1 \mathbf{J}_1^\top)^{-1}$$

where $\mathbf{J}_1^\dagger$ is the Moore-Penrose pseudoinverse of the Jacobian of $\mathbf{h}_1$ and $\dot{\boldsymbol{\xi}}_1^*$ is the desired task velocity.

To achieve the secondary task without affecting the primary one, use the null space projection:

$$\dot{\mathbf{q}} = \mathbf{J}_1^\dagger \dot{\boldsymbol{\xi}}_1^* + (\mathbf{I} - \mathbf{J}_1^\dagger \mathbf{J}_1) \mathbf{J}_2^\dagger \dot{\boldsymbol{\xi}}_2^* \tag{2.41}$$

The term $(\mathbf{I} - \mathbf{J}_1^\dagger \mathbf{J}_1)$ ensures that the secondary task is executed only in the null space of the primary task, i.e., without affecting the primary task. This idea can then be generalized to handle any number of tasks. This can be done in either a *successive* or *augmented* fashion. The projection matrix for task i in the *successive* method is calculated as

$$\mathbf{P}_i = \mathbf{P}_{i-1} (\mathbf{I} - \mathbf{J}_{i-1}^\top (\mathbf{J}_{i-1}^\#)^\top) \tag{2.42}$$

where $\mathbf{J}_{i-1}^\#$ is a generalized inverse of $\mathbf{J}_{i-1}$, choosing $\mathbf{J}_{i-1}^\# = \mathbf{J}_{i-1}^\top (\mathbf{J}_{i-1}^\top \mathbf{J}_{i-1})^{-1}$ gives the same projector as in Eq. (2.41). In the *augmented* method, we use an augmented Jacobian for each task. the augmented Jacobian for task i is given by

$$\mathbf{J}_i^{aug} = \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_i \end{bmatrix}, \tag{2.43}$$

here, i also denotes the priority, and 1 is the highest. The projection matrix for task i is then calculated as

$$\mathbf{P}_i^{aug} = \mathbf{I} - (\mathbf{J}_{i-1}^{aug})^\top (\mathbf{J}_{i-1}^{aug,\#})^\top \tag{2.44}$$

These methods are equivalent for less than four tasks but not otherwise, the successive method cannot guarantee strict priority between tasks separated by two or more priority levels which the augmented does. On the other hand, the augmented method is computationally more expensive. In this thesis we'll use the projector first introduced in [11] and later, further generalized in [34]. It's an augmented approach, where pseudoinverses are not used, instead, the projector is derived by calculating an orthonormal basis for $\text{Row}(\mathbf{J}_i^{aug})$. Additionally, a priority matrix is introduced in the projector that allows for continuous priority modulation from strict to any softness level, as well as task insertion and deletion as the task can be projected onto its own null space [63, 64, 65].

Now we will present the algorithm from [11, 34]. Let $n_t \in \mathbb{N}$ denote the number of tasks, task dimension $m_i \in \mathbb{N}$ of task $i \in \{1, \ldots, n_t\}$. The diagonal priority

matrix $\mathbf{A}_i \in \mathbb{R}^{n_t \times n_t}$ (2.45) for task i, encodes the relative priority to all n_t tasks, including itself.

$$\mathbf{A}_i = diag(A_{i1}\mathbf{I}_{m_1}, .., A_{in_t}\mathbf{I}_{m_t}) \tag{2.45}$$

where $A_{ij} \in [0, 1]$ and it indicates the priority of task j relative to task i:

$$\begin{aligned} A_{ij} = 0 &\Rightarrow j \text{ strictly lower priority than } i \\ 0 < A_{ij} < 1 &\Rightarrow j \text{ soft priority relative to } i \\ A_{ij} = 0 &\Rightarrow j \text{ strictly higher priority than } i. \end{aligned}$$

Let

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_{n_t} \end{bmatrix} \in \mathbb{R}^{(m_1+...+m_{n_t})\times n}$$

be the augmented Jacobian containing all n_t task Jacobians. Let $\mathbf{W} \in \mathbb{R}^{n\times n}$ be a weighting matrix that can be used to shape the projector. In this work we use $\mathbf{W} = \mathbf{I}_n$ which gives a statically-consistent projector, another option is $\mathbf{W} = \mathbf{M}^{-1}$, where $\mathbf{M} \in \mathbb{R}^{n\times n}$ is the joint space inertia matrix, this choice produces a dynamically consistent projector. However, we perform kinematic control, therefore, this projector is not needed. The pseudo-codes for the algorithms are shown in Alg.1 and Alg.2. The small scalar $\epsilon > 0$ used in Alg. 1 and Alg. 2 is for numerical stability.

Algorithm 1 Weighted shaped projector for task i

procedure DYNGHCPROJECTOR($\mathbf{W}, \mathbf{A_i}, \mathbf{J}_{aug}, \epsilon$)
  $n \leftarrow GetNumOfCols(\mathbf{J}_{aug})$
  $index \leftarrow GetRowsIndexDescOrder(\mathbf{A}_i)$
  $\mathbf{A}_{i_s} \leftarrow SortRows(\mathbf{A}_i, index)$
  $\mathbf{J}_{aug_s} \leftarrow SortRows(\mathbf{J}_{aug}, index)$
  $\mathbf{L} \leftarrow CholeskyDecomposition(\mathbf{W})$
  $\mathbf{B}_i, \mathbf{R}_i, \mathbf{p}, \mathbf{r} \leftarrow modifiedqrd((\mathbf{L}^\top \mathbf{J}_{aug_s}), \epsilon)$ ▷ Alg. 2
  $\mathbf{A}_{i_{sr}} \leftarrow GetSubDiagMatrix(\mathbf{A}_{i_s}, \mathbf{p})$
  return $\mathbf{P}_i \leftarrow (\mathbf{L}^\top)^{-1}(\mathbf{I} - \mathbf{B}_i\mathbf{A}_{i_{sr}}\mathbf{B}_i^\top)\mathbf{L}^\top$
end procedure

In Alg. 1, $\mathbf{A}_i$ is sorted in descending order, i.e., the first diagonal element in $\mathbf{A}_{i_s}$ has the largest value and the last diagonal value has the smallest. Increased values correspond to increased priority relative to task i. The augmented Jacobian is sorted correspondingly, hence, the upper Jacobians in $\mathbf{J}_{aug_s}$ belong to the tasks with relatively high priority value w.r.t. task i. The matrix $\mathbf{L}$ is related to $\mathbf{W}$ as $\mathbf{W} = \mathbf{LL}$. The column vectors of $\mathbf{B}_i$ hold orthonormal basis for Row$(\mathbf{L}^\top \mathbf{J}_{aug_s})$ and $\mathbf{A}_{i_{sr}}$ is the priority matrix that correspond to the independent rows of $\mathbf{J}_{aug_s}$, these were calculated to get $\mathbf{Q}_i$, and the indices stored in $\mathbf{p}$. Finally, the projector for task i is returned as

$$\mathbf{P}_i = (\mathbf{L}^\top)^{-1}(\mathbf{I} - \mathbf{B}_i\mathbf{A}_{i_{sr}}\mathbf{B}_i^\top)\mathbf{L}^\top \tag{2.46}$$

Algorithm 2 Modified QR decomposition for a matrix X

procedure MODIFIEDQRD($\mathbf{X}, \epsilon$)
  $m \leftarrow GetNumOfCols(\mathbf{X})$
  $n \leftarrow GetNumOfRows(\mathbf{X})$
  $i \leftarrow 0$
  for $k \leftarrow 0$ to $m - 1$ **do**
    if $i \geq n$ **then**
      break
    end if
    $\mathbf{B}[:, i] \leftarrow \mathbf{X}[:, k]$
    for $j \leftarrow 0$ to $i - 1$ **do**
      $\mathbf{R}[j, i] \leftarrow \mathbf{B}[:, i]^\top \mathbf{X}[:, k]$
      $\mathbf{B}[:, i] \leftarrow \mathbf{B}[:, i] - \mathbf{B}[:, j](\mathbf{B}[:, i]^\top \mathbf{B}[:, j])$
    end for
    $\mathbf{R}[i, i] \leftarrow \text{norm}(\mathbf{B}[:, i])$
    if $\mathbf{R}[i, i] > \epsilon$ **then**
      $\mathbf{B}[:, i] \leftarrow \mathbf{B}[:, i]/\mathbf{R}[i, i]$
      $\mathbf{p}[i] \leftarrow k$
      $i \leftarrow i + 1$
    end if
  end for
  $r \leftarrow i$
  return $\mathbf{B}, \mathbf{R}, \mathbf{p}, \mathbf{r}$
end procedure

and the full projector is

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_{n_t} \end{bmatrix} \tag{2.47}$$

The source code is found at `https://github.com/ndehio/2019_ICRA`.

2.4.2 QP Optimization

The objective is to velocity control a virtual model of the physical robot, integrate the virtual state, and command the resulting joint position of the physical robot (Sec. 1.2). For a set of $n_t \in \mathbb{N}$ tasks $\{\xi_1, \ldots, \xi_{n_t}$, hierarchical control can be performed by using nullspace projection as in Sec. 2.4.1, however, in that case it is not straightforward to enforce task inequality constraints (Sec. 1.1.2.1). In contrast, inequality constraints are easily enforced using numerical optimization.

We will use the Quadratic Programming (QP) formulation presented in [11], but adapted for velocity control. Specifically, we aim to identify vectors and matrices (including their dimensions) $\mathbf{x}$, $\mathbf{H}$, $\mathbf{f}$, $\mathbf{A}_{ineq}$, $\mathbf{h}$, $\mathbf{A}_{eq}$, and $\mathbf{b}$ in the optimization problem (2.48). Each task $\boldsymbol{\xi}_i$ will be individually optimized and yield an optimal joint task velocity $\dot{\mathbf{q}}_i^*$. Note, $\dot{\mathbf{q}}_i^*$ is the optimal joint velocity for task i without consid-

eration of other tasks. The solution is then given by projecting each $\dot{\mathbf{q}}_i^*$ into the nullspace of higher priority tasks using the nullspace projector $\mathbf{P}$ from Eq.(2.47).

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^\top \mathbf{H}\mathbf{x} + \mathbf{f}^\top \mathbf{x} \tag{2.48}$$

$$\text{Subject to:} \quad \mathbf{A}_{ineq}\mathbf{x} \leq \mathbf{h} \tag{2.49}$$

$$\mathbf{A}_{eq}\mathbf{x} = \mathbf{b} \tag{2.50}$$

Firstly, we address the objective function. Given our focus on velocity control, it is logical to minimize the task velocity errors. These errors are defined as $\dot{\mathbf{\Delta}}_i = \dot{\boldsymbol{\xi}}_i^* - \mathbf{J}_i\dot{\mathbf{q}}_i'$, where $i \in \{1, 2, .., n_t\}$ denotes the task index, $\dot{\boldsymbol{\xi}}_i^* \in \mathbb{R}^{m_i}$ desired task velocity (Sec. 1.1.1, $\mathbf{J}_i \in \mathbb{R}^{m_i \times n}$ represents the task Jacobian, and $\dot{\mathbf{q}}_i' \in \mathbb{R}^n$ is the task's joint velocity to optimize. Consequently, for the objective function we have:

$$\min_{\mathbf{x}} \sum_{i=1}^{n_t} \|\dot{\mathbf{\Delta}}_i\|^2. \tag{2.51}$$

Now

$$\begin{aligned}\|\dot{\mathbf{\Delta}}_i\|^2 &= (\dot{\boldsymbol{\xi}}_i^* - \mathbf{J}_i\dot{\mathbf{q}}_i')^\top(\dot{\boldsymbol{\xi}}_i^* - \mathbf{J}_i\dot{\mathbf{q}}_i') \\ &= (\dot{\boldsymbol{\xi}}_i^d)^\top\dot{\boldsymbol{\xi}}_i^* - 2(\dot{\boldsymbol{\xi}}_i^d)^\top\mathbf{J}_i\dot{\mathbf{q}}_i' + (\dot{\mathbf{q}}_i')^\top\mathbf{J}_i^\top\mathbf{J}_i\dot{\mathbf{q}}_i'\end{aligned}$$

Note, $\left(\dot{\boldsymbol{\xi}}_i^*\right)^\top \dot{\boldsymbol{\xi}}_i^*$ is unaffected by $\dot{\mathbf{q}}_i'$ and can be ignored. Let $\dot{\mathbf{q}}' = \left[\dot{\mathbf{q}}_1', \cdots, \dot{\mathbf{q}}_{n_t}'\right]^\top \in \mathbb{R}^{n_t n}$. Then,

$$\sum_{i=1}^{n_t} \|\dot{\mathbf{\Delta}}_i\|^2 = \dot{\mathbf{q}}'^\top diag(\mathbf{J}_1^\top\mathbf{J}_1, \ldots, \mathbf{J}_{n_t}^\top\mathbf{J}_{n_t})\dot{\mathbf{q}}' - 2\left[(\dot{\boldsymbol{\xi}}_1^d)^\top\mathbf{J}_1 \quad \ldots \quad (\dot{\boldsymbol{\xi}}_{n_t}^d)^\top\mathbf{J}_{n_t})\right]\dot{\mathbf{q}}'$$

We identify

$$\begin{aligned}\mathbf{x} &= \dot{\mathbf{q}}' \in \mathbb{R}^{n_t n} \\ \mathbf{H} &= 2diag(\mathbf{J}_1^\top\mathbf{J}_1, \ldots, \mathbf{J}_{n_t}^\top\mathbf{J}_{n_t}) \in \mathbb{R}^{n_t n \times n_t n} \\ \mathbf{f}^\top &= -2\left[(\dot{\boldsymbol{\xi}}_1^d)^\top\mathbf{J}_1 \quad \ldots \quad (\dot{\boldsymbol{\xi}}_{n_t}^d)^\top\mathbf{J}_{n_t})\right] \in \mathbb{R}^{1 \times n_t n}\end{aligned}$$

For numerical stability, a small regularization term $\rho > 0$ is added to $\mathbf{H}$.

$$\mathbf{H} = 2diag(\mathbf{J}_1^\top\mathbf{J}_1, \ldots, \mathbf{J}_{n_t}^\top\mathbf{J}_{n_t}) + \rho\mathbf{I}.$$

Now, we address the constraints. The null space projector $\mathbf{P} = \left[\mathbf{P}_1 \quad .. \quad \mathbf{P}_{n_t}\right] \in \mathbb{R}^{n \times n_t n}$ (Alg. 1) and the optimized augmented joint velocity $\dot{\mathbf{q}}'$ give the commanded joint velocity $\dot{\mathbf{q}}_{cmd} = \mathbf{P}\dot{\mathbf{q}}'$, which must satisfy the joint velocity constraints. However, we also have joint position and acceleration constraints. Let $\underline{\mathbf{q}}, \underline{\dot{\mathbf{q}}}, \underline{\ddot{\mathbf{q}}} \in \mathbb{R}^n$ denote the lower limits for position, velocity, and acceleration respectively. In the same form,

the upper limits are defined as $\overline{\mathbf{q}}$, $\dot{\overline{\mathbf{q}}}$ and $\ddot{\overline{\mathbf{q}}}$. Using the method from [66], we'll turn these limits into a single joint velocity constraint

$$\underline{\dot{\mathbf{q}}}_m = \max\{\frac{\underline{\mathbf{q}} - \mathbf{q}}{\Delta t},\ \underline{\dot{\mathbf{q}}},\ -\sqrt{2\underline{\ddot{\mathbf{q}}}(\mathbf{q} - \underline{\mathbf{q}})}\} \tag{2.52}$$

$$\dot{\overline{\mathbf{q}}}_m = \min\{\frac{\overline{\mathbf{q}} - \mathbf{q}}{\Delta t},\ \dot{\overline{\mathbf{q}}},\ \sqrt{2\ddot{\overline{\mathbf{q}}}(\overline{\mathbf{q}} - \mathbf{q})}\} \tag{2.53}$$

where, min{ } and max{ } operate row-wise. That is,

$$\underline{\dot{\mathbf{q}}}_m \leq \mathbf{P}\dot{\mathbf{q}}' \leq \dot{\overline{\mathbf{q}}}_m \tag{2.54}$$

and we can set (2.55)

$$\mathbf{h} = \begin{bmatrix} \dot{\overline{\mathbf{q}}}_m \\ -\underline{\dot{\mathbf{q}}}_m \end{bmatrix} \in \mathbb{R}^{2n} \tag{2.56}$$

$$\mathbf{A}_{ineq} = \begin{bmatrix} \mathbf{P} \\ -\mathbf{P} \end{bmatrix} \in \mathbb{R}^{2n \times n_t n} \tag{2.57}$$

In our case, we don't have equality constraints. Therefore, the linear system $\mathbf{A}_{eq}\mathbf{x} = \mathbf{b}$ is not used.

2.5 Admittance Control

Admittance in mechanical systems is a measure of how easily a system allows mechanical motion when subjected to a force. To enable KT, an admittance controller is used in this work (See Sec. 3.2.2). Below the basic theory is explained, and in Sec. 3.3, experiments are presented.

Assume an ideal F/T-sensor, located in a frame s. Let w, denote a reference frame; $\mathbf{p}_s^w \in \mathbb{R}^3$ sensor position, $\mathbf{p}_{CoM}^w \in \mathbb{R}^3$ end-effector center of mass (CoM), $m \in \mathbb{R}$ end-effector mass and $\mathbf{g}^w \in \mathbb{R}^3$ gravitational acceleration and $\mathbf{F}_{ext}^w, \mathbf{T}_{ext}^w \in \mathbb{R}^3$ applied force and torque on the end-effector.

The sensor measures the force applied by the end-effector on the sensor which includes the effect of gravity, end-effector motion, $\mathbf{F}_{ext}^w$ and $\mathbf{T}_{ext}^w$. Since $\mathbf{F}_{ext}^w$ and $\mathbf{T}_{ext}^w$ are sought, the effect of gravity and motion has to be compensated. Let $\mathbf{F}_{sens}^s, \mathbf{T}_{sens}^s \in \mathbb{R}^3$ denote the measured force and torque.

$$\mathbf{F}_{sens}^s = m(\mathbf{g}^s - \ddot{\mathbf{p}}_{CoM}^s) + \mathbf{F}_{ext}^s \Rightarrow \mathbf{F}_{ext}^w = \mathbf{R}_s^w(\mathbf{F}_{sens}^s - m(\mathbf{g}^s - \ddot{\mathbf{p}}_{CoM}^s)) \tag{2.58}$$

$$\mathbf{T}_{sens}^s = (\mathbf{p}_{CoM}^s - \mathbf{p}_s^s) \times \mathbf{F}_{sens}^s \Rightarrow \mathbf{T}_{ext}^w = (\mathbf{p}_{CoM}^w - \mathbf{p}_s^w) \times \mathbf{F}_{ext}^w \tag{2.59}$$

Note that the end-effector inertia is ignored in the torque calculation. Since the kinematic model, used to control our robot, is integrated using velocity, $\mathbf{F}_{ext}^w$ and $\mathbf{T}_{ext}^w$ have to be turned into a velocity command.

$$\begin{bmatrix} \dot{\mathbf{p}}_{ref}^w \\ \dot{\boldsymbol{\omega}}_{ref}^w \end{bmatrix} = \begin{bmatrix} A_F \mathbf{F}_{ext}^w \\ A_\omega \mathbf{T}_{ext}^w \end{bmatrix} \tag{2.60}$$

where $\dot{\mathbf{p}}_{ref}^w, \dot{\boldsymbol{\omega}}_{ref}^w \in \mathbb{R}^3$ are the resulting Cartesian space reference velocities and $A_F, A_{omega} \in \mathbb{R}$ describe the force and torque admittance respectively.

2.6 Butterworth Filter

In Sec. 2.5, an ideal sensor was assumed. However, in practice, sensors often contain noise, as discussed in Section 3.2.1. High-frequency noise components can be mitigated by employing a low-pass filter. One effective type of low-pass filter is the Butterworth filter.

The Butterworth filter is categorized as an Infinite Impulse Response (IIR) filter. This designation implies that, in its ideal form, the filter's impulse response never reaches zero. This characteristic is fundamentally different from Finite Impulse Response (FIR) filters, where the impulse response settles to zero after a certain period. IIR filters generally require fewer coefficients (Eq. (2.61)) than FIR filters to achieve a similar frequency response, making them more computationally efficient. Due to their recursive nature, IIR filters can achieve sharper cutoffs with fewer filter coefficients than FIR filters. There are other pros and cons with IIR vs FIR filter types but the stated advantages of the IIR filter is why it was chosen here.

A key attribute of the Butterworth filter is its maximally flat frequency response in the passband. This means that within the range of frequencies that the filter allows to pass (up to the cutoff frequency), the response is as flat as possible, minimizing distortion to the signal. The 'flatness' of the response distinguishes Butterworth filters from other types of filters, such as Chebyshev or elliptical filters, which may exhibit ripples in the passband.

A digital IIR-filter can, in general, be described by a difference equation

$$y[k] = \frac{1}{a_0}(b_0x[k]+b_1x[k-1]+\ldots+b_Bx[k-B]-a_1y[k-1]-a_2y[k-2]-\ldots-a_Ay[k-A]) \tag{2.61}$$

where $y[k] \in \mathbb{R}$ is the filter output at timestep $k \in \mathbb{Z}$ and $x[k] \in \mathbb{R}$ is the filter input (signal to filter). The integers $A, B \in \mathbb{Z}$ denote feedback and feedforward filter order respectively. Note, that the filter has to be initialized to work properly. The coefficients $a_i, b_i \in \mathbb{R}$, determine the filter properties.

In the design of a Butterworth filter, several parameters are important. The cutoff frequency defines the point at which the filter starts attenuating the frequency components of the signal. Frequencies below this cutoff are allowed to pass with minimal attenuation, while those above it are progressively attenuated. The sampling frequency is another important parameter in the case of digital Butterworth filters. It denotes the rate at which the signal is sampled to be processed digitally. The relationship between the cutoff frequency and the sampling frequency is essential to ensure appropriate filter performance and to avoid issues such as aliasing. The cutoff and sampling frequencies and filter order can be given as design requirements which in turn determine the filter coefficients $a_i, b_i \in \mathbb{R}$ in Eq. (2.61).

In general, filters exhibit a characteristic frequency response. Figs.2.4 and 2.5 illustrate the frequency responses of fourth and fifth-order lowpass Butterworth fil-

ters, respectively. Note, that the higher-order filter has a more ideal magnitude response (frequencies outside the passband have a lower gain relative to the fourth-order filter). On the other hand, it induces a larger phase shift. This observation underscores a fundamental trade-off in higher-order Butterworth filters: as the filter order increases, the magnitude response approaches the ideal at the price of a larger phase shift. Consequently, this leads to larger time delays Fig.2.6. This phenomenon highlights the balance required between filter order and the resulting phase and time delay implications in filter design.

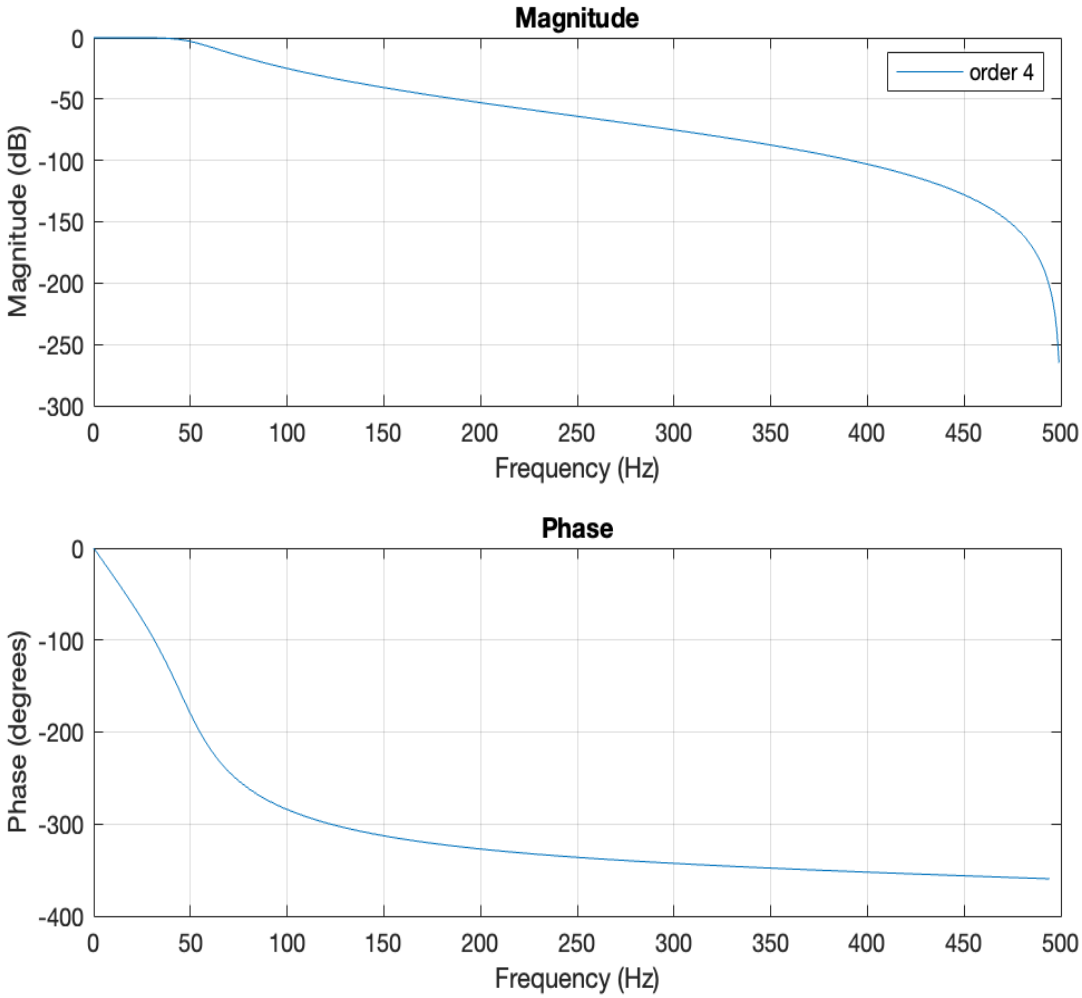


Figure 2.4: Frequency response of a fourth-order Butterworth filter.

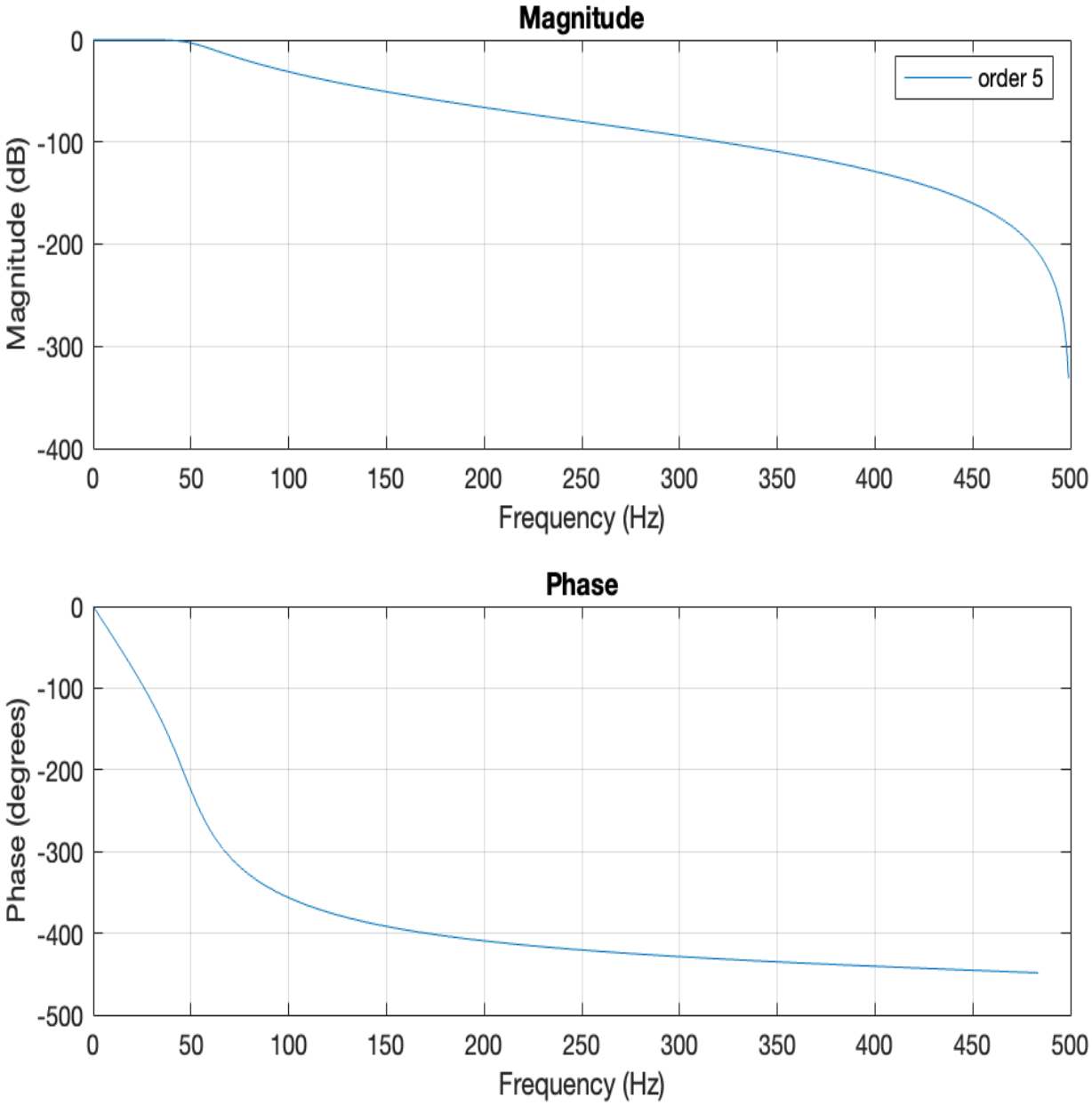


Figure 2.5: Frequency response of a fifth-order Butterworth filter.

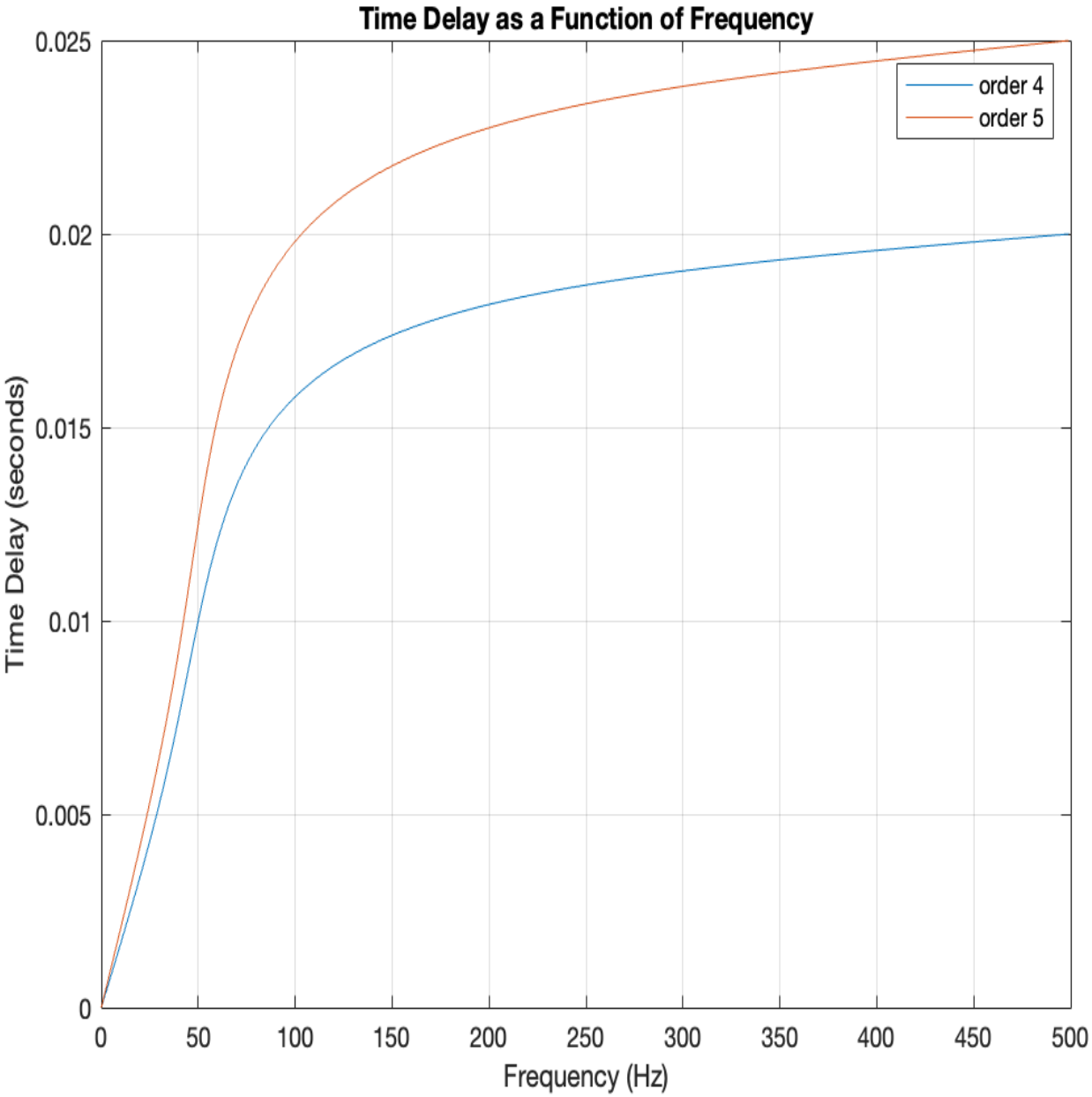


Figure 2.6: Time delays induced by the Butterworths filters of orders four and five.

3

Robot Kinesthetic Teaching: Implementation and Experimental Validation

This chapter delves into, as the title suggests, the implementation and experimental validation of a Kinesthetic Teaching (KT) interface (or system), examining the interface's composite elements. The discussion begins in Sec. 3.1, where we explore the system's software implementation and control algorithm. Subsequently, Sec. 3.2 presents the algorithms used for KT.

The concluding three sections are dedicated to the evaluation and analysis of the previously discussed implementation. Sec. 3.3 focuses on the testing of algorithms associated with KT. In Sec. 3.4, we conduct a comparative analysis of executing a learned task with different task specifications (hierarchies). Lastly, Sec. 3.5 discusses an attempt to simulate a wiping task.

3.1 System Design

In this section, the software implementation of the system in Fig. 1.1 (repeated below for convenience) is explained in detail. The software architecture is presented in Sec. 3.1.1, i.e. which data structures are used, their design, and how they relate to the system (Fig. 1.1). For this work, a control algorithm was developed it's explained in Sec. 3.1.2 and it makes use of the mentioned data structures, and theory presented in Chap.2.

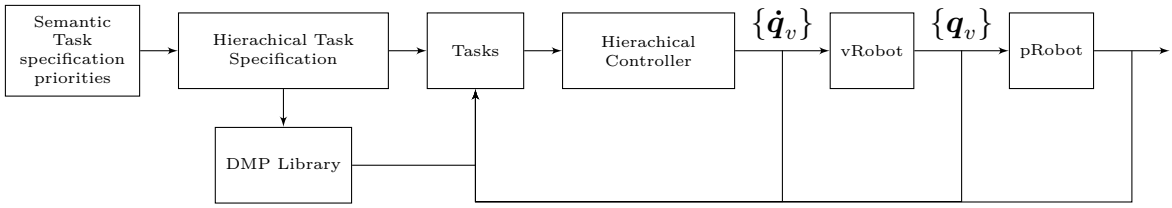


Figure 3.1: System design. The user provides a task specification. This specification is used to create an ordered list of task objects. A task object has a state, references, Jacobian, a compensation. The hierarchical controller processes the task objects to set up a generalized null space projector and solve a QP. This results in a joint velocity command sent to the virtual robot. Integrating the virtual robot joint state gives the position command for the physical robot.

3.1.1 Software Architecture

The software architecture has been designed such that the control loop (Sec.3.1.2, Alg. 3) can accommodate different hierarchies of tasks without the necessity to change any code. And, code for new task types is supposed to be easily added. The data structures used to accomplish this are named *RobotState* (Sec. 3.1.1.1), $Task$ (Sec. 3.1.1.2), $CanonicalSystem$ (Sec. 3.1.1.3) and $DurationSystem$ (Sec. 3.1.1.3). Below, we'll explain these.

3.1.1.1 RobotState

The *RobotState* structure holds variables, vectors, and matrices that indicate the state of the virtual and physical robots, and other helpers. The reason for having the states of both the physical and virtual robots is because task definitions (Sec. 1.1.1) can depend on both. Hence, if one wishes to define a task using some information currently not present in the *RobotState* structure, one could add it here. The C++ header for the *RobotState* data structure is shown in App. A, List. A.1.

A *RobotState* instance *robotState* is shared between the 'Tasks' block and the 'Hierarchical Controller' blocks (Fig 3.1), i.e. these blocks have read/write access to *robotState*. Since some functions used to set *robotState* members are asynchronously called, the read/write access to certain *robotState* members needs protection, hence, it has mutex member variables.

3.1.1.2 Task

The *Task* data structure (App. A, List. A.2) represents a single task (Sec. 1.1.1). The data structure has members representing task type, reference source, task state, control parameters, task Jacobian, task compensation, DMPs, and optimization parameters (Sec. 2.2.4). Some members of *Task* are function pointers, these are called $stateFcn$, $referenceFcn$, $taskJacobian$, $taskCompensation$ (Sec. 1.1.1). The functions pointed to, use a *RobotState* (Sec. 3.1.1.1) instance as input. Which functions, these function pointers point to depends on the task specification (type, reference source, etc). For example, if the type is a Cartesian pose, the state function sets the

task state from the virtual pose in the *RobotState* instance, if the reference source is the Force/Torque (F/T) sensor, the corresponding function pointer points to a function implementing admittance (3.2). If the reference source is a DMP, it uses the phase from the *RobotState* instance and the task member DMPs to generate the reference. The 'Tasks' block in Fig. 1.1 holds an ordered list of *task* instances.

To define new tasks, one can add functions that the task member functions can point to.

3.1.1.3 Canonical System and Duration System

The canonical system (Eq. (2.8)) is used to modulate the temporal evolution of a task (if a DMP generates the reference). The canonical system has a duration parameter $\tau \in \mathbb{R}$ that describes the desired total time duration. The *Duration* (App. A List. A.3), is a data structure with several parameters and a member function pointer that is used to modulate the desired time duration parameter in the canonical system. For example, if a task target moves away from the current task state, this function could increase the duration time [53].

The *Canonical_System* data structure (App. A, List A.4) is similar to *Duration*, the only difference is the name. They are separated for clarity, it makes it clear which members of *RobotState* to use and set.

3.1.2 Control Loop

In Alg. 3, the control loop developed in this work is presented. After initialization of the 'Tasks' block and the DMPs from the 'DMP Library', the control loop determines the system's (Fig 3.1) behavior. Alg. 3 is task-agnostic, reprogramming is not needed for different task types and hierarchies, but it requires the task objects to be correctly defined (Sec. 2.3). An explanation of the algorithm now follows.

When the controller is initialized, the virtual robot state $\mathbf{q}_v \in \mathbb{R}^n$ is set to equal the physical robot state $\mathbf{q}_p \in \mathbb{R}^n$. The virtual position ($\mathbf{p}_v^w \in \mathbb{R}^3$) and virtual unit quaternion ($\mathbf{Q}_v^w \in \mathbb{R}^4$) are computed using forward kinematics (Sec. 2.1). Calculation of the geometric Jacobian ($\mathbf{J}_v^w \in \mathbb{R}^{6\times n}$) is performed and it's used to determine the virtual end-effector linear and angular velocities ($\dot{\mathbf{p}}_v^w, \dot{\boldsymbol{\omega}}_v^w$). Reading and low-pass filtering of the wrench and joint sensor values is done in the function *readSensor()*, and the applied wrench ($\mathbf{W}_{ext}^w$) is estimated (Sec. 3.2.1). Task states ($\boldsymbol{\xi}_i, \dot{\boldsymbol{\xi}}_i$) and task references ($\boldsymbol{\xi}_i^r, \dot{\boldsymbol{\xi}}_i^r, \ddot{\boldsymbol{\xi}}_i^r$) for task $i \in \{1, \ldots, n_t\}$ are retrieved, which together with the control parameters ($M_i, D_i, K_i \in \mathbb{R}$) and control loop time constant $\Delta t \in \mathbb{R}$ are used to calculate the desired task velocity $\dot{\boldsymbol{\xi}}_i^*$. Populating the optimization matrices $\mathbf{H}$ and $\mathbf{f}$ (Sec. 2.4.2) is done with the task Jacobian, compensation, and frame matrix. Then, the projector $\boldsymbol{P}$ is calculated (Alg. 2), and the QP optimization problem is solved with the *OSQP* solver [60] which results in an augmented optimal virtual joint velocity vector $\dot{\mathbf{q}}_v' \in \mathbb{R}^{nn_t}$. Integrating the virtual joint state $\mathbf{q}_v$ by adding $\dot{\mathbf{q}}_v'$ multiplied with the nullspace projector and time constant Δt, results in the joint state given as a command to the physical robot joint position controllers. Finally,

the duration and phase states are updated (Sec. 3.1.1.3) which is relevant when task references are generated from DMPs.

Algorithm 3 Control Loop

procedure ControlLoop
  Setting the virtual and physical robot states
  $robotState.\mathbf{p}_v^w$, $robotState.\mathbf{Q}_v^w \leftarrow fkEeWrtTorso(\mathbf{q_v})$
  $\tilde{\mathbf{Q}} \leftarrow \mathbf{Q}_v^w * \bar{\mathbf{Q}}_0^w$
  $robotState.\mathbf{J}_v^w \leftarrow JacobianEeWrtTorso(\mathbf{q_v})$
  $robotState.\boldsymbol{\eta} \leftarrow log(\tilde{\mathbf{Q}})$
  $robotState.\dot{\mathbf{p}}_v^w$, $robotState.\omega_v^w \leftarrow \mathbf{J}_v^w \dot{\mathbf{q}}_v$
  $\dot{\mathbf{Q}}_v^w \leftarrow \frac{1}{2} \begin{bmatrix} 0 \\ \omega_v^w \end{bmatrix} * \mathbf{Q}_v$
  $robotState.\dot{\eta} \leftarrow \mathbf{J_Q}(\tilde{\mathbf{Q}})(\dot{\mathbf{Q}}_v * \bar{\mathbf{Q}}_0)$ ▷ Sec. 2.1.5
  $readFTSensor()$ ▷ Set variables related to the physical robot
  $n_t \leftarrow tasks.size()$
  for $i \leftarrow 0$ to $n_t - 1$ **do**
    $task \leftarrow tasks[i]$
    M, D, $K \leftarrow task.controlParams$
    $\boldsymbol{\xi}_i$, $\dot{\boldsymbol{\xi}}_i \leftarrow task.stateFcn()$
    $\boldsymbol{\xi}_i^r$, $\dot{\boldsymbol{\xi}}_i^r$, $\ddot{\boldsymbol{\xi}}_i^r \leftarrow task.referenceFcn()$
    $\dot{\boldsymbol{\xi}}_i^* = \dot{\boldsymbol{\xi}}_i + (M_i \ddot{\boldsymbol{\xi}}_i^r + D_i(\dot{\boldsymbol{\xi}}_i^r - \dot{\boldsymbol{\xi}}_i) + K_i(\boldsymbol{\xi}_i^r - \boldsymbol{\xi}_i))\Delta t$
    $\mathbf{J}_{task} = task.taskJacobian()$
    $\dot{\boldsymbol{\xi}}_{comp} = task.taskCompensation()$
    $\mathcal{F} \leftarrow task.frameFcn()$
    $taskVel[i] \leftarrow \dot{\boldsymbol{\xi}}_i^* + \dot{\boldsymbol{\xi}}_{comp}$ ▷ Sec. 1.1.1
    $taskJac[i] \leftarrow \mathcal{F}^\top \mathbf{J}_{task}$ ▷ Sec. 2.3.1
    $\mathbf{H}[ind : ind + q_{size}, ind : ind + q_{size}] \leftarrow taskJac[i]^\top taskJac[i]$ ▷ Sec. 2.4.2
    $\mathbf{f}[ind : ind + q_{size}] \leftarrow taskJac[i]^\top taskVel[i]$ ▷ Sec. 2.4.2
  end for
  $\mathbf{H} \leftarrow \mathbf{H} + \rho\mathbf{I}$
  $\mathbf{P} \leftarrow nullSpaceProj(taskJac)$ ▷ Alg. 2
  $\dot{\mathbf{q}}_v' \leftarrow OSQP(2\mathbf{H}, -2\mathbf{f}, \mathbf{P}, constraints)$
  $\mathbf{q}_v \leftarrow \mathbf{q}_v + \mathbf{P}\dot{\mathbf{q}}_v'\Delta t$
  $robotState.\tau, robotState.\dot{\tau}, robotState.\ddot{\tau} \leftarrow durationFcn()$
  $robotState.x, robotState.\dot{x}, robotState.\ddot{x} \leftarrow canSysFcn()$
end procedure

3.2 Admittance Interface for Kinesthetic Teaching

The physical robot (Tiago [61]) is controlled via a virtual robot (kinematic model). A kinematic model has no concept of force and torque (Sec. 2.1.2), but, in kinesthetic teaching (KT) the robot must react to physical interaction. How then to influence

the kinematic model via physical interaction? An admittance (force to motion) interface is needed. In Sec. 3.2.1 the hardware of the interface is explored and the software, i.e. the admittance algorithm is explained in Sec. 3.2.2.

3.2.1 FT-sensor Calibration

Tiago [61] is equipped with an F/T sensor, it's mounted in its wrist (Fig. 3.2), which we'll use in our admittance interface. The basic theory for the admittance algorithm was developed in Sec. 2.5, and it was noted that sensor measurements due to gravity and motion have to be filtered out. Static, low-frequency, and high-frequency noise are also present in the sensor data, hence, for a correct response by the robot, gravity, motion, and noise factors have to be accounted for. PAL Robotics has provided a custom-made filter for this purpose, along with a calibration procedure. However, the PAL filter and calibration procedure were not available in the robot simulation environment. Hence, to test our KT interface in simulation it was necessary to develop a filter (Sec. 3.2.2) which we'll also compare to the PAL filter (Sec. 3.3).

The low-frequency noise in Tiago's F/T sensor is mostly due to temperature and pressure [67], however, Tiago has custom software and a calibration procedure to model this noise. When the calibration procedure has been run, one can retrieve several parameters and a data stream that corrects for low-frequency noise, gravity, and acceleration. This data stream is updated at a frequency of 50Hz [67].

Among the parameters retrievable after the calibration, are estimations of the end-effector center of mass (CoM) w.r.t. to the sensor frame ($\mathbf{p}^s_{CoM} \in \mathbb{R}^3$), mass ($m \in \mathbb{R}$), gravitational acceleration ($g \in \mathbb{R}$) as well as force ($\mathbf{F}^s_{offset} \in \mathbb{R}^3$) and torque ($\mathbf{T}^s_{offset} \in \mathbb{R}^3$) offsets in the sensor frame (s). The force and torque offsets are the mentioned static noise. Note, these parameters are needed in Algs.4 and 5 in Sec. 3.2.2.

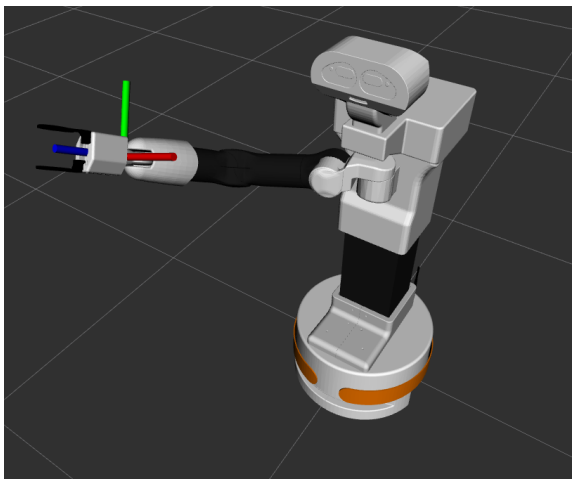


Figure 3.2: The frame in the wrist indicates the F/T sensor location.

3.2.2 Admittance Interface and Control

Provided in this section is an explanation of how wrench measurements are transformed into reference linear and angular velocities for the robot end-effector. Also, the filtering of end-effector motion, mass, static, and high-frequency noise from the sensor itself (Sec. 3.2.1), is described.

We exploit the calibration (Sec. 3.2.1) procedure to find force ($\mathbf{F}^s_{offset} \in \mathbb{R}^3$) and torque ($\mathbf{T}^s_{offset} \in \mathbb{R}^3$) offsets in the sensor frame (s), end-effector center of mass (CoM) w.r.t. to the sensor frame($\mathbf{p}^s_{CoM} \in \mathbb{R}^3$), mass ($m \in \mathbb{R}$), gravitational acceleration ($g \in \mathbb{R}$), these are used in Algs.4 and 5. Unlike the PAL filter, our filter does not compensate for the low-frequency noise due to temperature and pressure [67]. On the other hand, high-frequency noise is mitigated by low-pass filtering measurements. A Butterworth filter (Sec. 2.6) of order 5 (iir_ros repository `https://github.com/seanyun/iir_ros`) is used.

A description of the admittance algorithm Alg. 4 now follows. In Alg. 4, $\mathbf{W}^s \in \mathbb{R}^6$ is the measured and low-pass-filtered wrench, expressed in the sensor frame (s), $\mathbf{R}^w_s$ is the rotation matrix, that describes the sensor frame relative to the reference frame (w), $\mathbf{q}_p \in \mathbb{R}^n$ is the low-pass filtered physical joint measurement, $\mathbf{W}^s_{offset} \in \mathbb{R}^6$ is a constant wrench correction given by the calibration procedure (Sec. 3.2.1) which is the zero vector in the simulation environment. The function $wrenchDueToGravityAndAcc()$ (Alg. 5) estimates the wrench on the sensor due to motion and gravity (Sec. 2.5) and $\mathbf{W}^w_{ext}$ is the estimated applied wrench on the end-effector. The thresholds $f_{threshold}, \tau_{threshold}$ are user-defined scalars and denote force and torque norm thresholds. The wrench thresholds are necessary since the estimated wrench isn't perfect, due to sensor noise and model errors. If the thresholds are not used the estimation error will induce movement in the physical robot (since a velocity command is generated) even when no wrench is applied (Fig. 3.3). Hence, thresholds are used to ensure that the robot reacts only if the applied wrench estimation is large relative to the accumulative effect of noise and model errors. Finally, Alg. 4 returns a reference pose velocity ($\dot{\mathbf{x}}_{ref} \in \mathbb{R}^6$). The scalars A_{force} and A_{torque} represent the admittance for force and torque respectively. In the next paragraph the filter algorithm for motion and gravity Alg. 5 is detailed.

In Alg. 5, $\mathbf{q} \in \mathbb{R}^n$ and $\dot{\mathbf{q}} \in \mathbb{R}^n$, are low-pass filtered sensor readings of joint position and velocity. The rotation matrix $\mathbf{R}^w_s \in \mathbb{R}^{3\times3}$, describes the orientation of the sensor frame w.r.t. the reference frame w. $\mathbf{J}^w_s \in \mathbb{R}^{6\times n}$ is the Jacobian of the sensor frame w.r.t. the reference frame. Linear and angular velocities $\mathbf{p}^w_s, \boldsymbol{\omega}^w_s \in \mathbb{R}^3$ of the sensor are calculated with the Jacobian $\mathbf{J}^w_s$ and joint velocities $\dot{\mathbf{q}}$. To account for the wrench applied on the sensor by the end-effector, the end-effector CoM acceleration $\ddot{\mathbf{p}}^w_{CoM}$ is needed (Sec. 2.5). Using Euler backward differentiation, we approximate $\ddot{\mathbf{p}}^w_{CoM}$. Hence, before estimating the current CoM velocity $\dot{\mathbf{p}}^w_{CoM} \in \mathbb{R}^3$, we set $\dot{\mathbf{p}}^w_{CoM,prev} \in \mathbb{R}^3$ to the previously estimated value (initialized as the zero vector). Then $\dot{\mathbf{p}}^w_{CoM}$ is calculated from the sensor velocities by utilizing the known CoM vector $\mathbf{p}^s_{CoM}$(given by the calibration) and the control loop time constant Δt. Lastly, the applied force $\mathbf{F}^w_{pred} \in \mathbb{R}^3$ and torque $\mathbf{T}^w_{pred} \in \mathbb{R}^3$ estimations by the end-effector on the sensor are

Algorithm 4 Admittance Control

procedure AdmittanceControl($\mathbf{W}^s$)
  $\mathbf{W}^w_{ext} \leftarrow \mathbf{R}^w_s(\mathbf{q}_p)\mathbf{W}^s - wrenchDueToGravityAndAcc() - \mathbf{R}^w_s(\mathbf{q}_p)\mathbf{W}^s_{offset}$ ▷ Alg. 5
  $\mathbf{f}^w_{ext} \leftarrow \mathbf{W}^w_{ext}(1:3)$
  $\boldsymbol{\tau}^w_{ext} \leftarrow \mathbf{W}^w_{ext}(4:6)$
  $f_{norm} \leftarrow \|\mathbf{f}^w_{ext}\|$
  $\tau_{norm} \leftarrow \|\boldsymbol{\tau}^w_{ext}\|$
  if $f_{norm} > f_{threshold}$ **then**
    $\dot{\mathbf{p}}^w_{ref} \leftarrow A_{force}(f_{norm} - f_{threshold})\mathbf{f}^w_{ext}/f_{norm}$
  else
    $\dot{\mathbf{p}}^w_{ref} \leftarrow \mathbf{0}$
  end if
  if $\tau_{norm} > \tau_{threshold}$ **then**
    $\omega^w_{ref} \leftarrow A_{torque}(\tau_{norm} - \tau_{threshold})\boldsymbol{\tau}^w_{ext}/\tau_{norm}$
  else
    $\omega^w_{ref} \leftarrow \mathbf{0}$
  end if
  $\dot{\mathbf{x}}^w_{ref} = \begin{bmatrix}\dot{\mathbf{p}}^w_{ref} \\ \omega^w_{ref}\end{bmatrix}$
  return $\dot{\mathbf{x}}^w_{ref}$
end procedure

calculated by also accounting for the effect of gravity where the end-effector mass m and gravitational acceleration g are given by the calibration procedure. Note, that the induced torque due to the end-effector inertia is neglected.

Algorithm 5 Wrench Prediction

procedure wrenchDueToGravityAndAcc
  $\mathbf{q} \leftarrow low - pass(\mathbf{q}_p)$
  $\dot{\mathbf{q}} \leftarrow low - pass(\dot{\mathbf{q}}_p)$
  $\mathbf{R}^w_s \leftarrow fkSensorWrtTorso()$
  $\mathbf{J}^w_s \leftarrow sensorJacobian(\mathbf{q})$
  $\begin{bmatrix}\mathbf{p}^w_s & \boldsymbol{\omega}^w_s\end{bmatrix}^\top \leftarrow \mathbf{J}^w_s\dot{\mathbf{q}}$
  $\dot{\mathbf{p}}^w_{CoM,prev} \leftarrow \dot{\mathbf{p}}^w_{CoM}$
  $\dot{\mathbf{p}}^w_{CoM} \leftarrow \mathbf{p}^w_s + \omega^w_s \times \mathbf{R}^w_s\mathbf{p}^s_{CoM}$
  $\mathbf{F}^w_{pred} \leftarrow m(\mathbf{g}^w - (\dot{\mathbf{p}}^w_{CoM} - \dot{\mathbf{p}}^w_{CoM,prev})/\Delta t)$ ▷ Euler backward differentiation
  $\mathbf{T}^w_{pred} \leftarrow \mathbf{p}^w_{CoM} \times \mathbf{F}^w_{pred}$ ▷ End-effector inertia is neglected
  return $\mathbf{F}^w_{pred}, \mathbf{T}^w_{pred}$
end procedure

3.3 Kinesthetic Teaching and Filter Comparison

Two filters - one from PAL and ours - for end-effector motion, mass, and sensor noise are compared in this section. First, in Sec. 3.3.1, the filters are compared during contact-free motion. Filter comparison during KT is then undertaken in Sec. 3.3.2, where the velocity reference is generated by Alg.4. The reference tracking is evaluated as well.

3.3.1 Contact-free Motion

In the context of this work, contact-free motion means that the robot end-effector moves without environmental contact. But, as stated, the F/T sensor is located in the wrist and unfiltered sensor readings include effects due to gravity, motion, and noise (Sec. 3.2.1). The ideal filter perfectly compensates for these effects and contact-free motion, therefore, indicating how well a filter approximates the ideal.

In Fig. 3.3 we compare our filter Alg. 5 and the custom-made filter from PAL robotics during contact-free motion. There is more high-frequency noise in the PAL filter than ours. Hence, using the PAL filter would necessitate larger thresholds in Alg. 4 but this could be mitigated by low-pass filtering. On the other hand, our filter is delayed (≈ 0.01s, see Fig. 3.6) and this is due to low-pass filtering (Sec. 2.6). Lowering the order of the filter would lessen the delay. Force errors of several Newtons are observed but negligible absolute torque errors, which justify the omission of end-effector inertia (Sec. 2.5). However, the omission of inertia is not appropriate if the end-effector is holding an object or if there is significant angular acceleration. Static noise is observed to be configuration-dependent which is most evident in the force y-direction. For example, the y-direction force is less than 0.5N before the motion starts but about -1N when the motion is finalized. Neither filter is concluded to be close to ideal, hence, the thresholds in in Alg. 4 are necessary. See the video `https://youtu.be/t0Wgd-3bS7c` for a recording of this data.

3.3.2 Kinesthetic Teaching

This section presents comparisons between our and the PAL filter during KT, i.e. comparison of predicted applied wrenches on the end-effector. During KT, Alg. 4 uses the predicted applied wrench to generate end-effector velocity references and we'll evaluate the reference tracking. References are assumed to be good approximations of teacher intent, and in this case, the reference tracking would indicate the intuitiveness of the KT interface. Observe that the references are generated for the virtual robot, i.e. the kinematic control model. Hence, it's also important to compare the virtual and physical robot motion. An analysis of the effects on tracking performance due to joint limits for the kinematic control model is provided.

Data presented here were collected while moving the end-effector to a stack of cubes such that the gripper encloses the cubes, and then the end-effector was pushed down and up to trace the stack, see Fig. 3.4 or the video `https://youtu.be/t0Wgd-3bS7c`.

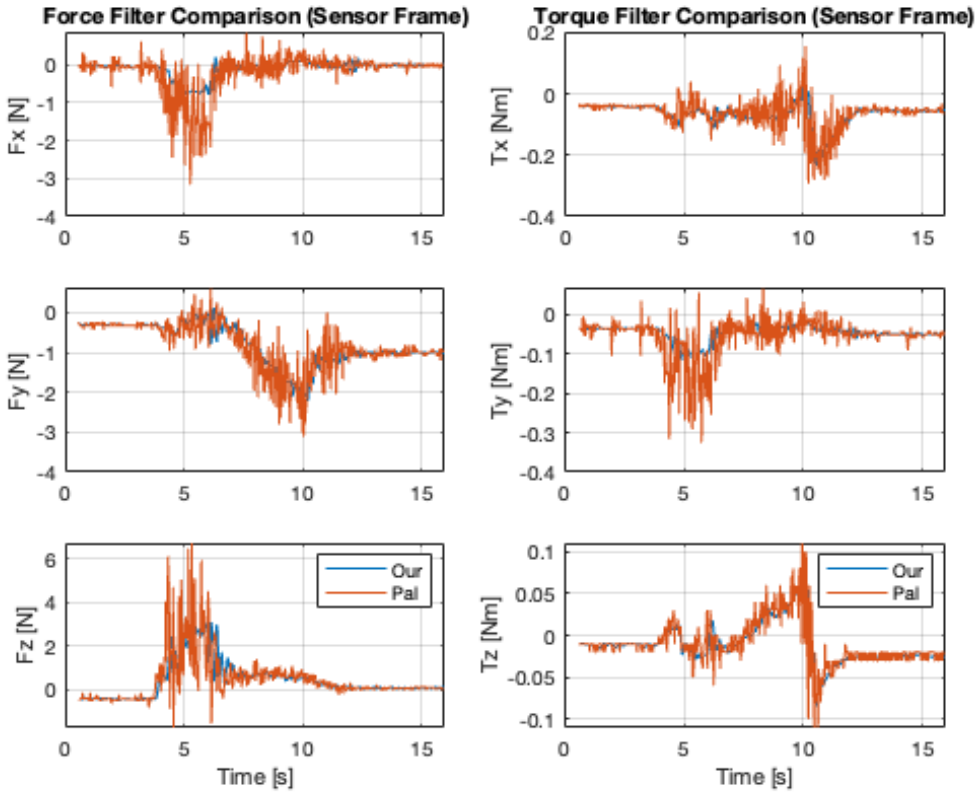


Figure 3.3: Comparison between our filter (blue) and the custom-made filter from PAL-Robotics (orange). The data is collected during contact-free motion. The ideal filter would produce values of zero force and torque since no wrench is applied.

Filter comparisons are presented in Figs. 3.5, and 3.6. Reference tracking data is found in Fig. 3.7 and data for virtual vs physical joint position and velocities are found in Figs. 3.8 and 3.9. The KT task specification is found in Table. 3.1. In App. C, some data corresponding to KT experiments with more restrictive joint velocity limits is presented and which can also be seen in the video.

Figure 3.4: Depiction of the trace cube stack task. The task was to teach the robot to trace the stack of cubes by first going to a grasp pose for cube 'A' and then tracing the stack down to cube 'B' and up again.

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pose	F/T sensor	(0,80,0)	$\mathcal{F}_1 = \boldsymbol{I}_6$ $(m = 6)$	
2	joint pos	$\mathbf{0}_{8\times1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table 3.1: Kinesthetic Teaching Task Specification. The highest priority task is the full Cartesian end-effector pose which is indicated by the task type and the frame. The reference source for the Cartesian pose task is the Force/Torque (F/T) sensor from which reference Cartesian velocities are retrieved via Alg. 4. The only non-zero control parameter is the damping D, hence, the task is velocity-controlled. No inequality constraints are considered since the operator is assumed to make sure the robot stays in the workspace and avoids collisions. The secondary task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is also velocity-controlled since the only non-zero control parameter is the damping D, and zero velocity is the reference. The inequality constraints from Eq. (2.54) are active for this task.

In Fig. 3.5, we see the comparison between our filter Alg. 5 and the PAL filter during KT. As in Sec. 3.3.1, the filters are comparable, but there is a slight delay in ours and more high-frequency noise in theirs. However, as the magnitude of the applied wrench increases, the impact of the high-frequency noise becomes negligible. For example, the high-frequency noise is visible between 25s-30s in the force x-direction and between 10s-15s in the torque x-direction. Fig. 3.6 shows a zoomed-in version where the delay in our filter is clear.

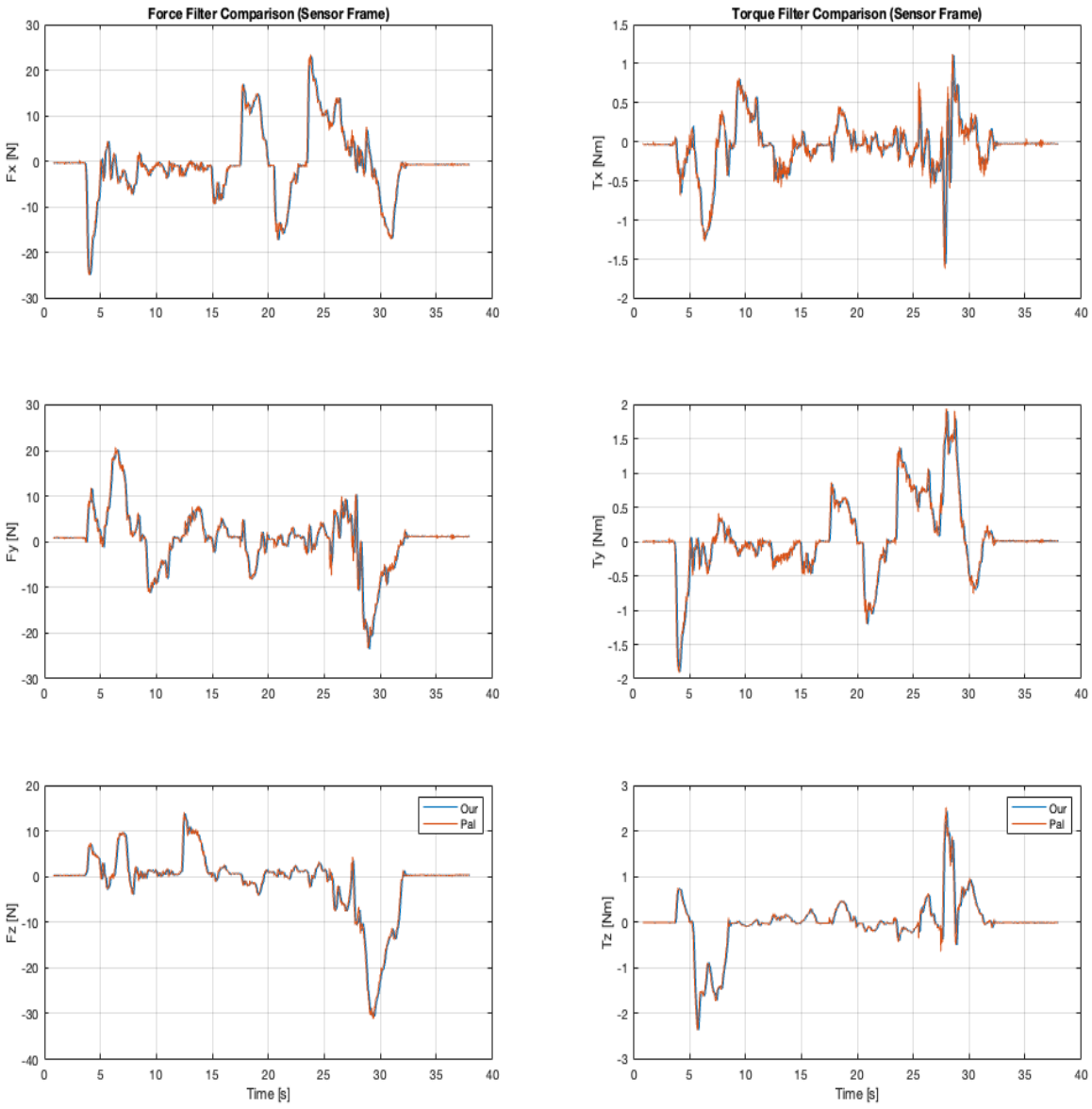


Figure 3.5: Comparison between the custom-made PAL filter (orange) and ours (blue) during KT. Applied force predictions are graphed in the right column and in the left column, are applied torque predictions. At this scale, it's hard to tell any difference, to see the time delay in our filter, see Fig. 3.6.

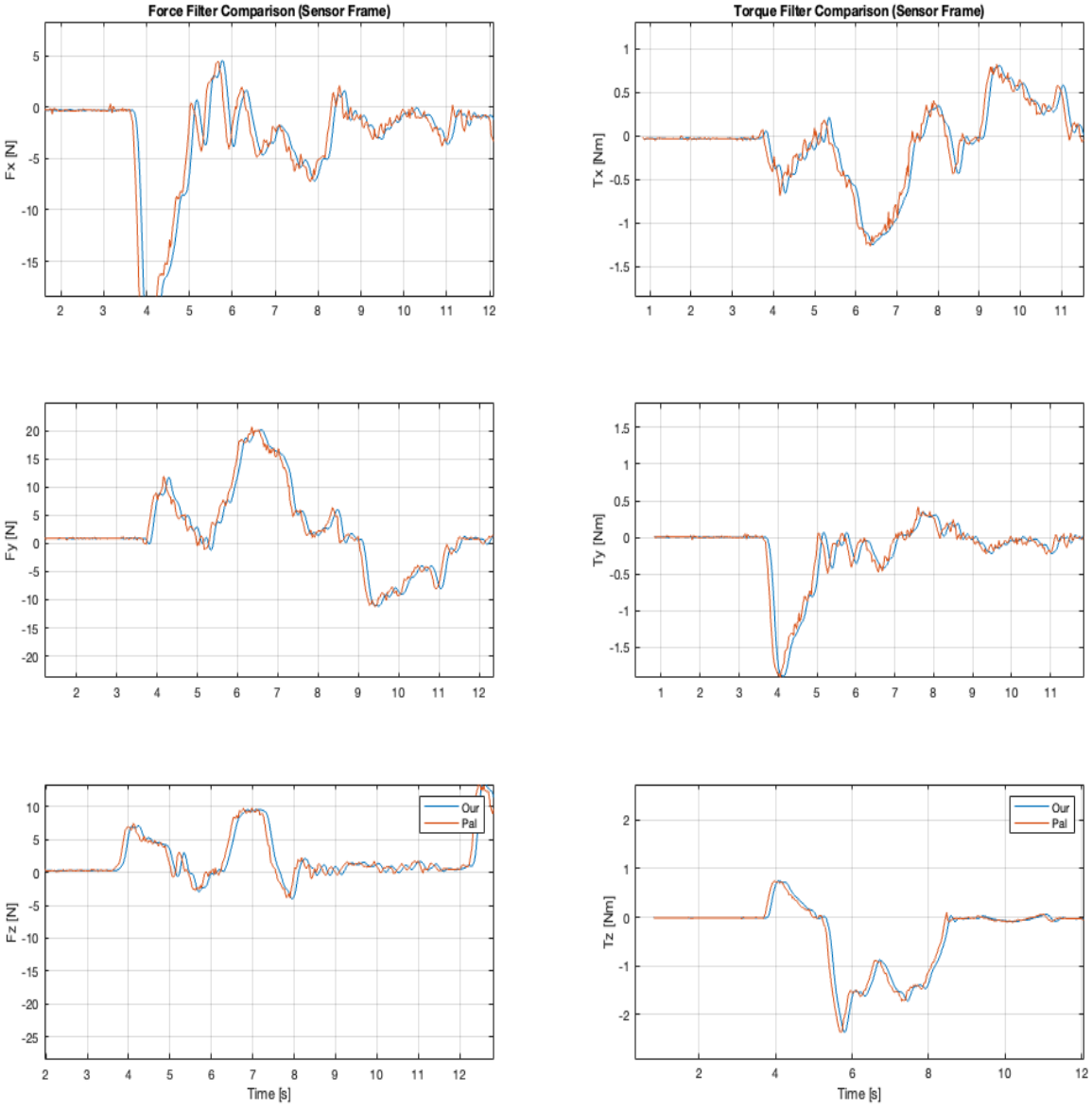


Figure 3.6: Comparison between the filter from PAL Robotics (orange) and ours (blue) during KT. The graphs in the right column are applied force predictions and in the left column, are applied torque predictions. We see that our filter is about 0.01s delayed relative to the PAL filter.

Reference tracking during KT is depicted in Fig. 3.7. Good tracking is observed for the virtual robot, except around the 10s mark. This can be explained by analyzing Figs. 3.8 and 3.9 that graph joint positions, velocities, and the corresponding constraints of the virtual and physical robots. Several of the virtual joint velocities are at their limits around the 10s mark. This means the optimizer couldn't find joint velocity solutions to minimize the tracking errors without violating limits. Note, that the robot has 8 DoF and the high priority pose task is 6 DoF, hence, the optimizer should "struggle" to find good solutions when three or more limits are reached. Around the 10s mark, up to 5 joints are at their velocity limit. It's therefore not surprising to see deteriorated tracking. Let's now address the difference in motion between virtual and physical robots.

In Fig. 3.7, we observe that the physical robot velocities are significantly slower than the virtual, and delayed. In the video `https://youtu.be/t0Wgd-3bS7c`, we also don't observe severe oscillations as seen in the virtual robot velocities. This corresponds well with the joint velocity measures in Figs. 3.8 and 3.9. Why a significant difference in virtual and physical motion? The admittance algorithm directly translates wrenches to velocities, which leads to rapid velocity and acceleration changes (see Fig. 3.5) and inertial effects become significant when acceleration and jerk increase. In this case, the virtual robot which has no inertia (kinematic model) fails to be a good approximation, it will move faster than the physical robot. Yet, the physical robot ends up moving in the right direction and it's quite intuitive (see video). This is explained by the similar joint positions between the virtual and physical robots and that our control algorithm provides the virtual joint positions to the physical controllers. The velocity oscillations are averaged out.

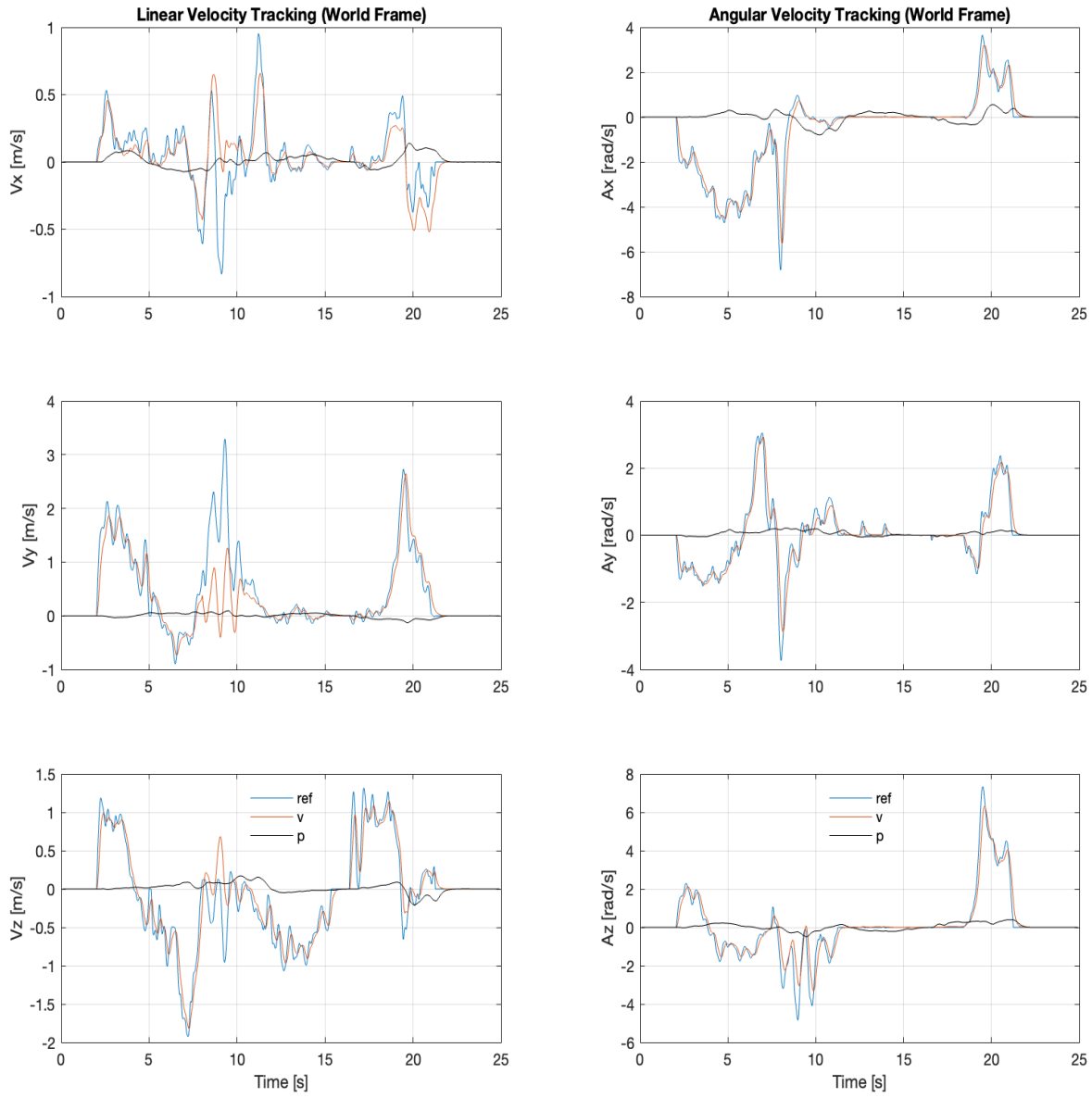


Figure 3.7: Reference tracking during KT. Linear velocity tracking is depicted in the right-column graphs. The left column graphs depict angular velocity tracking. Blue lines are references for the virtual robot and the orange lines are the velocities of the virtual robot. Black lines depict the motion of the physical robot end-effector.

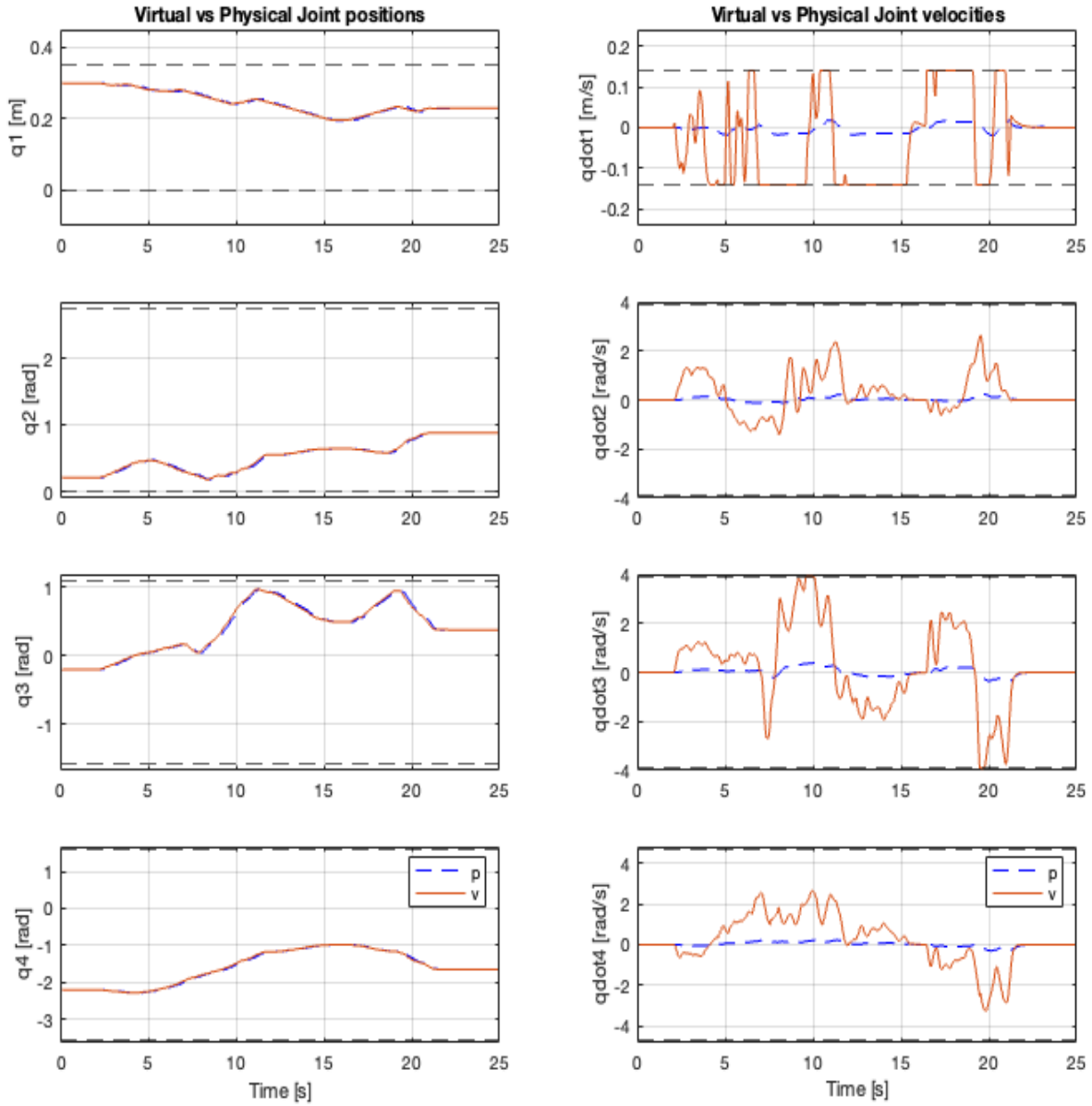


Figure 3.8: In the right column, the first four virtual (orange) vs physical (blue dashed) joint positions are compared. Upper and lower black dashed lines depict upper and lower joint position limits. In the left column, the virtual (orange) vs physical (blue dashed) joint velocities are compared. Upper and lower black dashed lines represent the upper and lower joint velocity limits imposed on the virtual robot.

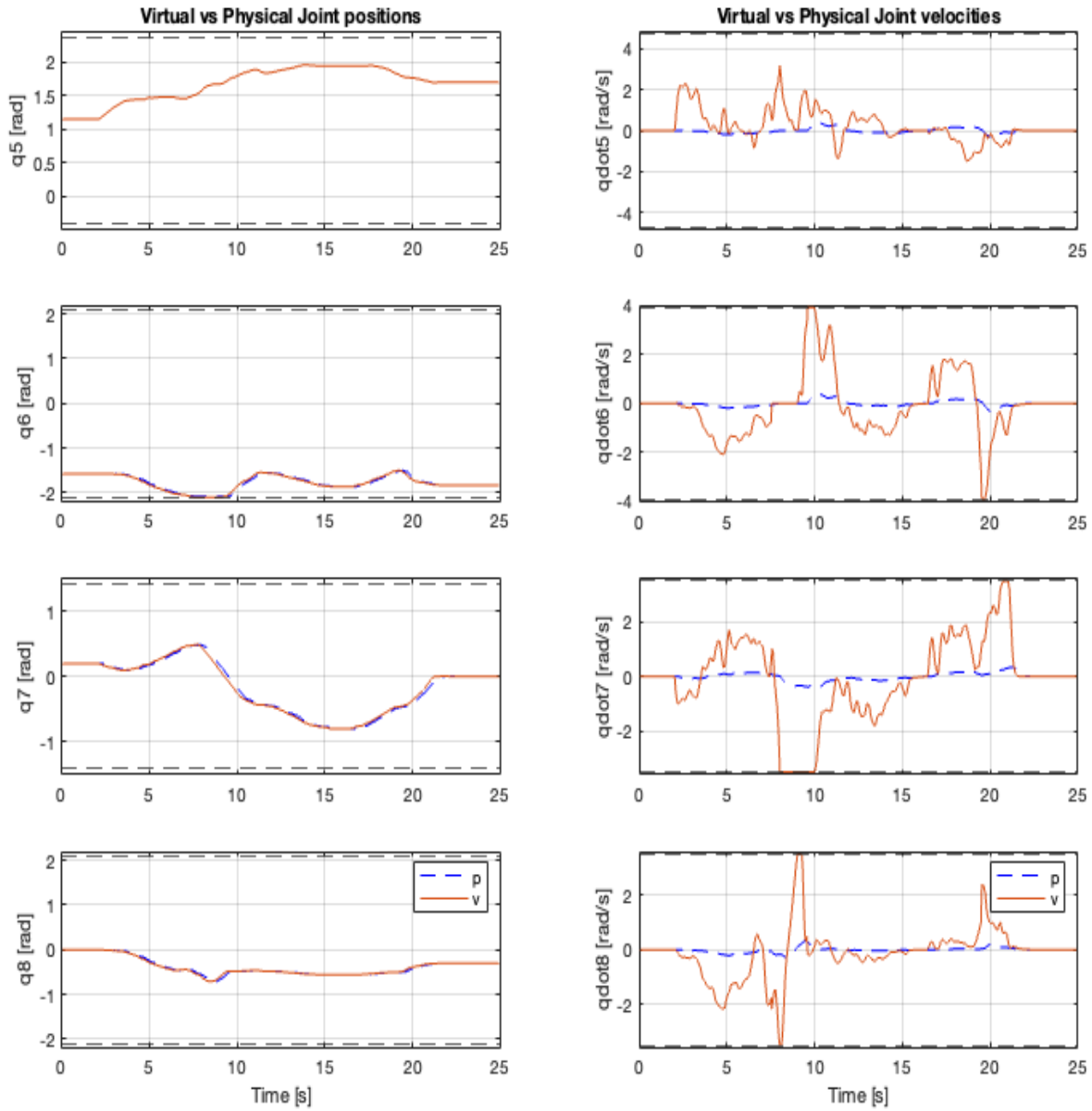


Figure 3.9: In the right column, the last four virtual (orange) and physical (blue dashed) joint positions are compared. Upper and lower black dashed lines depict the upper and lower joint position limits imposed on the virtual robot. In the left column, virtual and physical joint velocities are compared. Upper and lower black dashed lines represent the upper and lower joint velocity limits.

3.4 Comparing hierarchies

Here we validate that our system supports hierarchical control where task references are generated from DMPs, trained using KT. Tracking performance is evaluated and the preservation of inequality constraints is verified. We compare task tracking with different task specifications (hierarchies), three hierarchies are investigated, see Secs. 3.4.1,3.4.2 and 3.4.3. For each hierarchy, the Cartesian virtual and physical end-effector positions and velocities are compared to the corresponding references. Eqs. (B.4), (B.5) and (B.6) give the lower, and upper Cartesian space position and velocity limits used for the position tasks below. Virtual and physical joint positions and velocities are plotted to ensure that joint inequality constraints are respected.

$$\underline{\mathbf{p}}_{ee}^{w} = \begin{bmatrix} -0.15 & -3.0 & -1.5 \end{bmatrix}^{\top} \tag{3.1}$$

$$\overline{\mathbf{p}}_{ee}^{w} = \begin{bmatrix} 3.0 & 3.0 & 3.0 \end{bmatrix}^{\top} \tag{3.2}$$

$$\underline{\dot{\mathbf{p}}}_{ee}^{w} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \end{bmatrix}^{\top} = -\overline{\dot{\mathbf{p}}}_{ee}^{w} \tag{3.3}$$

For these experiments, the robot was taught to move the end-effector into a grasping pose for a cube, see Fig. B.1. The teaching and execution of this task can be seen in the video, and data relating to the teaching is found in App. B. Teaching and execution data for two other taught tasks (see video) are in App. B.

Figure 3.10: Depiction of the grasping task. The task was to teach the robot to get into a grasping pose for the red cube marked by a red 'A'.

3.4.1 Position Task

This section analyzes data for the execution of the taught grasp pose task with the task hierarchy given by Table. 3.2. An analysis of the results for a different task hierarchy is given in the next section.

Task Specification Table					
Priority	Type	Ref. source	Ctrl.p (M,D,K)	Frame	ineq. constr
1	cart. pos	DMP-MPC	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{I}_3 \\ \mathbf{0}_3\end{bmatrix}$ $(m=3)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	joint pos	$\mathbf{0}_{8\times 1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m=8)$	Eq. (2.54)

Table 3.2: The highest priority task is the Cartesian space end-effector position, $\mathbf{p}_{ee}^{w}$, indicated by the type and the frame. The secondary task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

In the example presented here, the robot executes a position task (high priority) combined with a joint position task (low priority). The figures below show that the robot is capable of tracking the end-effector position and linear velocity references while maintaining all upper and lower limits. We observe that both the position and velocity limits of joint q_1 are occasionally reached, however, the tracking performance isn't affected since the position task dimension is 3, therefore, the tracking, at any time instant, shouldn't deteriorate as long as no more than 5 limits are reached (not considering internal and algorithmic singularities). This task doesn't include orientation, as expected, reference orientations and reference angular velocities are not tracked. Virtual and physical robots match closely. Pose control is validated in the next experiment.

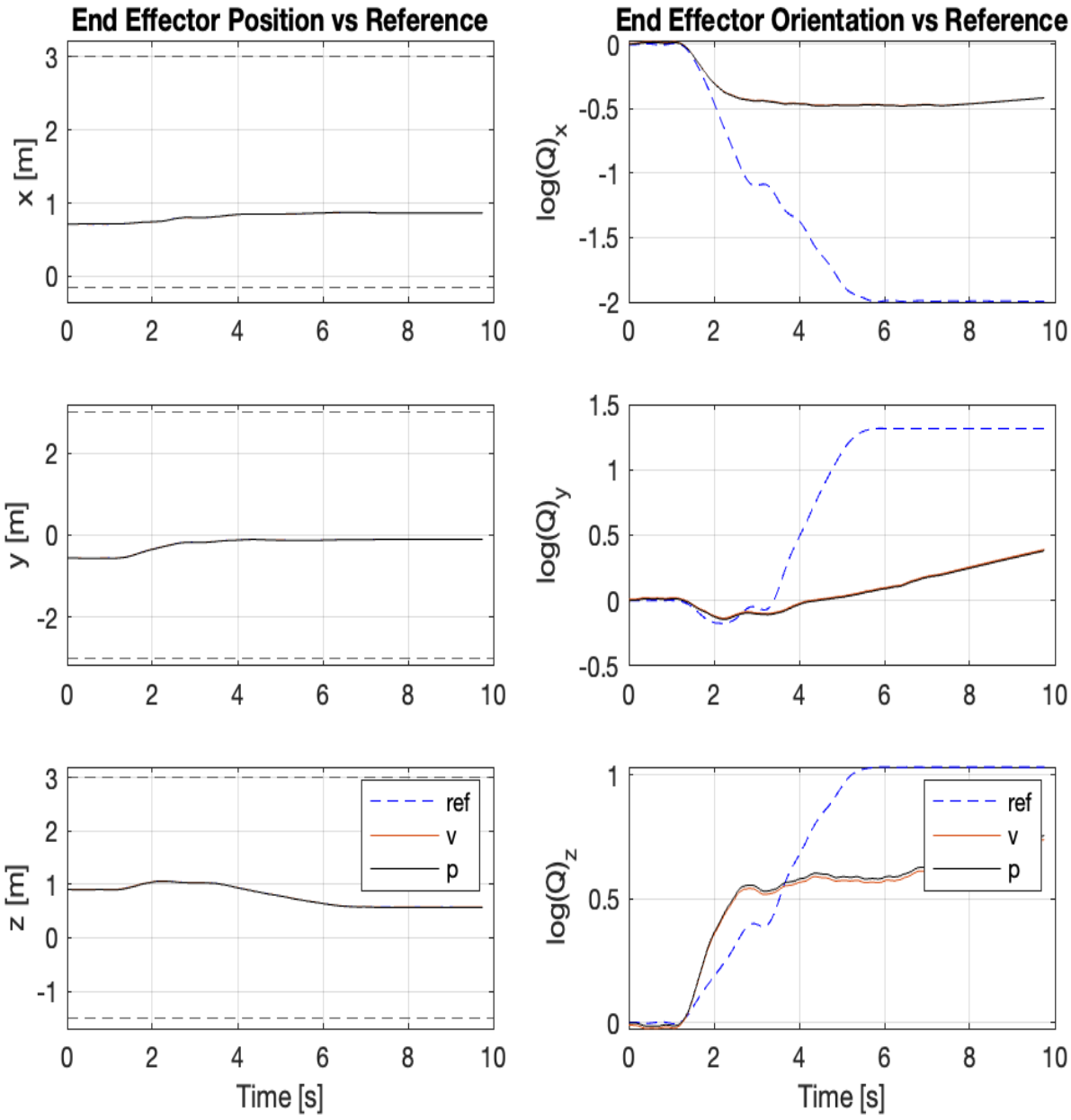


Figure 3.11: The task specification is given by Table. 3.2. The graphs in the left column depict the DMP-MPC-generated position reference (dashed blue), virtual (orange), and physical (black) position. Upper and lower dashed lines denote the upper and lower position limits, Eqs. (B.4), (B.5). The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec. 2.1.5) (blue dashed line), virtual (orange), and physical (black) orientation. As expected, the orientation reference is not tracked, compare this with the corresponding graphs in Figs. 3.15, 3.19.

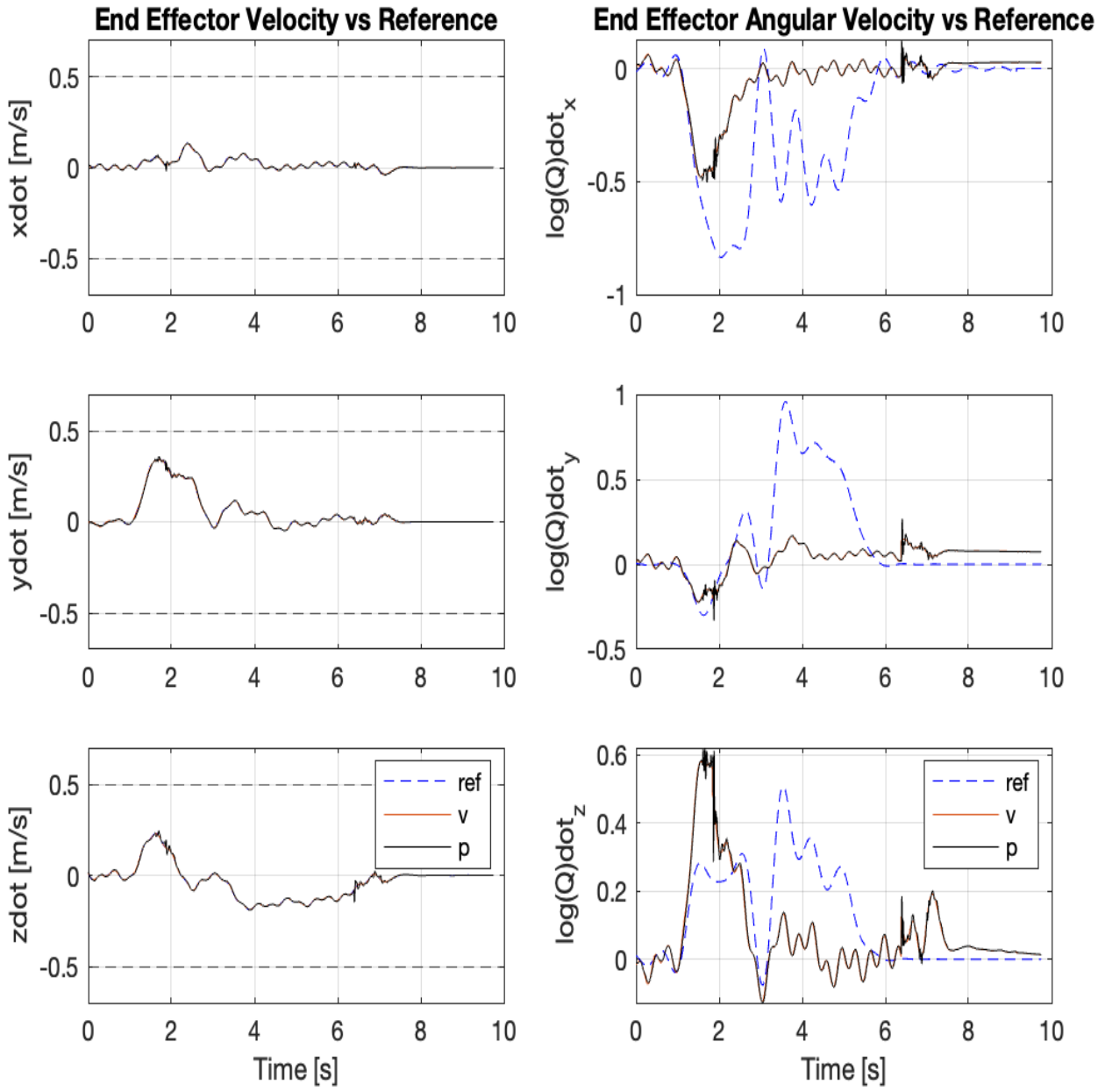


Figure 3.12: The task specification is given by Table. 3.2. The graphs in the left column depict the DMP-MPC-generated velocity reference (dashed blue), virtual (orange), and physical (black) velocity. Upper and lower dashed lines in the right column graphs denote upper and lower velocity limits Eqs. (B.4), (B.5). The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec. 2.1.5) (blue dashed line) and corresponding virtual (orange) and physical (black) velocities. Since orientation is not part of the task description, the reference is not tracked, compare this with the corresponding graphs in Figs. 3.16, 3.20.

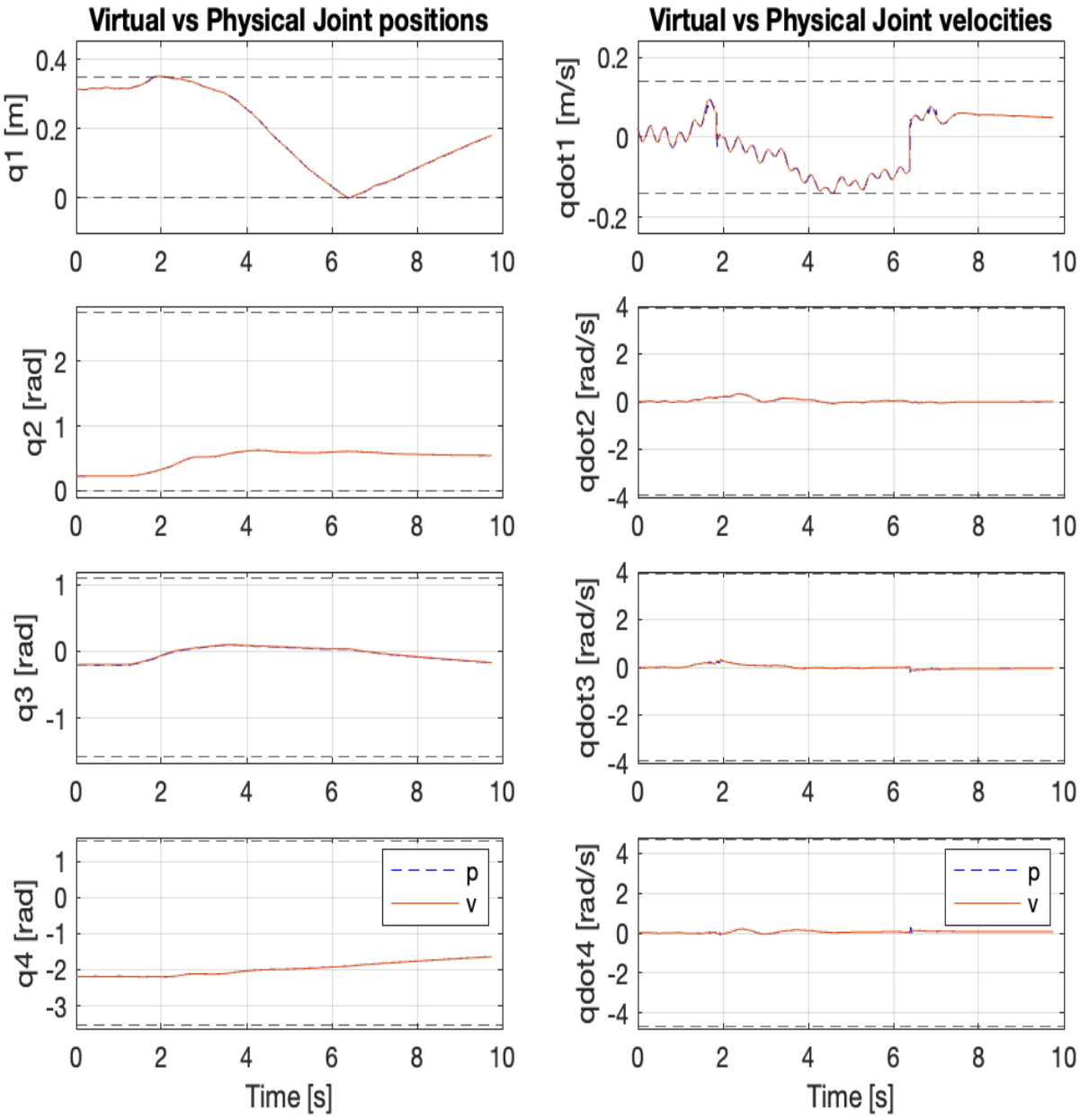


Figure 3.13: The task specification is given by Table. 3.2. The graphs in the right column show virtual (orange) and physical (dashed blue) joint positions for the first four joints; dashed lines denote virtual (and physical) upper and lower joint position limits. The left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote virtual velocity limits. The first joint is prismatic and the rest are revolute (Sec. 2.1.2). The corresponding graphs for the last four joints are seen in Fig. 3.14.

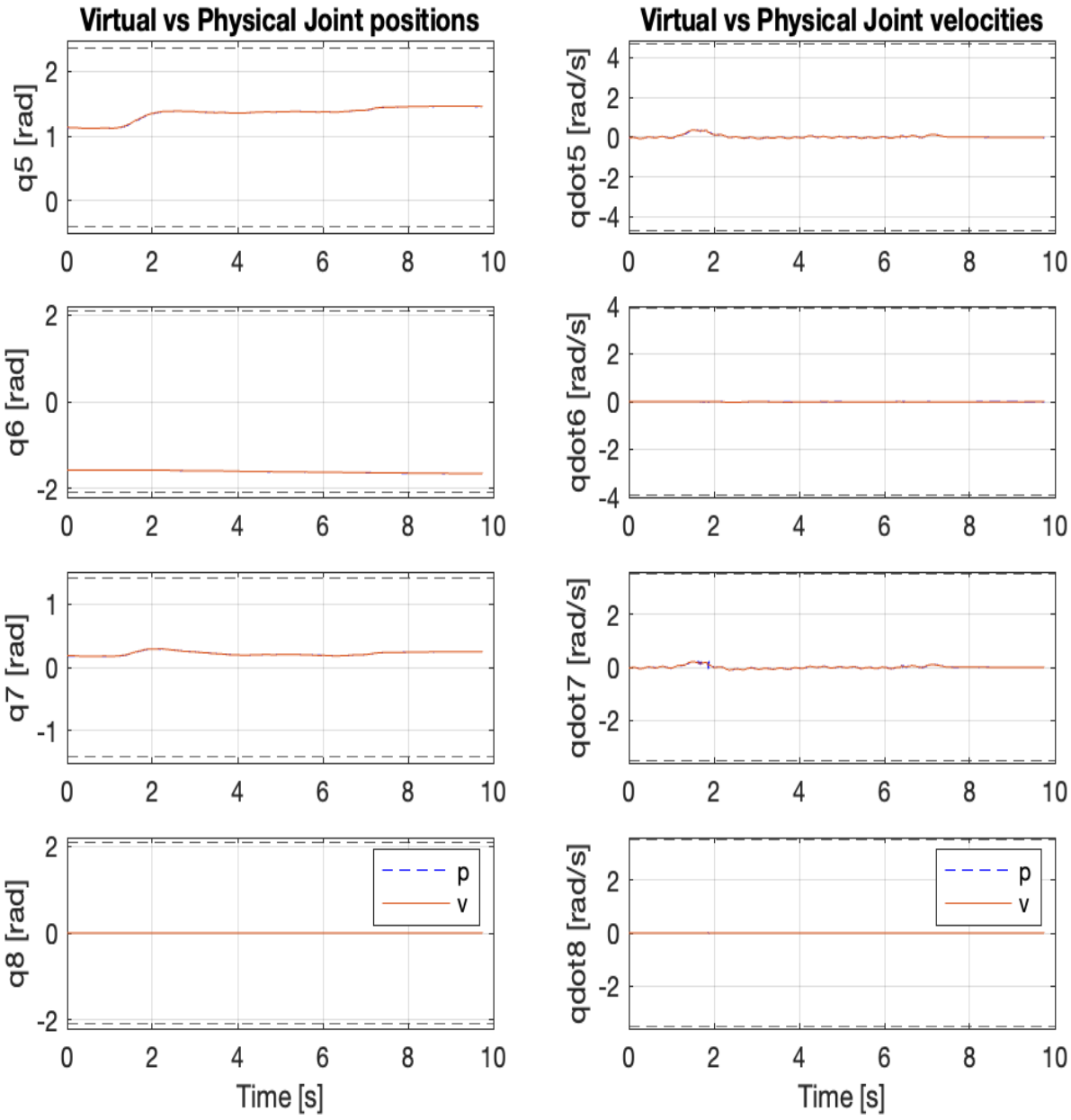


Figure 3.14: The task specification is given by Table. 3.2. The graphs in the right column show virtual (orange) and physical (dashed blue) joint positions for the last four joints; dashed lines denote virtual (and physical) upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec. 2.1.2). The corresponding graphs for the first four joints are seen in Fig. 3.13.

3.4.2 Pose Task

Here, we analyze the data for the execution of the taught grasp pose with the task hierarchy given by Table. 3.3. The robot executed a pose task (high priority)

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pose	DMP-MPC DMP	(1,80,300)	$\mathcal{F}_1 = \boldsymbol{I}_6$ $(m = 6)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	joint pos	$\mathbf{0}_{8\times1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table 3.3: Pose Task Specification. A DMP-MPC generates the position reference and an ordinary DMP for orientation; the frame indicates that it's a full pose task. There are no orientation constraints (limits). The secondary task is a joint task (indicated by the type) that considers all joints (indicated by the frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are used.

combined with a joint position task (low priority). The figures below show the robot's capability of tracking the end-effector pose and velocity references while maintaining all kinematic limits. Position and orientation tracking is good. A different hierarchy is validated in the following experiment.

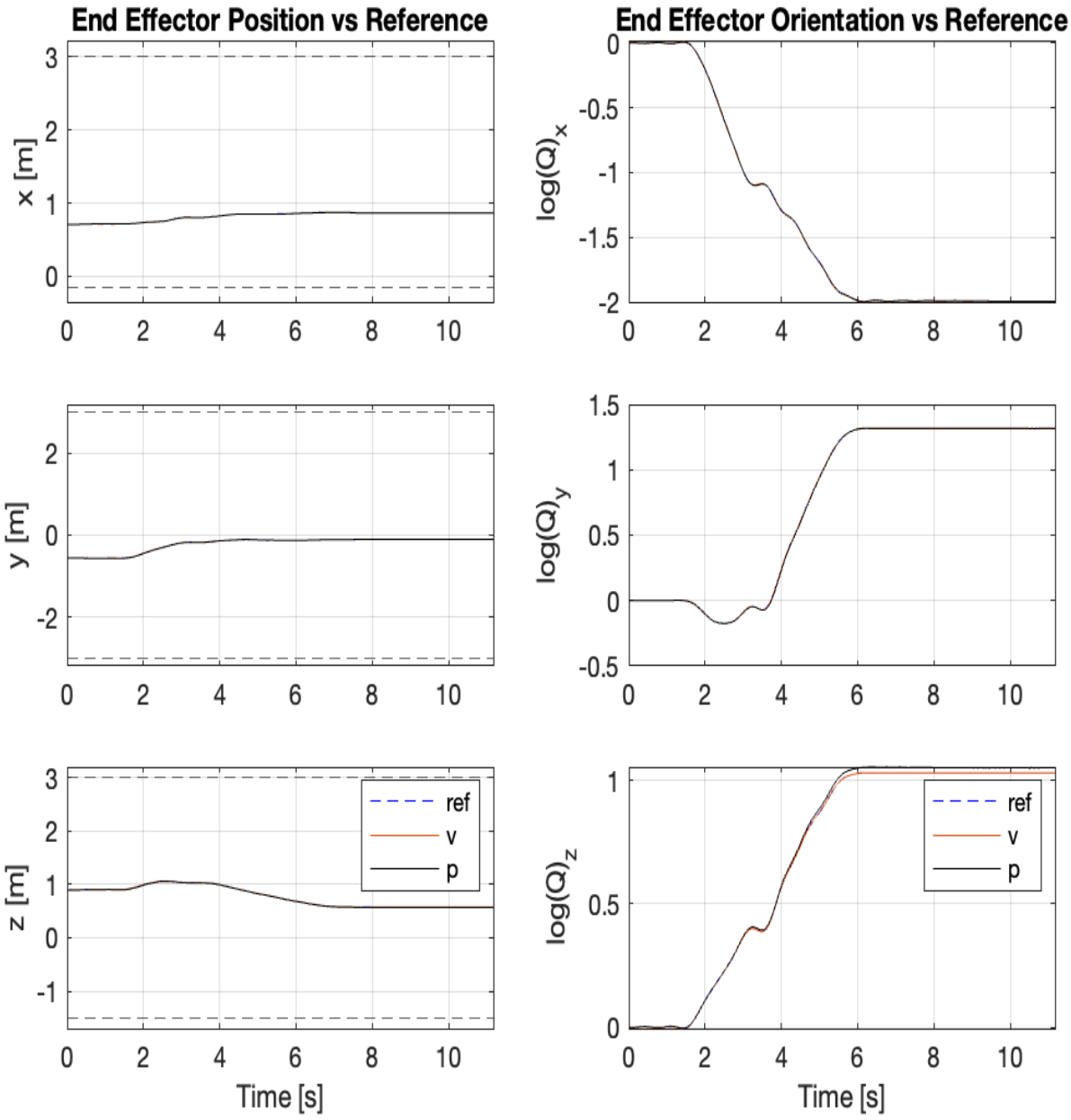


Figure 3.15: Task specification is given by Table. 3.3. The graphs in the left column depict the DMP-MPC-generated position reference (blue dashed line), virtual (orange), and physical (black) position; upper and lower dashed lines denote the corresponding position limits. The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec. 2.1.5) (blue dashed line), virtual (orange), and physical (black) orientation. Compare with Figs. 3.11, 3.19.

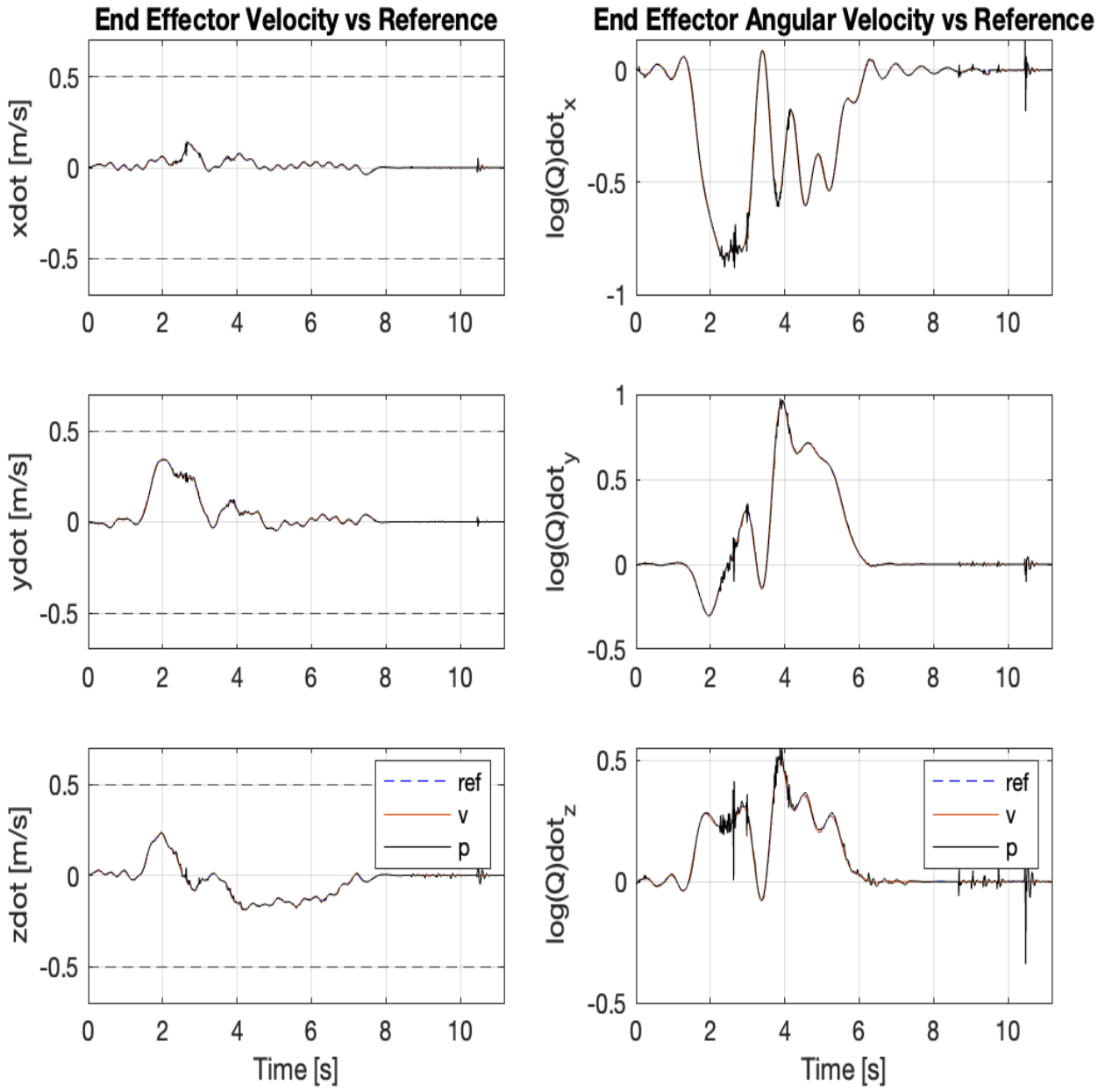


Figure 3.16: Task specification is given by Table. 3.3. The graphs in the left column depict the DMP-MPC-generated velocity reference (blue dashed line), virtual (orange), and physical (black) velocities; upper and lower dashed lines denote corresponding velocity limits. The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec. 2.1.5) (blue dashed line) and the corresponding virtual (orange) and physical (black) angular velocities. Virtual tracking of the angular velocity reference is, as expected, better than the previous case (Fig. 3.12).

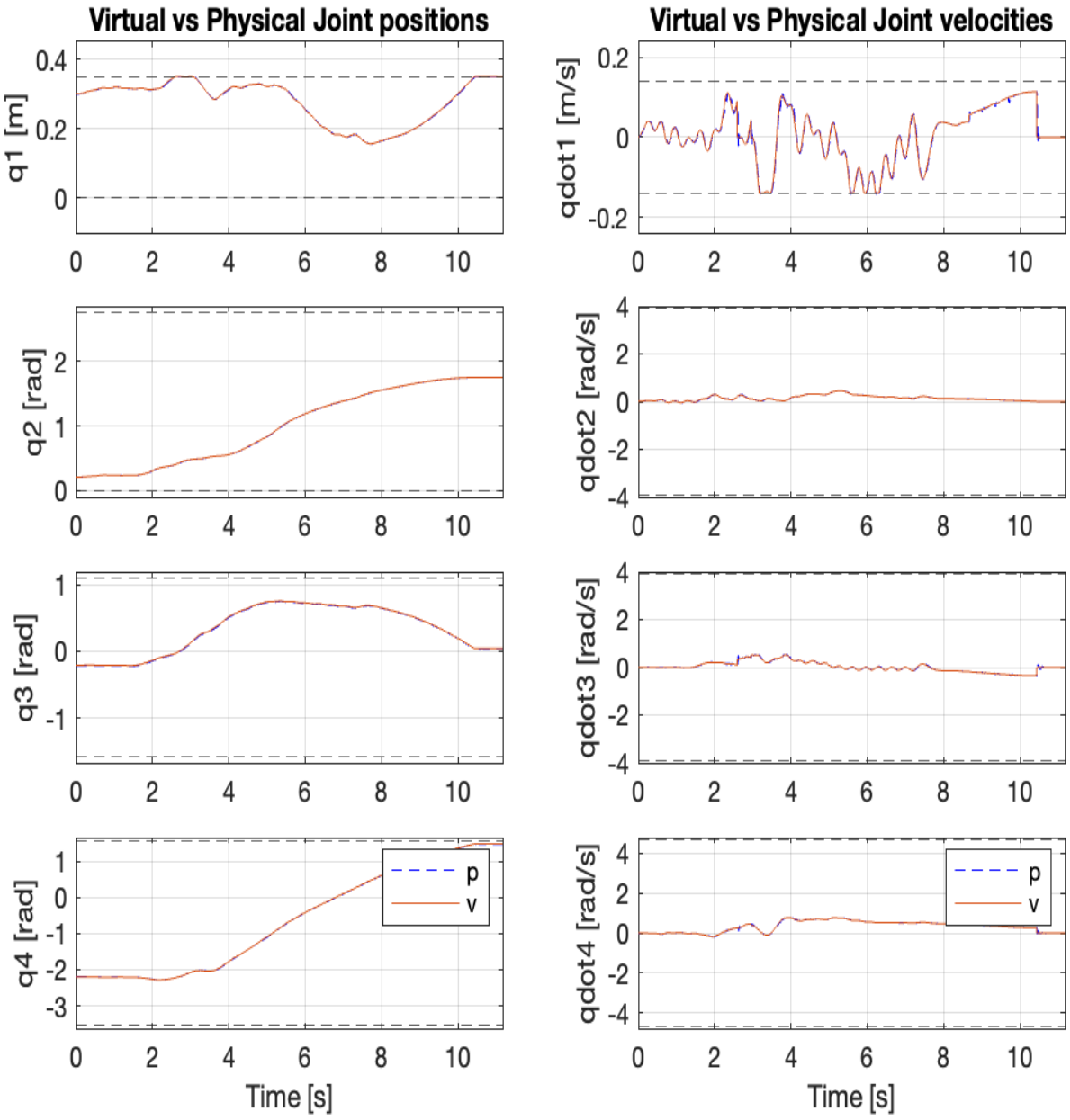


Figure 3.17: The task specification is given by Table. 3.3. The graphs in the right column show virtual (orange) and physical (dashed blue) joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec. 2.1.2). The corresponding graphs for the last four joints are seen in Fig. 3.18.

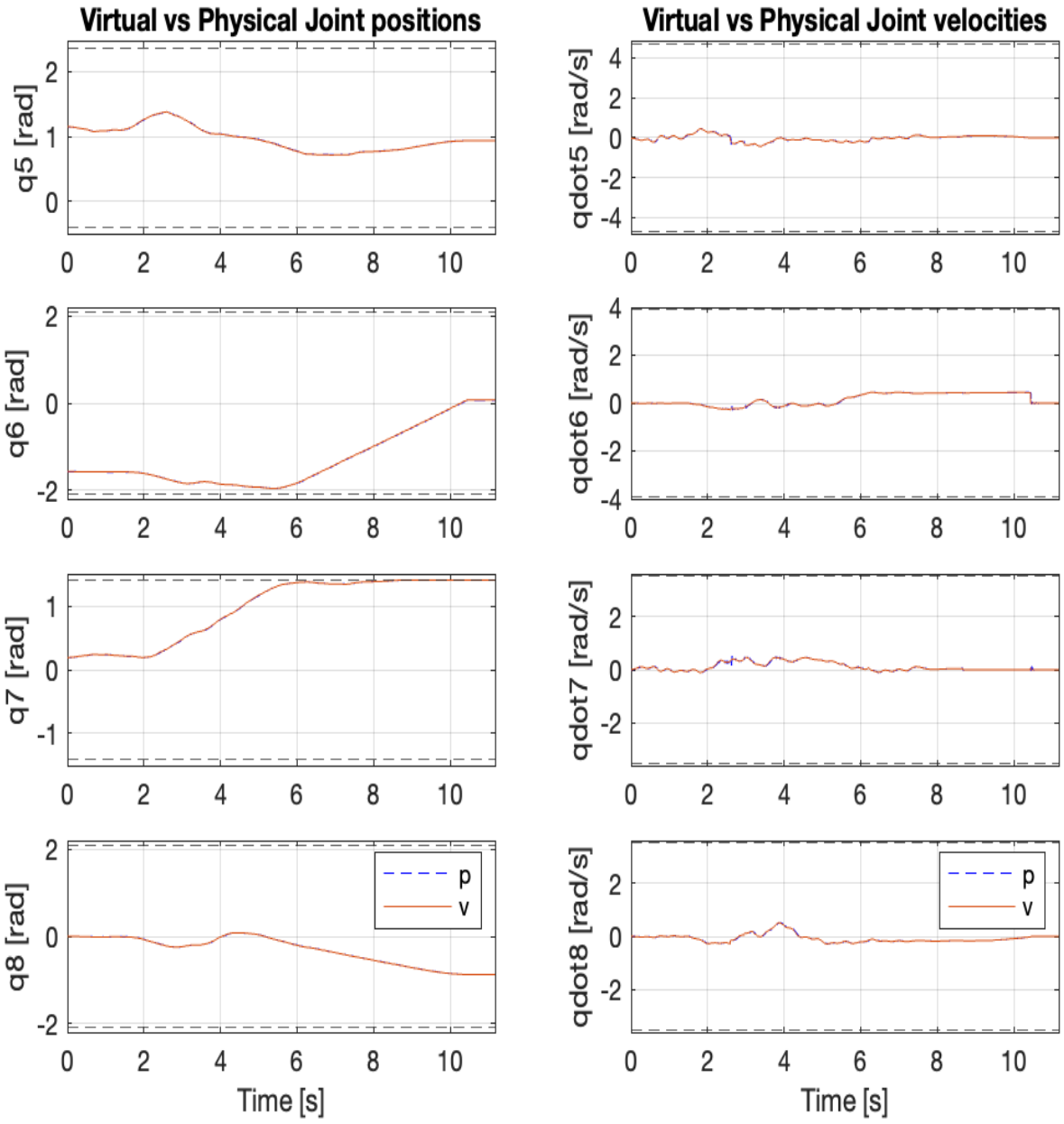


Figure 3.18: The task specification is given by Table. 3.3. The graphs in the right column show virtual (orange) and physical (dashed blue) joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec. 2.1.2). The corresponding graphs for the first four joints are seen in Fig. 3.17.

3.4.3 High Priority Position and Secondary Orientation Task

This section analyzes data for the execution of the taught grasp pose with the task hierarchy given by Table. 3.4.

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pos	DMP-MPC	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{I}_3 \\ \mathbf{0}_3\end{bmatrix}$ $(m=3)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	cart. or	DMP	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{0}_3 \\ \mathbf{I}_3\end{bmatrix}$ $(m=3)$	
3	joint pos	$\mathbf{0}$	(0,80,0)	$\mathcal{F}_3 = \boldsymbol{I}_8$ $(m=8)$	Eq. (2.54)

Table 3.4: Task Specification with a primary position task, secondary orientations task, and thirdly, a joint task. The task dimension is indicated by the corresponding frame. A DMP-MPC generates the position reference and an ordinary DMP generates the orientation reference. The third task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

A position task (high priority) combined with an orientation task (secondary priority) and a joint position task (low priority) were executed. Notice, that the orientation reference is not tracked as well as the position reference, and the same is observed for the positional and angular velocities. At around the 7s mark (Fig. 3.19), the position task has converged to its goal and we observe that the virtual orientation starts to approach the reference, albeit slowly. This slow convergence is unlikely due to any joint limits as only q_7 is at its positional limit. We know from Sec. 3.4.2, that orientation can be tracked much better. The only thing that has changed is the hierarchy, hence, this is a downside of using strict task hierarchies, at least as implemented in this work.

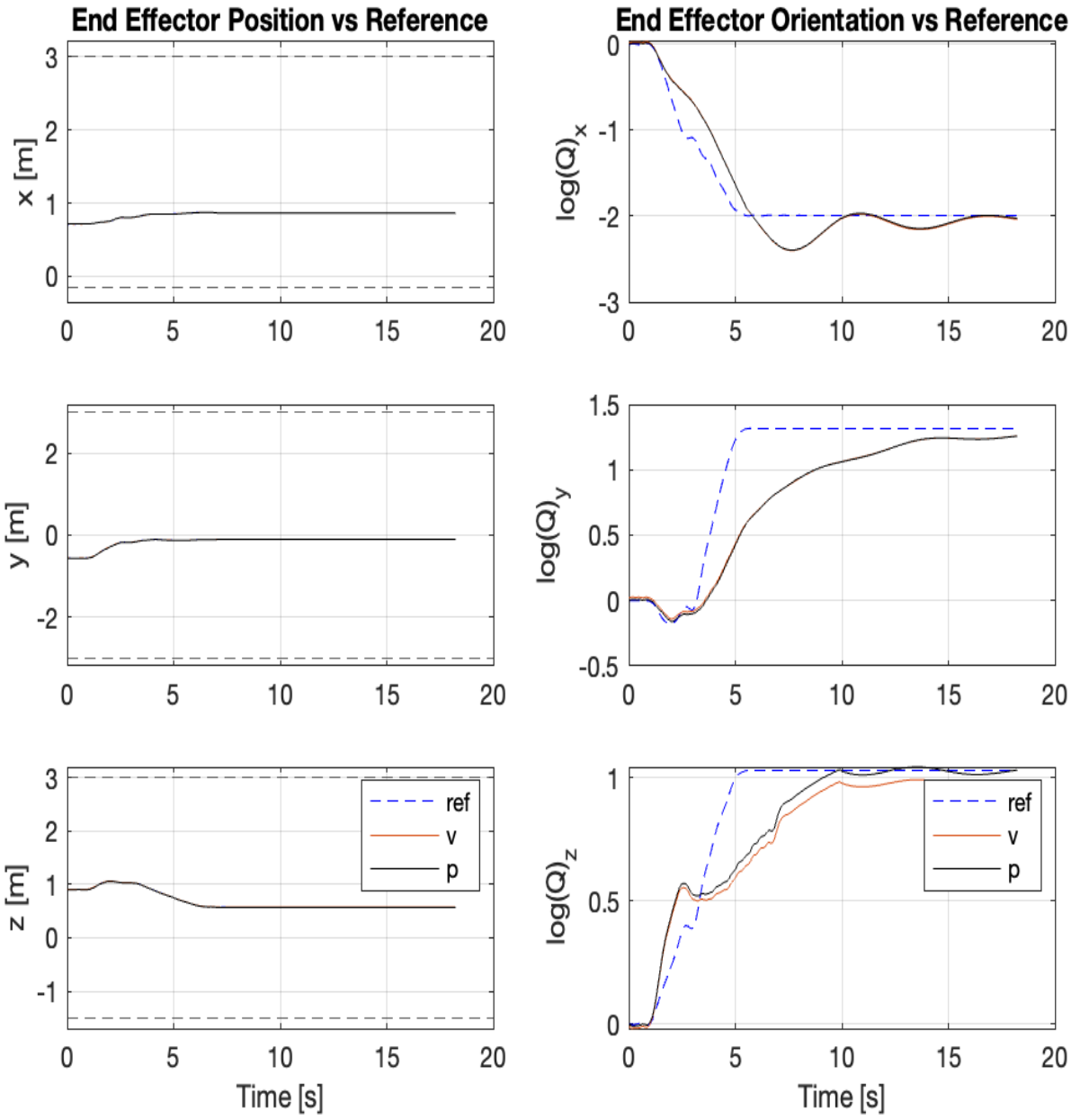


Figure 3.19: Task specification is given by Table. 3.4. The graphs in the left column depict the DMP-MPC-generated position reference (blue dashed line), virtual (orange), and physical (black) positions; upper and lower dashed lines denote the corresponding position limits. Graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec. 2.1.5) (blue dashed line) and the corresponding virtual (orange) and physical (black) orientation. Orientation reference tracking is worse than position tracking, compare this with the corresponding graphs in Figs. 3.11, 3.15.

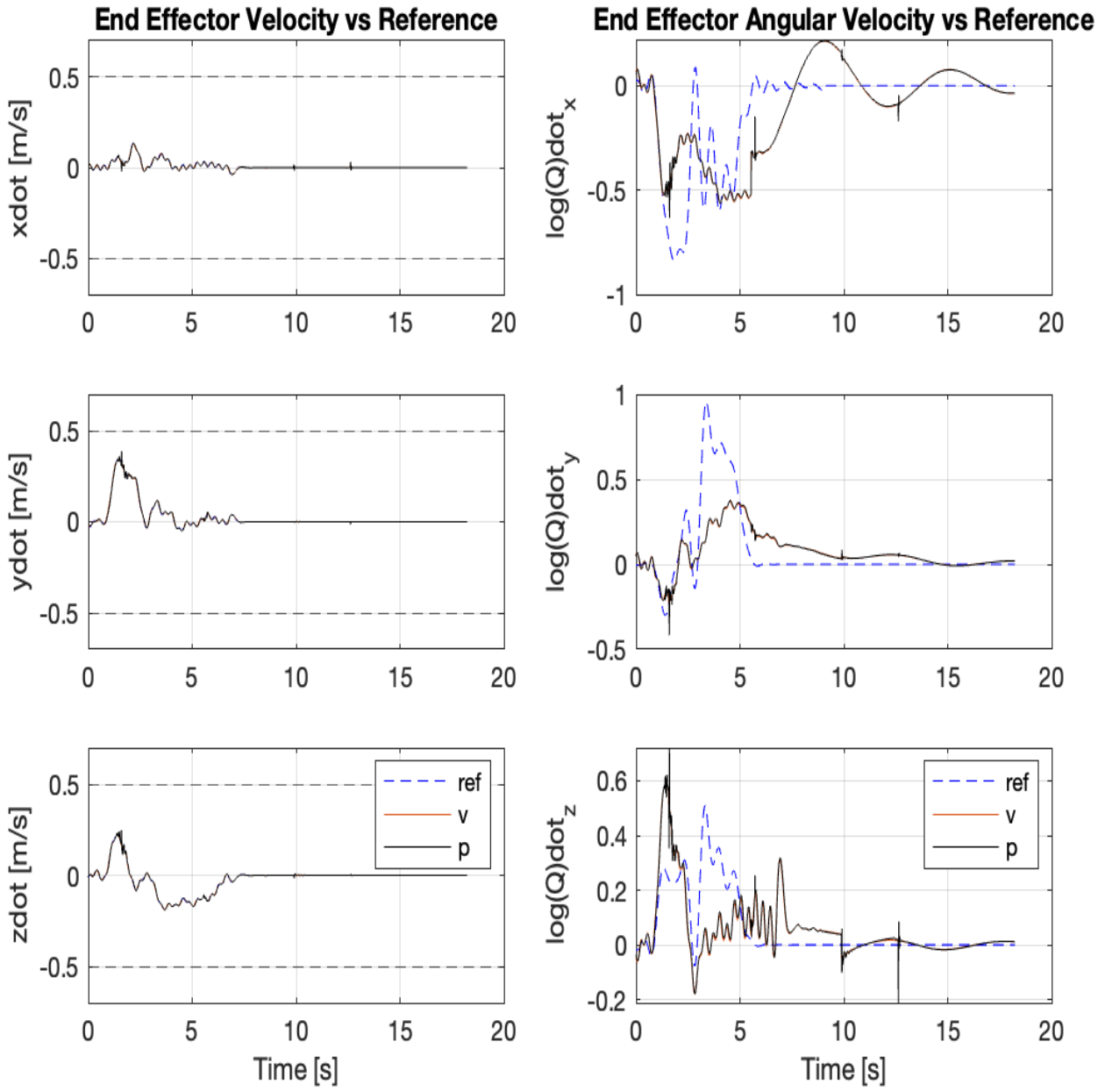


Figure 3.20: Task specification is given by Table. 3.4. The graphs in the left column depict the DMP-MPC-generated velocity reference (blue dashed line), virtual (orange), and physical (black) velocities; upper and lower dashed lines denote the corresponding velocity limits. Graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec. 2.1.5) (blue dashed line) and the corresponding virtual (orange) and physical (black) angular velocities. The angular velocity reference is somewhat tracked, compare this with the right graphs in Figs. 3.12, 3.16.

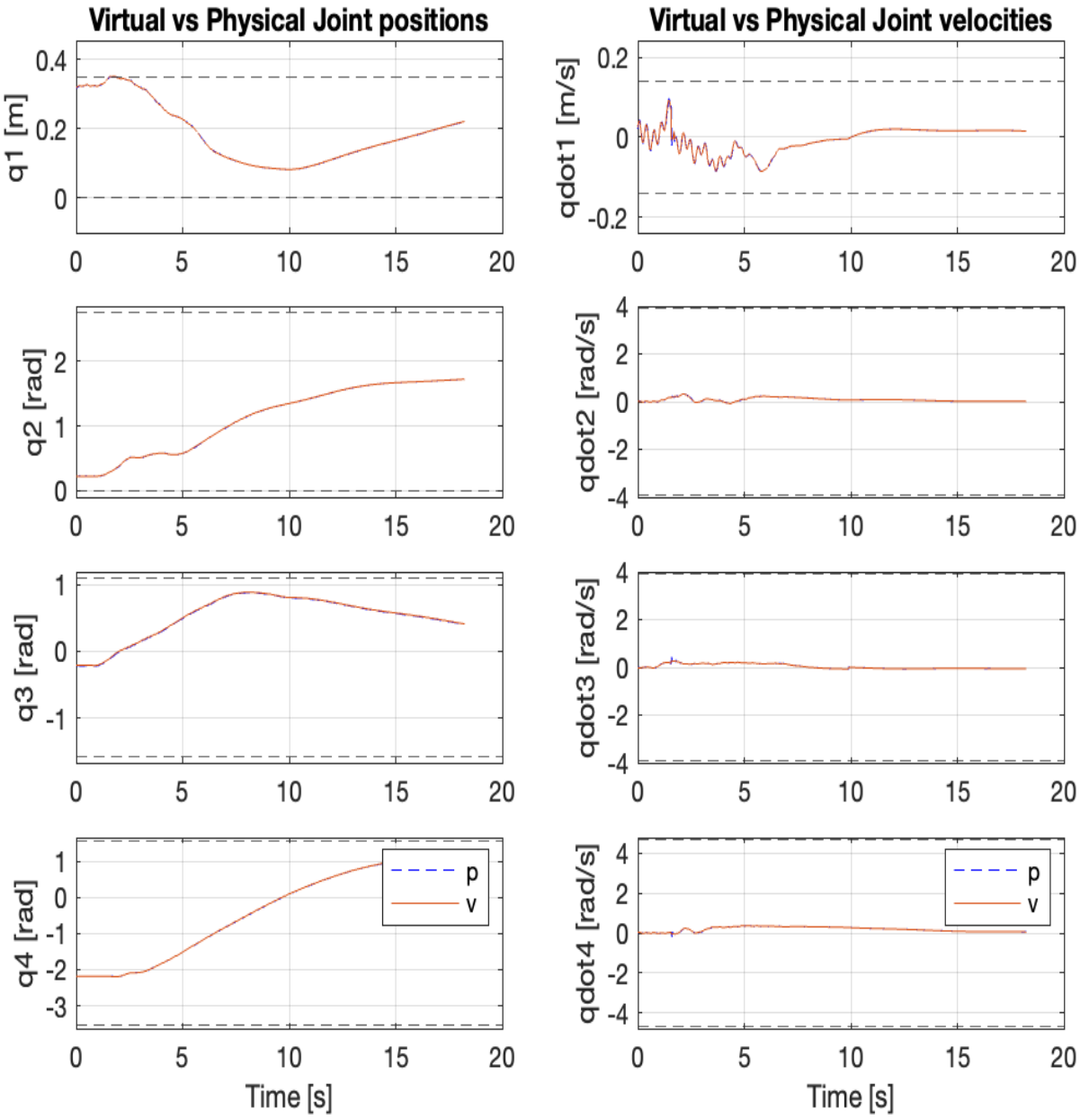


Figure 3.21: The task specification is given by Table. 3.4. The graphs in the right column show virtual (orange) and physical (dashed blue) joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec. 2.1.2). The corresponding graphs for the last four joints are in Fig. 3.22.

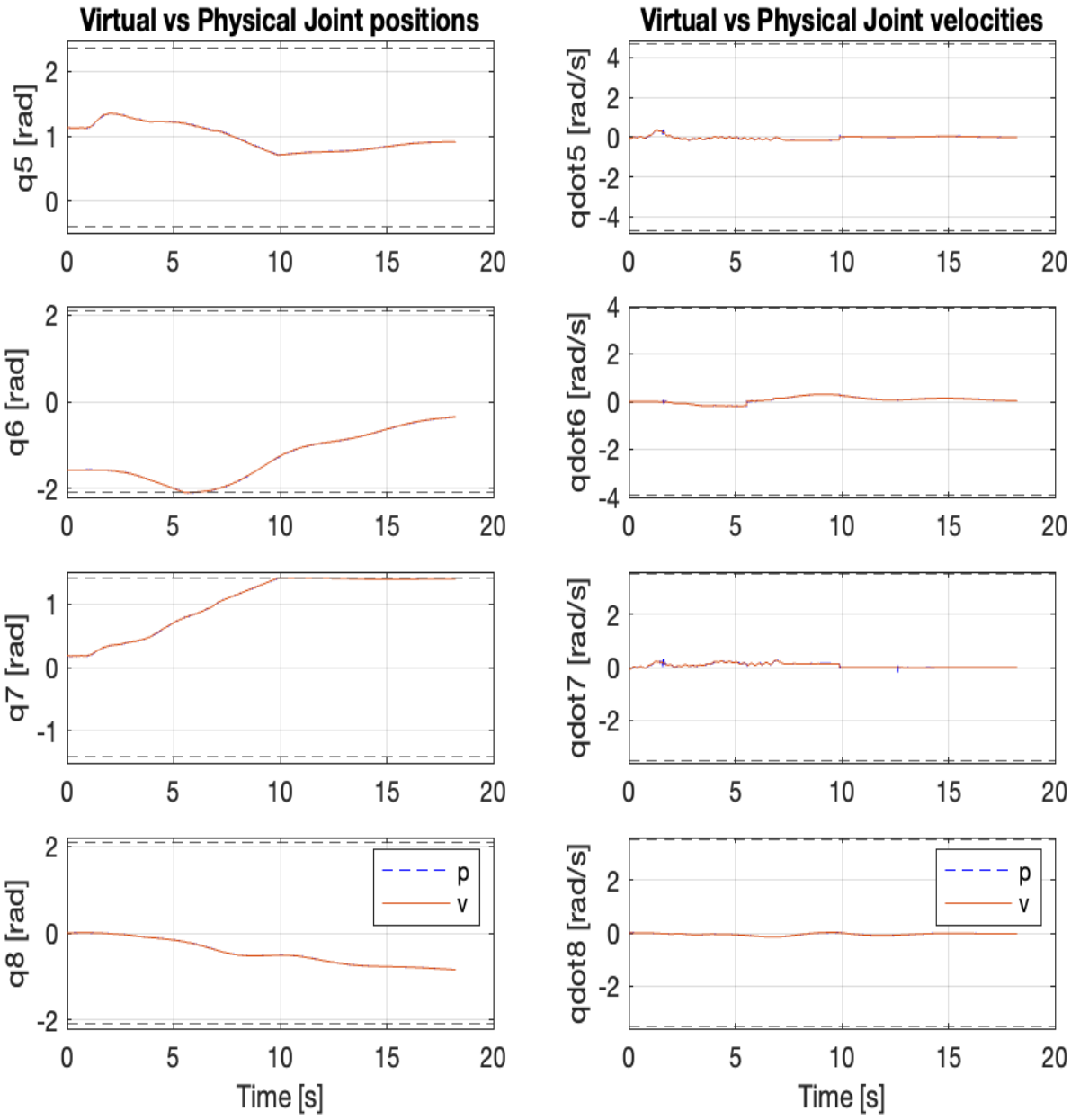


Figure 3.22: The task specification is given by Table. 3.4. The graphs in the right column show virtual (orange) and physical (dashed blue) joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec. 2.1.2). The corresponding graphs for the first four joints are seen in Fig. 3.21.

3.4.4 Summary of Results: Comparing Hierachies

In conclusion, our system does support hierarchical control, task tracking is highly affected by the task hierarchy. This is best demonstrated by comparing the execution for the full pose specification (Table. 3.3) and the specification with high-priority position and secondary orientation task (Table. 3.4). The robot is redundant w.r.t. both of these specifications (not considering the low-priority joint task), yet, the performance when executing the latter is considerably worse for the orientation even though we know, it's possible to track it without deteriorated positional tracking performance. In contrast to the tracking performance during KT (3.3), physical and virtual joint velocities were very similar. This is due to comparably slow virtual behavior, in this case, inertial effects are negligible.

3.5 Indirect Force Task

Up to this point, the focus has been on teaching and executing kinematic tasks. In this section, we explore an attempt to teach a dynamic task within a simulation environment. The motivation behind this exploration is the potential of utilizing the Dynamic Movement Primitives-Model Predictive Control (DMP-MPC) framework to learn force/torque references. This approach allows for the enforcement of maximum and minimum contact forces and jerks, ensuring the reference trajectory is smooth. However, the effectiveness of this method hinges on having a control strategy capable of reasonably following force references. Unfortunately, the strategy implemented here falls short of achieving this, and we will conclude with insights into the reasons for its inadequacy.

The task under consideration is a wiping motion, which involves maintaining contact between the end-effector and a surface (table) while executing a circular movement. Consequently, there is a force task acting along the surface normal, coinciding with the reference frame's z-axis. In the reference frame's x and y-axes, the end-effector follows a pre-defined circular velocity path. The orientation of the end-effector is maintained in its initial state, as depicted in Figure 3.23. The specifications of this task are detailed in Table 3.5. Note, that the highest priority task is an indirect force and mathematically, it's defined as

$$\mathbf{f} = K(\boldsymbol{p}_v - \boldsymbol{p}_p) \tag{3.4}$$

where $\mathbf{f} \in \mathbb{R}^3$, $K \in \mathbb{R}^+$, $\boldsymbol{p}_v \in \mathbb{R}^3$ and $\boldsymbol{p}_p \in \mathbb{R}^3$. The latter two, $\boldsymbol{p}_v$ and $\boldsymbol{p}_p$, are the Cartesian space positions of the virtual and physical robots respectively. Hence, the indirect force $\mathbf{f}$ is defined as a spring between the end-effector position of the virtual and physical robots. See Sec. 2.3 for the Jacobian and task compensation. The execution of this task can be seen in the video `https://youtu.be/t0Wgd-3bS7c`.

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	ind. Force	-1	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $(m = 1)$	
2	cart. pose	$\begin{bmatrix} \boldsymbol{p}_{circle}(t) \\ \boldsymbol{o}_{init} \end{bmatrix}$	$\begin{bmatrix} (0, 80, 0) \\ (1, 80, 300) \end{bmatrix}$	$\mathcal{F}_1 = \begin{bmatrix} \mathbf{I}_3 \\ \mathbf{I}_3 \end{bmatrix}$ $(m = 6)$	
3	joint pos	$\mathbf{0}$	(0,80,0)	$\mathcal{F}_3 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table 3.5: Task Specification with a primary indirect force task, secondary pose task, and thirdly, a joint task. The task dimension is indicated by the corresponding frame. The reference for the indirect force task is minus 1 Newtons. Note, in the z-direction (table normal), this means that the virtual robot should be positioned below the table when the desired indirect force is achieved. The reference for the pose task is given by user-defined functions. The third task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

In Figure 3.24, we observe effective tracking of the indirect force task but the measured force, varies greatly. Initially, the virtual end-effector is positioned below the physical one, leading to an initial negative indirect force. However, the physical robot catches up and surpasses the virtual, resulting in a positive indirect force value before contact. The end-effector velocity generates a notable impact force. Notably, the indirect force model does not consider actual forces, allowing the actual force to vary independently (within physical constraints) without affecting the indirect force control. For instance, starting with the robot in contact will cause the contact force to increase as the virtual robot moves below the table surface. An effective control strategy would result in indirect and measured forces to exhibit similar behaviors, albeit with potential differences in magnitude. Hence, the performance depends on the choice of spring constant and control parameters, which might have to reflect the stiffness of the robot and environment. We should also note the consequences of kinematic orientation control here. The robot is set to keep its initial orientation, and since it's purely kinematic, it can be seen as a stiff torsional spring (in 3D). As the robot moves around in a circle the controller will correct for orientational errors which will influence the contact force due to twisting and geometry. One could experiment with the control parameters or define an indirect torque task, which of course also involves tuning. An alternative method involves using admittance control, similar to the one used for Kinesthetic Teaching (KT). In KT, the reference contact force is set to zero to avoid contact, but it could be adjusted to a non-zero value, to seek contact. However, tuning these parame-

ters is likely difficult. Implementing admittance control at the acceleration level, as discussed in Section 4.1, might be advantageous. Teaching Dynamic Movement Primitives (DMPs) for either measured or indirect forces poses no inherent issues, allowing for the generation of reference trajectories with the DMP-MPC framework, to impose various constraints. Nonetheless, without a suitable control strategy, such testing is not fruitful. Additionally, kinesthetically teaching tasks like wiping using the wrist sensor is challenging. To teach wiping, for example, would require steering the end-effector with one hand and applying force behind the sensor with another. Moreover, the admittance algorithm employed for KT withdraws from contact and is therefore unsuitable for teaching tasks that require maintaining contact.

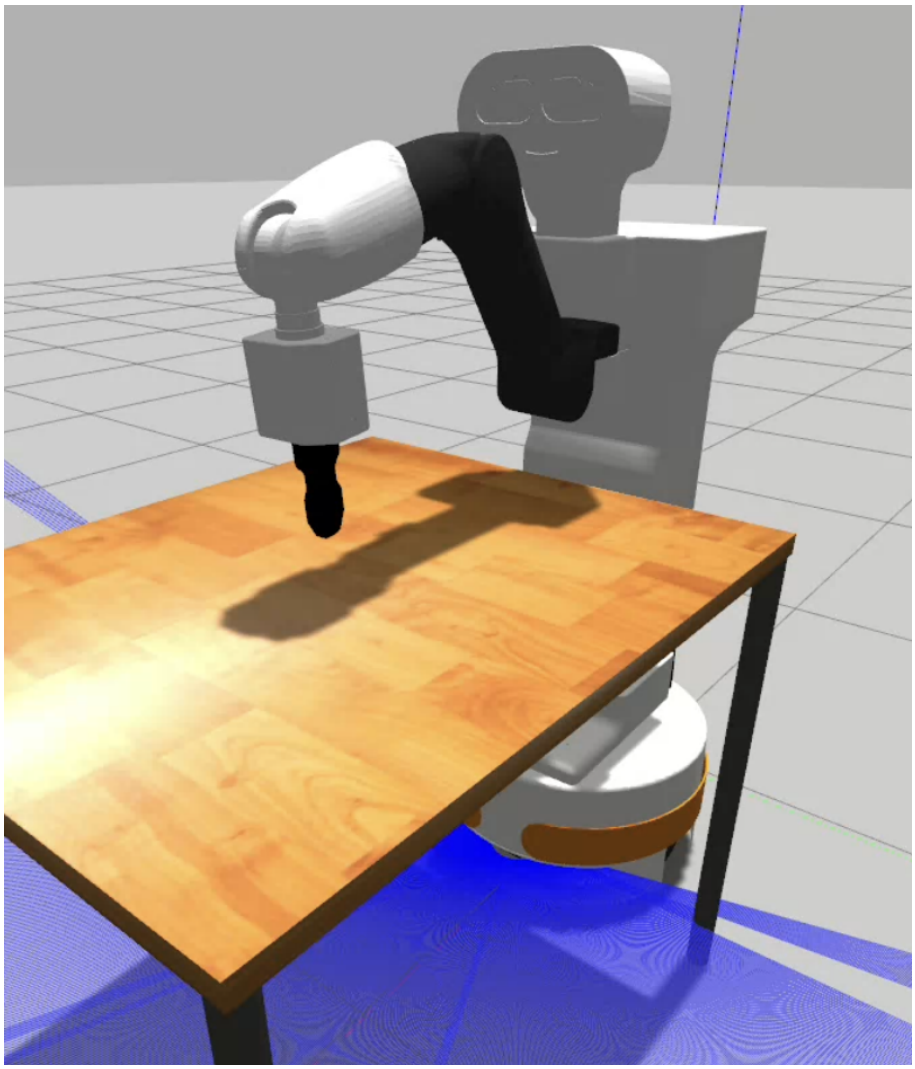


Figure 3.23: Setup in Gazebo for the indirect force task. The end-effector will seek contact in the direction of the table normal and follow a circular path along the table surface.

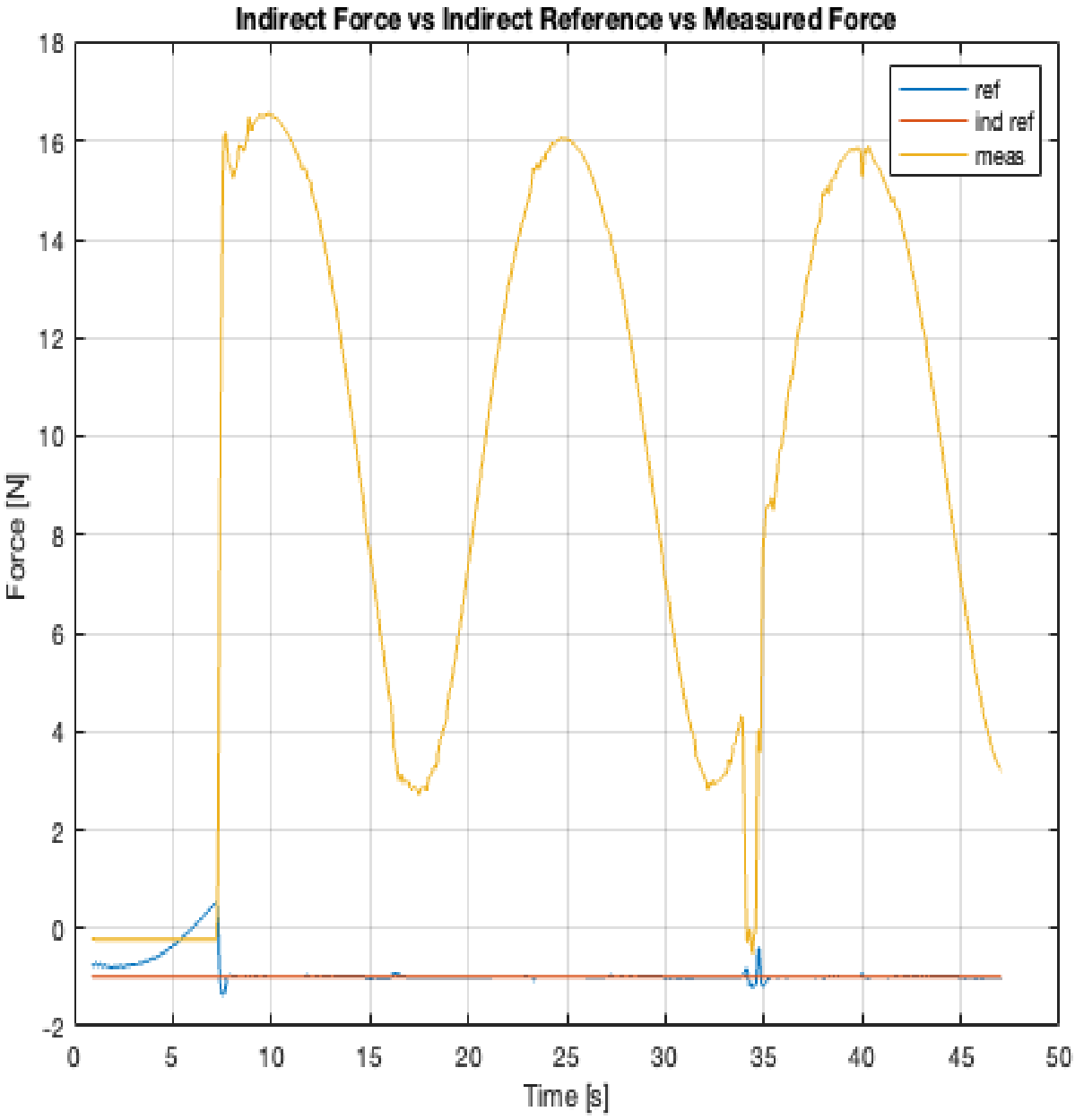


Figure 3.24: The graph shows the indirect force (blue), indirect force reference (red), and the measured force (yellow). We observe good tracking of the indirect force but the actual force varies a lot. Contact is made at around the 7s mark.

4
Discussion And Conclusion

In this chapter, we discuss findings, relate them to the research questions, and suggest directions for future work. For convenience, the research questions (RQs) are restated here. From the literature review, we identified the following open questions

- How can optimal prioritization of Dynamic Movement Primitives (DMPs) be learned from demonstrations?
- What method allows for the automatic selection of an optimal combination and prioritization of DMPs from a library of DMPs?

and this thesis aimed to investigate the performance achieved by letting a user choose DMPs and how to prioritize them, that is

- Can effective performance be achieved by allowing a user to define the hierarchy of DMPs?

4.1 Wrench Filtering and Kinesthetic Teaching

To address the RQ, "Can effective performance be achieved by allowing a user to define the hierarchy of DMPs?", it was necessary to facilitate the training of DMPs. To this end, we developed an interface for kinesthetic teaching (KT). This interface, discussed below, involved an F/T sensor, and algorithms for filtering, admittance, and control.

A similar performance between our filter and the PAL filter was observed in Sec. 3.3, and neither was close to the ideal filter. A future improvement to our filter would be to use a more accurate model of the end-effector and sensor noise. More accurate models would allow lower thresholds and therefore also mitigate the unintuitive behavior that may arise due to the use of thresholds (see below). Also, the PAL filter was only available for the physical robot, our filter was, therefore, necessary for testing in simulation. Hence, our filter can in the future be used to conduct KT with a Digital Twin of the robot in a Virtual Reality (VR) setting. Training in VR would simplify the teaching in simulation and offer safer and more efficient solutions in industrial contexts [68, 69, 70, 71].

The observed differences in virtual and physical robot velocities (and acceleration) in Sec. 3.3.2 are not ideal as the control model should be a good approximation of the physical system. One consequence of this mismatch is that the same velocity and acceleration limits, should not be used for the control model and physical robot. If the same limits are applied, the operator would have to accept relatively slow

motion (small applied wrenches) to have the robot move as intended, because the control model quickly reaches velocity limits to minimize the tracking error. However, it's more nuanced. The KT task is a pose velocity task, and pose velocity is related to the joint velocities through the geometric Jacobian which is configuration-dependent. In practice then, the robot will vary between seemingly compliant or stiff behavior, depending on configuration and input wrench (see video at, the corresponding data in Appendix). Furthermore, the robot arm can start to oscillate if the robot is in a stiff configuration and contact is dropped. A doubling of the velocity limits was tested which improved tracking performance and the experience. However, since thresholding is used (Alg. 4), unintuitive behavior can still occur when applying wrenches around the threshold as the robot can suddenly stop and start. This would suggest lowering the velocity control parameter D. Another option to deal with the model mismatch (and thresholding issues) is to change the reference generation. For example, the admittance algorithm (Alg. 4) could instead be applied at the acceleration level. As noted in Sec. 3.3.2, the admittance generated velocity reference contains high frequencies since it directly corresponds to the applied wrench and it necessitates fast tracking for the virtual robot. Instead, one could let the admittance generated acceleration be the reference task acceleration $\ddot{\boldsymbol{\xi}}_{i,ref}$ in Eq. (2.36) (repeated below, with no velocity reference and $K = 0$).

$$\dot{\boldsymbol{\xi}}_i^* = \dot{\boldsymbol{\xi}}_i + (M\ddot{\boldsymbol{\xi}}_{i,ref} - D\dot{\boldsymbol{\xi}}_i +)\Delta t$$

Then, one could tune M and D to approximate the effective inertia of the physical end-effector (corresponds to low pass filtering). This could yield smoother virtual behavior, improve the correspondence between virtual and physical motion, and mitigate sudden stops and starts around threshold wrenches. Finally, one could use a dynamic control model which is discussed below in Sec. 4.4.

For the experiments in Sec. 3.4, it was assumed that the velocity references were good approximations of user intent, and tracking performance would therefore reflect the intuitiveness of the KT interface. These experiments (adjustment of limits) justify the assumption since low limits led to poor reference tracking and unintuitive behavior, and vice versa.

4.2 Comparing Hierachies

To evaluate the 'performance' part of the RQ 'Can effective performance be achieved by allowing a user to define the hierarchy of DMPs?', we built a system that supports hierarchical control, where task priorities are user-defined and task references given by DMPs.

The results showed some areas for further development. Deteriorated tracking performance of lower priority tasks due to strict prioritization was noted. Using soft or time-varying priorities could solve the problem. However, properly defining soft or time-varying priorities is probably difficult and time-consuming. Introducing more degrees of freedom, manipulation, and obstacles, increase complexity even further.

The implication and answer to the RQ is then: Effective performance can be achieved for simple tasks but it's unlikely that user-defined hierarchies are near-optimal in the general case.

4.3 Optimization Challenges and Potential Solutions

In Sec. 2.4.2, we examined the optimization scheme used in our system. This optimization is local and may encounter issues such as local optima w.r.t. the task specification, or experiencing slow convergence for secondary tasks, as evidenced in Fig. 3.19. To circumvent these local optima, one strategy could involve manipulating the priority matrix (Eq. (2.45)), such as giving blocked tasks a soft priority relative to blocking tasks. Implementing this priority modulation presents a challenge: one possibility is to include the priorities as variables within the optimization framework, subject to relevant constraints. However, integrating this into the quadratic programming (QP) form (Eq. (2.48)) is complex due to the interaction between joint and priority values in the priority matrix (Eq. (2.46)). Identifying and utilizing unique mixed terms to define the optimization vector could be a potential solution. Anyhow, as mentioned in the previous Sec. 4.2, in a general situation, task prioritization is not trivial and robot programming would be difficult if the operator had to specify the time variation of the tasks.

Model Predictive Control (MPC) in the joint space could be explored to avoid local optima by minimizing task errors over a predefined N-step horizon of the joint trajectory. This method isn't straightforward since the model is

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{P}(\mathbf{q}_k)\dot{\mathbf{q}}_k \Delta t$$

, where the projection matrix $\mathbf{P}(\mathbf{q}_k)$ is state-dependent. A viable approach could be the linearization of $\mathbf{P}(\mathbf{q}_k)$ at the current joint state. While this might result in accurate predictions over only a short horizon (given the highly non-linear nature of $\mathbf{P}(\mathbf{q}_k)$), it could be sufficient for improved performance. Another option would be to use the approach in [72]. The controller uses the robot kinematic model to generate a convex optimization problem that can be solved efficiently. The physical robot is controlled via the virtual robot as in [20]. The state of the virtual robot is integrated from a velocity command to give position commands to the physical robot.

4.4 Challenges of a Kinematic Control Model

Using a dynamic control model would likely improve the tracking performance during both KT and task execution. Firstly, a dynamic control model would allow for using actual velocity and acceleration limits in the optimization without performance detriment. Secondly, faster movement than with a kinematic model could be

achieved since inertial effects are accounted for. However, a dynamic model would increase computational cost. Also, a dynamic model can directly respond to wrenches and therefore, there would be no need to translate wrenches to velocities as in Alg. 4.

A kinematic model is sufficient for positioning tasks (including in-contact tasks in structured environments) but it is not well suited for tasks involving contact with an unstructured environment (uncertain geometry, stiffness, movement of objects) since small errors can lead to large contact forces. When accounting for forces using a kinematic model, the admittance interface/model has to be tuned for different tasks which is non-trivial (Sec. 3.5). While this research focused on kinematic tasks, specifying tasks involving force and torque is feasible [20] (Sec. 3.5). The DMP-MPC framework could be utilized to learn force and torque trajectories, ensuring that the references (for force/torque and jerk) remain within acceptable limits. It would be interesting to explore these tasks in a VR setting and evaluate their real-world applicability. Effective manipulation or in-contact tasks, as indicated by current research, benefit significantly from the implementation of dynamic models in control systems [73].

4.5 Conclusion

- Can effective performance be achieved by allowing a user to define the hierarchy of DMPs?

Yes, for simple tasks, effective performance can be achieved by allowing a user to define the hierarchy of DMPs but, the user-defined hierarchy is unlikely to be optimal in the sense of minimizing task errors over the task execution.

In this thesis, we have explored the potential of user-defined DMP hierarchies in robotic control, addressing challenges in prediction, control models, and optimization. We built a general interface and our results demonstrate the feasibility of this approach and suggest directions for enhancing robot performance and user interactions.

4.6 Ethics and Sustainability

The integration of robots into the workforce can alleviate humans from repetitive, hazardous, and ergonomically unsound tasks, potentially enhancing productivity and economic growth. This evolution could lead to a scenario where displaced workers are reabsorbed into new sectors of an expanding economy. Nevertheless, there is a risk of enduring unemployment for some.

A critical concern is the potential misuse of robotic technology. The dual nature of technology means that advancements capable of yielding beneficial outcomes can also facilitate harm. While some technologies, like nuclear weapons or chemical agents, allow for relatively straightforward monitoring and regulation, the programming of robots presents a more complex challenge. International frameworks and

legal standards may play a crucial role in establishing ethical guidelines, yet they fall short of guaranteeing universal adherence to ethical practices. The hope remains, however, that the benefits will outweigh the drawbacks.

Collaborative robots (cobots) and Digital Twins (DTs) could improve efficiency in terms of energy and material use, contributing to reduced production costs and enhancing the feasibility of smaller, local manufacturing facilities. Moreover, the demographic shift towards an aging population necessitates a more efficient healthcare system [74]. Cobots could be a valuable asset in supporting healthcare professionals.

Bibliography

[1] I. F. of Robotics. (2023) World robotics 2023 report: Asia ahead of europe and the americas. [Online]. Available: https://ifr.org/ifr-press-releases/news/world-robotics-2023-report-asia-ahead-of-europe-and-the-americas#:~:text=The%20Americas%20In%20the%20Americas%2C,to%2039%2C576%20units

[2] A. Sosa-Ceron, H. Gonzalez-Hernandez, and J. Reyes-Avendaño, "Learning from demonstrations in human–robot collaborative scenarios: A survey." *Robotics*, vol. 11, no. 6, 2022. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85144697491&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[3] S. Kakade, B. Patle, and A. Umbarkar, "Applications of collaborative robots in agile manufacturing: a review," *Robotic Systems and Applications*, vol. 3, no. 1, pp. 59–83, jun 2023. [Online]. Available: https://doi.org/10.21595/rsa.2023.23238

[4] A. Zanchettin, E. Croft, H. Ding, and M. Li, "Collaborative robots in the workplace [from the guest editors]." *IEEE Robotics Automation Magazine, Robotics Automation Magazine, IEEE, IEEE Robot. Automat. Mag*, vol. 25, no. 2, pp. 16 – 17, 2018. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8385413&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[5] C. Schou, J. Damgaard, S. Bøgh, and O. Madsen, "Human-robot interface for instructing industrial tasks using kinesthetic teaching," in *IEEE ISR 2013*, 2013, pp. 1–6.

[6] A. Weiss, A.-K. Wortmeier, and B. Kubicek, "Cobots in industry 4.0: A roadmap for future practice studies on human-robot collaboration." *IEEE Transactions on Human-Machine Systems*, vol. 51, no. 4, pp. 335–345 – 345, 2021. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85111070979&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[7] A. Jacopo, S. Matteo, and M. Riccardo, "Editorial: Learning, perception, and collaboration for robots in industrial environments." *Frontiers in Robotics and AI*, vol. 9, 2022. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edsdoj&AN=edsdoj.964c82e289784fd6b09acd9ca2ccf675&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[8] B. Yalçinkaya, M. Couceiro, S. Soares, and A. Valente, "Human-aware collaborative robots in the wild: Coping with uncertainty in activ-

ity recognition." *Sensors*, vol. 23, no. 7, 2023. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85152335555&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[9] Q. Li, Y. Pang, W. Cai, Y. Wang, Q. Li, and M. Zhao, "An overview of multi-task control for redundant robot based on quadratic programming," in *Proceedings of 2023 Chinese Intelligent Automation Conference*, Z. Deng, Ed. Singapore: Springer Nature Singapore, 2023, pp. 641–666.

[10] M. Fiore, G. Meli, A. Ziese, B. Siciliano, and C. Natale, "A general framework for hierarchical redundancy resolution under arbitrary constraints." *IEEE Transactions on Robotics, Robotics, IEEE Transactions on, IEEE Trans. Robot*, vol. 39, no. 3, pp. 2468 – 2487, 2023. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.10008952&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[11] M. Liu, Y. Tan, and V. Padois, "Generalized hierarchical control." *Autonomous Robots*, vol. 40, no. 1, p. 17, 2016. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edsgao&AN=edsgcl.439196494&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[12] A. Dietrich, C. Ott, and A. Albu-Schaeffer, "An overview of null space projections for redundant, torque-controlled robots." *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, vol. 34, no. 11, pp. 1385 – 1400, 2015. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edswsc&AN=000361771400003&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[13] A. BAUER, D. WOLLHERR, and M. BUSS, "Human–robot collaboration: A survey," *International Journal of Humanoid Robotics*, vol. 05, no. 01, pp. 47–66, 2008. [Online]. Available: https://doi.org/10.1142/S0219843608001303

[14] S. Calinon, *Learning from Demonstration (Programming by Demonstration)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1007/978-3-642-41610-1_27-1

[15] J. L. Nevins and D. E. Whitney, *The Force Vector Assembler Concept*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1972, pp. 273–288. [Online]. Available: https://doi.org/10.1007/978-3-662-40393-8_19

[16] M. T. Mason, "Compliance and force control for computer controlled manipulators." *IEEE Transactions on Systems, Man, and Cybernetics, Systems, Man and Cybernetics, IEEE Transactions on, IEEE Trans. Syst., Man, Cybern*, vol. 11, no. 6, pp. 418 – 432, 1981. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.4308708&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[17] H. Bruyninckx and J. De Schutter, "Specification of force-controlled actions in the "task frame formalism"-a synthesis." *IEEE Transactions on Robotics and Automation, Robotics and Automation, IEEE Transactions on, IEEE Trans. Robot. Automat*, vol. 12, no. 4, pp. 581 – 589, 1996.

[Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.508440&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[18] J. De Schutter and H. Van Brussel, "Compliant robot motion i. a formalism for specifying complaint motion tasks." *International Journal of Robotics Research*, vol. 7, no. 4, p. 3, 1988. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=4705007&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[19] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation." *IEEE Journal on Robotics and Automation, Robotics and Automation, IEEE Journal of, IEEE J. Robot. Automat*, vol. 3, no. 1, pp. 43 – 53, 1987. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.1087068&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[20] E. Lutscher, E. Dean-Leon, and G. Cheng, "Hierarchical force and positioning task specification for indirect force controlled robots." *IEEE Transactions on Robotics, Robotics, IEEE Transactions on, IEEE Trans. Robot*, vol. 34, no. 1, pp. 280 – 286, 2018. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8167335&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[21] D. Whitney, "Resolved motion rate control of manipulators and human prostheses." *IEEE Transactions on Man-Machine Systems, Man-Machine Systems, IEEE Transactions on, IEEE Trans. Man-Machine Syst*, vol. 10, no. 2, pp. 47 – 53, 1969. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.4081862&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[22] "Automatic supervisory control of the configuration and behavior of multibody mechanisms." *IEEE Transactions on Systems, Man, and Cybernetics, Systems, Man and Cybernetics, IEEE Transactions on, IEEE Trans. Syst., Man, Cybern*, vol. 7, no. 12, pp. 868 – 871, 1977. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.4309644&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[23] A. Maciejewski and C. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments." *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117 – 117, 1985. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-0022130025&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[24] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems." *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments, Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, p. 1211, 1991. [Online]. Available: https://search.ebscohost.

com/login.aspx?direct=true&db=edseee&AN=edseee.240390&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[25] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics. [electronic resource] : Modelling, Planning and Control.*, ser. Advanced Textbooks in Control and Signal Processing. Springer London, 2009. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=cat07472a&AN=clec.SPRINGERLINK9781846286421&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[26] R. Boulic, D. Thalmann, and R. Mas, "A robust approach for the control of the center of mass with inverse kinetics." *Computers and Graphics (Pergamon)*, vol. 20, no. 5 SPEC. ISS., pp. 693–701 – 701, 1996. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-0030233522&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[27] G. Antonelli, S. Chiaverini, and G. Fusco, "A new on-line algorithm for inverse kinematics of robot manipulators ensuring path tracking capability under joint limits." *IEEE Transactions on Robotics and Automation, Robotics and Automation, IEEE Transactions on, IEEE Trans. Robot. Automat*, vol. 19, no. 1, pp. 162 – 167, 2003. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.1177175&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[28] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives." *I. J. Humanoid Robotics*, vol. 2, pp. 505–518, 12 2005.

[29] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[30] F. Flacco, A. De Luca, and O. Khatib, "Prioritized multi-task motion control of redundant robots under hard joint constraints." *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 3970 – 3977, 2012. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6385619&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[31] B. Nelson and P. Khosla, "Strategies for increasing the tracking region of an eye-in-hand system by singularity and joint limit avoidance." *The International Journal of Robotics Research*, vol. 14, no. 3, pp. 255–269 – 269, 1995. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-0029322904&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[32] P. Baerlocher and R. Boulic, "An inverse kinematics architecture enforcing an arbitrary number of strict priority levels." *Visual Computer*, vol. 20, no. 6, pp. 402–417 – 417, 2004. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-4444342694&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[33] D. Raunhardt and R. Boulic, "Progressive clamping." *Proceedings 2007 IEEE International Conference on Robotics and Automation, Robotics and Automation, 2007 IEEE International Conference on*, pp. 4414 – 4419, 2007. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.4209777&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[34] . . Dehio, N. (1 and J. . . . Steil, "Dynamically-consistent generalized hierarchical control." in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, no. 2019 International Conference on Robotics and Automation, ICRA 2019, (1)Karlsruhe Institute for Technology, 2019, pp. 1141–1147 – 1147. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85071430333&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[35] O. Kanoun, F. Lamiraux, P.-B. Wieber, F. Kanehiro, E. Yoshida, and J.-P. Laumond, "Prioritizing linear equality and inequality systems: Application to local motion planning for redundant robots." *2009 IEEE International Conference on Robotics and Automation, Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 2939 – 2944, 2009. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.5152293&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[36] O. Kanoun, F. Lamiraux, and P.-B. Wieber, "Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task." *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 785 – 792, 2011. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=63987707&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[37] J. Quiroz-Omana and B. Adorno, "Whole-body control with (self) collision avoidance using vector field inequalities." *IEEE Robotics and Automation Letters, Robotics and Automation Letters, IEEE, IEEE Robot. Autom. Lett*, vol. 4, no. 4, pp. 4048 – 4053, 2019. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8763977&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[38] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation." *International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028 – 1028, 2014. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-84901601589&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[39] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions." *2011 IEEE International Conference on Robotics and Automation, Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1283 – 1290, 2011. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.5980202&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[40] L. Mingxing, A. Micaelli, P. Evrard, A. Escande, and C. Andriot, "Interactive virtual humans: A two-level prioritized control framework with wrench bounds." *IEEE Transactions on Robotics, Robotics, IEEE Transactions on, IEEE Trans. Robot*, vol. 28, no. 6, pp. 1309 – 1322, 2012. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6264106&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[41] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey." 2021. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.2102.03861&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[42] A. Ijspeert, J. Nakanishi, and S. Schaal, "Trajectory formation for imitation with nonlinear dynamical systems." *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180), Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, Intelligent robots and systems*, vol. 2, p. 752, 2001. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.976259&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[43] ——, "Movement imitation with nonlinear dynamical systems in humanoid robots." *Proceedings 2002 IEEE International Conference on Robotics Automation (Cat. No.02CH37292)*, pp. 1398 – 1403, 2002. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=81989352&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[44] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[45] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal, "Online movement adaptation based on previous sensor experiences." *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 365 – 371. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6095059&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[46] A. Ude, B. Nemec, T. Petric, and J. Morimoto, "Orientation in cartesian space dynamic movement primitives." *2014 IEEE International Conference on Robotics and Automation (ICRA), Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2997 – 3004, 2014. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6907291&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[47] L. Koutras and Z. Doulgeri, "A correct formulation for the orientation dynamic movement primitives for robot control in the cartesian space,"

in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 293–302. [Online]. Available: https://proceedings.mlr.press/v100/koutras20a.html

[48] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal, "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 2587–2592.

[49] M. Ginesi, N. Sansonetto, and P. Fiorini, "Overcoming some drawbacks of dynamic movement primitives." *Robotics and Autonomous Systems*, vol. 144, 2021. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0921889021001299&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[50] L. Koutras and Z. Doulgeri, "A novel dmp formulation for global and frame independent spatial scaling in the task space." *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), Robot and Human Interactive Communication (RO-MAN), 2020 29th IEEE International Conference on*, pp. 727 – 732, 2020. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.9223500&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[51] A. D. Dragan, K. Muelling, J. Andrew Bagnell, and S. S. Srinivasa, "Movement primitives via optimization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2339–2346.

[52] A. Sidiropoulos and Z. Doulgeri, "A reversible dynamic movement primitive formulation." 2020. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.2010.07708&site=eds-slive&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[53] A. Sidiropoulos, D. Papageorgiou, and Z. Doulgeri, "A novel framework for generalizing dynamic movement primitives under kinematic constraints." *Autonomous Robots*, vol. 47, no. 1, pp. 37–50 – 50, 2023. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85139677383&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[54] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2013/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf

[55] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell, "Kernelized movement primitives." 2017. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1708.08638&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[56] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, IEEE Trans.*

Syst., Man, Cybern. B, vol. 37, no. 2, pp. 286 – 298, 2007. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.4126276&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[57] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann, "Probabilistic prioritization of movement primitives." *IEEE Robotics and Automation Letters, Robotics and Automation Letters, IEEE, IEEE Robot. Autom. Lett*, vol. 2, no. 4, pp. 2294 – 2301, 2017. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.7973083&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[58] V. Dambly, "Trajectory generation using dynamic movement primitives under hierarchical constraints." 2019. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=ir01625a&AN=cst.20.500.12380.300313&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[59] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320314000235

[60] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[61] P. Robotics, *Tiago User Manual*, PAL Robotics, 2023, available from PAL Robotics. [Online]. Available: http://www.pal-robotics.com/en/products/tiago/

[62] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. Wiley, 2020. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=cat09075a&AN=clpc.oai.edge.chalmers.folio.ebsco.com.fs00001000.e8be8baa.a497.42fa.821e.46eb58381bf7&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[63] F. Keith, P. Wieber, N. Mansard, and A. Kheddar, "Analysis of the discontinuities in prioritized tasks-space control under discreet task scheduling operations." *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 3887 – 3892, 2011. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6094706&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[64] T. Petrič and L. Žlajpah, "Smooth continuous transition between tasks on a kinematic control level: Obstacle avoidance as a control problem." *Robotics and Autonomous Systems*, vol. 61, no. 9, pp. 948 – 959, 2013. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0921889013000833&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[65] A. Dietrich, A. Albu-Schaffer, and G. Hirzinger, "On continuous null space projections for torque-based, hierarchical, multi-objective manipulation." *2012 IEEE International Conference on Robotics and Automation, Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 2978 – 2985, 2012. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6224571&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[66] F. Flacco, A. De Luca, and O. Khatib, "Motion control of redundant robots under joint constraints: Saturation in the null space." *2012 IEEE International Conference on Robotics and Automation, Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 285 – 292, 2012. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6225376&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[67] PAL Robotics, *TIAGo Handbook*, PAL Robotics, Barcelona, 2020.

[68] A. Ramasubramanian, R. Mathew, M. Kelly, V. Hargaden, and N. Papakostas, "Digital twin for human-robot collaboration in manufacturing: Review and outlook." *Applied Sciences (Switzerland)*, vol. 12, no. 10, 2022. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85132575780&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[69] A. Rivera-Pinto, J. Kildal, and E. Lazkano, "Toward programming a collaborative robot by interacting with its digital twin in a mixed reality environment." *INTERNATIONAL JOURNAL OF HUMAN-COMPUTER INTERACTION*, 2023. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edswsc&AN=001005917800001&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[70] N. Jose E., S. Diego G., R.-L. Angel, R.-L. Paola, A.-O. Andrea, and G. Marcelo V., "A scoping review on virtual reality-based industrial training." *Applied Sciences*, vol. 10, no. 8224, p. 8224, 2020. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edsdoj&AN=edsdoj.fe53561b2c04d0f929c98ccaa9897f4&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds

[71] I. Jang, H. Niu, E. C. Collins, A. Weightman, J. Carrasco, and B. Lennox, "Virtual kinesthetic teaching for bimanual telemanipulation," in *2021 IEEE/SICE International Symposium on System Integration (SII)*, 2021, pp. 120–125.

[72] L. Wen, H. Jiang, Y. Li, J. Wang, and Y. Liu, "Fast Joint Space Model Predictive Control for Reactive Manipulation," 2021.

[73] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems*, vol. 156, p. 104224, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889022001312

[74] D. Rouzet, A. C. Sánchez, T. Renault, and O. Roehn, "Fiscal challenges and inclusive growth in ageing societies," no. 27, 2019. [Online]. Available: https://www.oecd-ilibrary.org/content/paper/c553d8d2-en

A

Appendix A

```
#ifndef ROBOT_STATE_H
#define ROBOT_STATE_H

#include <Eigen/Dense>
#include <Eigen/Geometry>
#include <gmp_lib/math/quaternions.h>
#include <gmp_lib/math/conversions.h>
//include <gmp_lib/GMP/GMP_regressor.h>
#include <gmp_lib/math/math.h>
#include <unordered_map>
#include <mutex>

// Forward declaration of GMP_regressor
using namespace as64_::gmp_;

namespace tiago_qp_ghc_controller {

struct ArucoGoal {
        int marker_id;
        arma::mat R_goal_offset_wrt_torso_fixed_link;
        double radius_offset;
        double marker_size;
        arma::vec position;
        arma::vec Q;
};

struct ArucoObstacle {
    std::string name;

    int marker_id;

    double marker_size;

    bool seen;

    arma::vec radius;
    arma::vec position;
    arma::vec Q;
    arma::mat Sigma;
};

struct RobotState /*AndParams*/ {

    std::mutex pose_mutex_;
```

```
45      std::mutex vision_mutex_;
46      std::mutex ft_sensor_mutex_;
47      std::mutex joint_state_mutex_;
48      std::mutex phase_mutex_;
49      std::mutex gmp_mutex_;
50
51      Eigen::VectorXd virtualJointState_; // v stands for virtual
52      Eigen::VectorXd physicalJointState_; // p stands for physical
53      Eigen::VectorXd virtualJointVelocity_;
54      Eigen::VectorXd physicalJointVelocity_;
55      Eigen::VectorXd virtualJointAcceleration_;
56      Eigen::VectorXd physicalJointAcceleration_;
57      Eigen::VectorXd lowerPosLim_;
58      Eigen::VectorXd upperPosLim_;
59      Eigen::VectorXd velLim_;
60      Eigen::MatrixXd physicalJacEeWrtTorso_;
61      Eigen::MatrixXd virtualJacEeWrtTorso_;
62      Eigen::Matrix3d physicalRotMatEeWrtTorso_;
63      Eigen::Matrix3d virtualRotMatEeWrtTorso_;
64      Eigen::Vector3d forceOffset_;
65      Eigen::Vector3d torqueOffset_;
66
67      arma::vec3 virtualEePosition_;
68      arma::vec3 physicalEePosition_;
69      arma::vec4 virtualEeOrientation_; // Unit quaternion (w,x,y,z)
70      arma::vec4 initialOrientation_;
71      arma::vec4 physicalEeOrientation_;
72      arma::vec3 virtualEeEta_; // quatLog space for the orientation
73      arma::vec3 physicalEeEta_;
74      arma::vec3 virtualEeVelocity_;
75      arma::vec3 physicalEeVelocity_;
76      arma::vec3 virtualEeAngularVelocity_;
77      arma::vec3 physicalEeAngularVelocity_;
78      arma::vec3 virtualEeEtaVelocity_; // quatLog space for the
    orientation
79      arma::vec3 physicalEeEtaVelocity_;
80      arma::vec3 indirectForceState_; // Defined from the difference
    in physical and virtual states
81      arma::vec3 indirectForceStateDot_;
82      arma::vec3 indirectForceStateDDot_;
83      arma::vec3 indirectTorqueState_;
84      arma::vec3 indirectTorqueStateDot_;
85      arma::vec3 indirectTorqueStateDDot_;
86      arma::vec3 explicitForceState_; // Defined from sensor
    measurements
87      arma::vec3 explicitForceStateDot_;
88      arma::vec3 explicitForceStateDDot_;
89      arma::vec3 explicitTorqueState_;
90      arma::vec3 explicitTorqueStateDot_;
91      arma::vec3 explicitTorqueStateDDot_;
92      arma::vec4 Qg_original_;
93      arma::vec4 Q0_original_;
94      arma::mat goalPositionsFromVision_; // each column is a goal
    position.
95      arma::mat goalOrientationsFromVision_; // each column is a goal
     orientation. (quaternions)
```

```
96      arma::mat ObstaclesFromVision_; // each column is an obstacle
        position.
97      arma::mat33 virtualRotMat_;
98      arma::mat33 physicalRotMat_;
99      arma::mat33 L_;
100     arma::vec3 policyNormal_;
101     arma::vec3 angularPolicyNormal_;
102     arma::vec3 eePositionRef_;
103     arma::vec3 eePositionRefDot_;
104     arma::vec3 eePositionRefDDot_;
105     arma::vec3 eeOrientationRef_;
106     arma::vec3 eeAngularVelRef_;
107     arma::vec3 eeAngularVelRefDot_;
108     arma::vec jointRef_;
109     arma::vec jointRefDot_;
110
111     double phase_;
112     double phaseDot_;
113     double phaseDDot_;
114     double duration_;
115     double durationDot_;
116     double durationDDot_;
117     double KForce_;
118     double KTorque_;
119     double forceThreshold_;
120     double torqueThreshold_;
121     double t_;
122     double dt_;
123     double g_;
124     double m_;
125     double forceDiffGain_;
126     double torqueDiffGain_;
127     double linearVelLimit_;
128     double angularVelLimit_;
129     double jointVelRegularization_;
130     double positionDeviationThreshold_;
131     double orientationDeviationThreshold_;
132
133     int jointNumber_;
134     int nPhaseSteps_;
135
136     bool verbose_;
137
138     struct {
139
140         std::unordered_map<int, ArucoGoal> goal_map;
141         std::unordered_map<int, ArucoObstacle> seen_obstacles;
142
143     }vision;
144
145 };
146
147 }
148
```

```
149 #endif
```

Listing A.1: RobotState Class Definition

```
1  #ifndef TASK_H
2  #define TASK_H
3
4  #include <Eigen/Dense>
5  #include <Eigen/Geometry>
6  #include <memory>
7  #include <gmp_lib/math/quaternions.h>
8  #include <gmp_lib/math/conversions.h>
9  #include <gmp_lib/math/math.h>
10 #include <gmp_lib/GMP/GMP_regressor.h>
11 #include <gmp_lib/CanonicalSystem/CanonicalSystem.h>
12 #include <gmp_lib/GMP/GMP_MPC.h>
13 #include <tiago_qp_ghc_controller/TiagoKinematics.hpp>
14 #include <tiago_qp_ghc_controller/RobotState.hpp>
15 #include <ros/ros.h>
16
17 using namespace as64_::gmp_;
18
19 namespace tiago_qp_ghc_controller {
20
21     class Task {
22
23         public:
24             Task();
25             ~Task();
26
27             struct {
28                 int N_horizon;
29                 int N_kernels;
30
31                 double pred_time_step;
32                 double kernels_std_scaling;
33                 double kernels_trunc_thres;
34                 double opt_pos_gain;
35                 double opt_vel_gain;
36                 double time_limit;
37                 double abs_tol;
38                 double rel_tol;
39
40                 arma::vec pos_lim_low;
41                 arma::vec pos_lim_up;
42                 arma::vec vel_lim_low;
43                 arma::vec vel_lim_up;
44                 arma::vec accel_lim_low;
45                 arma::vec accel_lim_up;
46                 arma::vec slack_limits;
47                 arma::vec slack_gains;
48                 arma::vec final_state_err_tol; // = arma::vec {5e
    -3, 1e-2, 5e-2};
49                 arma::vec torso_ellipsoid_center; // [0.0, 0.0,
    0.5]
50                 arma::vec torso_ellipsoid_radius; // [0.15, 0.15,
```

```
    0.625] # Visualize in rviz
51                  arma::vec base_ellipsoid_center; // [0.125, 0.0,
    -0.05]
52                  arma::vec base_ellipsoid_radius; // [0.25, 0.25,
    0.125] #
53
54              } gmp_mpc_config;
55
56              std::string type_;
57              std::string subspaceFrame_;
58              std::string referenceType_;
59              std::string gmpType_;
60              std::string cartGmpFilePath_;
61              std::string quatGmpFilePath_;
62              std::string forceGmpFilePath_;
63              std::string torqueGmpFilePath_;
64              std::string jointGmpFilePath_;
65
66              Eigen::MatrixXd subspace_;
67              Eigen::Vector3d markerPosOffset_;
68              Eigen::Vector4d markerQuatOffset_;
69
70              double springConstant_;
71              double dampingConstant_;
72              double kernelTruncation_;
73
74              int goal_marker_id;
75
76              std::vector<double> stdScaling_; // this is at most
    size 2!
77              std::vector<int> NKernels_;
78
79              //CanonicalSystem *can_sys;
80              //std::unique_ptr<CanonicalSystem> can_sys;
81              //std::shared_ptr<CanonicalSystem> can_sys;
82
83              GMP::Ptr cartGmp_;
84              GMP_MPC::Ptr gmp_mpc_;
85              GMPo::Ptr quatGmp_;
86              GMP::Ptr forceGmp_;
87              GMP::Ptr torqueGmp_;
88              GMP::Ptr jointGmp_;
89              GMP::Ptr modelDeviationGmp_; // Were only going to use
    the regressor for this one
90              /*
91                  Hence, it needs kernels and scaling and truncation.
     It's init with the rest of
92                  the task. Need to add the modelDeviation kernels,
    scaling and truncation to
93                  the config file. These could be added to the
    stdScaling and Nkerne vectors .We still need our own weights
    though.
94                  Weights are init to zero but with the correct size!
     (ndof x Nkernels). ndof will be 6.
95
96                  For retraining, we need to add the modelDeviation
```

```
    weights to the regressor.
97
98              */
99
100             void (Task::*stateFcn)(const RobotState &robotState,
    arma::vec &taskState, arma::vec &taskStateDot);
101             void (Task::*referenceFcn)(RobotState &robotState, arma
    ::vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
102             void (Task::*taskJacobian)(const RobotState &robotState
    , Eigen::MatrixXd &J);
103             void (Task::*taskCompensation)(const RobotState &
    robotState, Eigen::VectorXd &comp);
104             void (Task::*frameFcn)(const RobotState &robotState,
    Eigen::MatrixXd &S);
105
106             // reference functions for the tasks
107             void poseGmpRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot); // poseGmpRef is the reference function for
    the pose task
108             void poseGmpVisionRef(RobotState &robotState, arma::vec
     &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot); // poseGmpRef is the reference function for
    the pose task
109             void cartGmpRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
110             void quatGmpRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
111             void cartGmpVisionRef(RobotState &robotState, arma::vec
     &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
112             void quatGmpVisionRef(RobotState &robotState, arma::vec
     &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
113             void cartGmpMpcRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
114             void quatGmpMpcRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
115             void cartGmpIterativeRef(RobotState &robotState, arma::
    vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
116             void quatGmpIterativeRef(RobotState &robotState, arma::
    vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
117             void forceGmpRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
118             void torqueGmpRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
119             void wrenchGmpRef(RobotState &robotState, arma::vec &
```

```
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
120             void jointGmpRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
121             void jointZeroVelRef(RobotState &robotState, arma::vec
    &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
122             void wrenchToTransVelRef(RobotState &robotState, arma::
    vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
123             void wrenchToAngularVelRef(RobotState &robotState, arma
    ::vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
124             void wrenchToPoseVelRef(RobotState &robotState, arma::
    vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
125             void markerToPosRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
126             void markerToEtaRef(RobotState &robotState, arma::vec &
    taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
127             void markerToPoseRef(RobotState &robotState, arma::vec
    &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
128             void constPoseGmpRef(RobotState &robotState, arma::vec
    &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
129             void poseGmpInspectionRef(RobotState &robotState, arma
    ::vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
130             void iterativePoseGmpRef(RobotState &robotState, arma::
    vec &taskRefState, arma::vec &taskRefStateDot, arma::vec &
    taskRefStateDDot);
131
132             // state functions for the tasks
133             void poseState(const RobotState &robotState, arma::vec
    &taskState, arma::vec &taskStateDot);
134             void ktPoseState(const RobotState &robotState, arma::
    vec &taskState, arma::vec &taskStateDot);
135             void indirectCartWrenchState(const RobotState &
    robotState, arma::vec &taskState, arma::vec &taskStateDot);
136             void explicitCartWrenchState(const RobotState &
    robotState, arma::vec &taskState, arma::vec &taskStateDot);
137             void jointState(const RobotState &robotState, arma::vec
     &taskState, arma::vec &taskStateDot);
138
139             // compensation functions for the tasks
140             void poseCompensation(const RobotState &robotState,
    Eigen::VectorXd &comp);
141             void wrenchCompensation(const RobotState &robotState,
    Eigen::VectorXd &comp); // Need the state of the physical and
    virtual robot
142             void jointCompensation(const RobotState &robotState,
    Eigen::VectorXd &comp);
```

```
143
144             // Jacobian functions for the tasks
145             void poseJacobian(const RobotState &robotState, Eigen::
    MatrixXd &J);
146             void wrenchJacobian(const RobotState &robotState, Eigen
    ::MatrixXd &J);
147             void jointJacobian(const RobotState &robotState, Eigen
    ::MatrixXd &J);
148
149             //Subspace functions
150             void subspaceSelection(const RobotState &robotState,
    Eigen::MatrixXd &S);
151             void subspaceEeOrientation(const RobotState &robotState
    , Eigen::MatrixXd &S);
152
153
154     };
155
156 }
157 #endif
```

Listing A.2: Task Class Definition

```
1  #ifndef DURATION_SYSTEM_FCNS_H
2  #define DURATION_SYSTEM_FCNS_H
3
4  #include <tiago_qp_ghc_controller/RobotState.hpp>
5
6  namespace tiago_qp_ghc_controller {
7
8
9          struct Duration {
10                 double duration;
11                 double alpha;
12                 double beta;
13                 double gamma;
14                 double delta;
15                 double epsilon;
16                 double eta;
17                 void (*durationFcn)(RobotState &robotState, const
    Duration &params);
18         };
19
20         void durationSys1(RobotState &robotState, const Duration &
    params); // duration state is in the robotState
21         void durationSys2(RobotState &robotState, const Duration &
    params);
22         void durationSys3(RobotState &robotState, const Duration &
    params);
23
24 }
25
26
27 #endif
```

Listing A.3: Duration Class Definition

```
#ifndef CANONICAL_SYSTEM_FCNS_H
#define CANONICAL_SYSTEM_FCNS_H

#include <tiago_qp_ghc_controller/RobotState.hpp>

namespace tiago_qp_ghc_controller {

    struct Canonical_System {
        double alpha;
        double beta;
        double gamma;
        double delta;
        double epsilon;
        double eta;
        void (*canSysFcn)(RobotState &robotState, const
    Canonical_System &params);
     };

     void canSys1(RobotState &robotState, const Canonical_System &
    params);// Linear forward
     void canSys2(RobotState &robotState, const Canonical_System &
    params); // constant phase to give the goal
     void canSys3(RobotState &robotState, const Canonical_System &
    params); // constant phase. In the middle for non goal directed
    behavior. (teaching)
     void canSys4(RobotState &robotState, const Canonical_System &
    params); // Linear reverse
     void canSys5(RobotState &robotState, const Canonical_System &
    params); // Bidirectional driving along policy using FT sensor
     void canSys6(RobotState &robotState, const Canonical_System &
    params); // Velocity teaching using the FT sensor
     void canSys7(RobotState &robotState, const Canonical_System &
    params); // Second order


}

#endif
```

Listing A.4: Canonical Class Definition

B

Appendix B

B.1 Trace Rectangle

In this section, we compare the task tracking for the same task (taught using KT) with different task specifications (hierarchies). The task is to move into a grasping pose, see Fig. B.1. The teaching and execution of this task can be seen in the video `https://youtu.be/t0Wgd-3bS7c`.

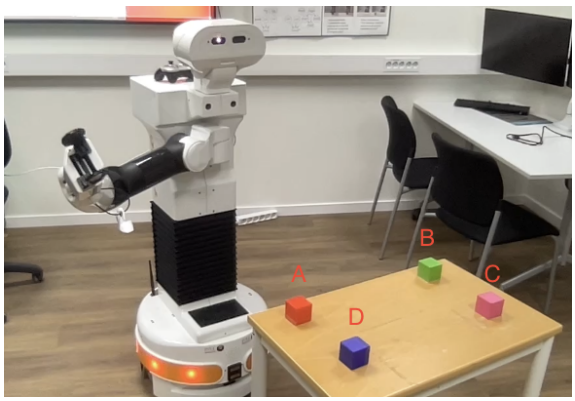


Figure B.1: Depiction of the rectangle task. The task was to teach the robot to trace the rectangle represented by the cubes 'A', 'B', 'C', and 'D'.

Eqs. (B.4), (B.5) and (B.6) give the lower, and upper Cartesian space position and velocity limits used for the position tasks below.

$$\underline{\mathbf{p}}_{ee}^{w} = \begin{bmatrix} -0.15 & -3.0 & -1.5 \end{bmatrix}^{\top} \tag{B.1}$$

$$\overline{\mathbf{p}}_{ee}^{w} = \begin{bmatrix} 3.0 & 3.0 & 3.0 \end{bmatrix}^{\top} \tag{B.2}$$

$$\underline{\dot{\mathbf{p}}}_{ee}^{w} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \end{bmatrix}^{\top} = -\overline{\dot{\mathbf{p}}}_{ee}^{w} \tag{B.3}$$

B.1.1 Position Task

Task Specification Table					
Priority	Type	Ref. source	Ctrl.p (M,D,K)	Frame	ineq. constr
1	cart. pos	DMP-MPC	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{I}_3 \\ \mathbf{0}_3\end{bmatrix}$ $(m = 3)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	joint pos	$\mathbf{0}_{8\times 1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table B.1: The highest priority task is the Cartesian space end-effector position, $\mathbf{p}_{ee}^{w}$, indicated by the type and the frame. The secondary task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

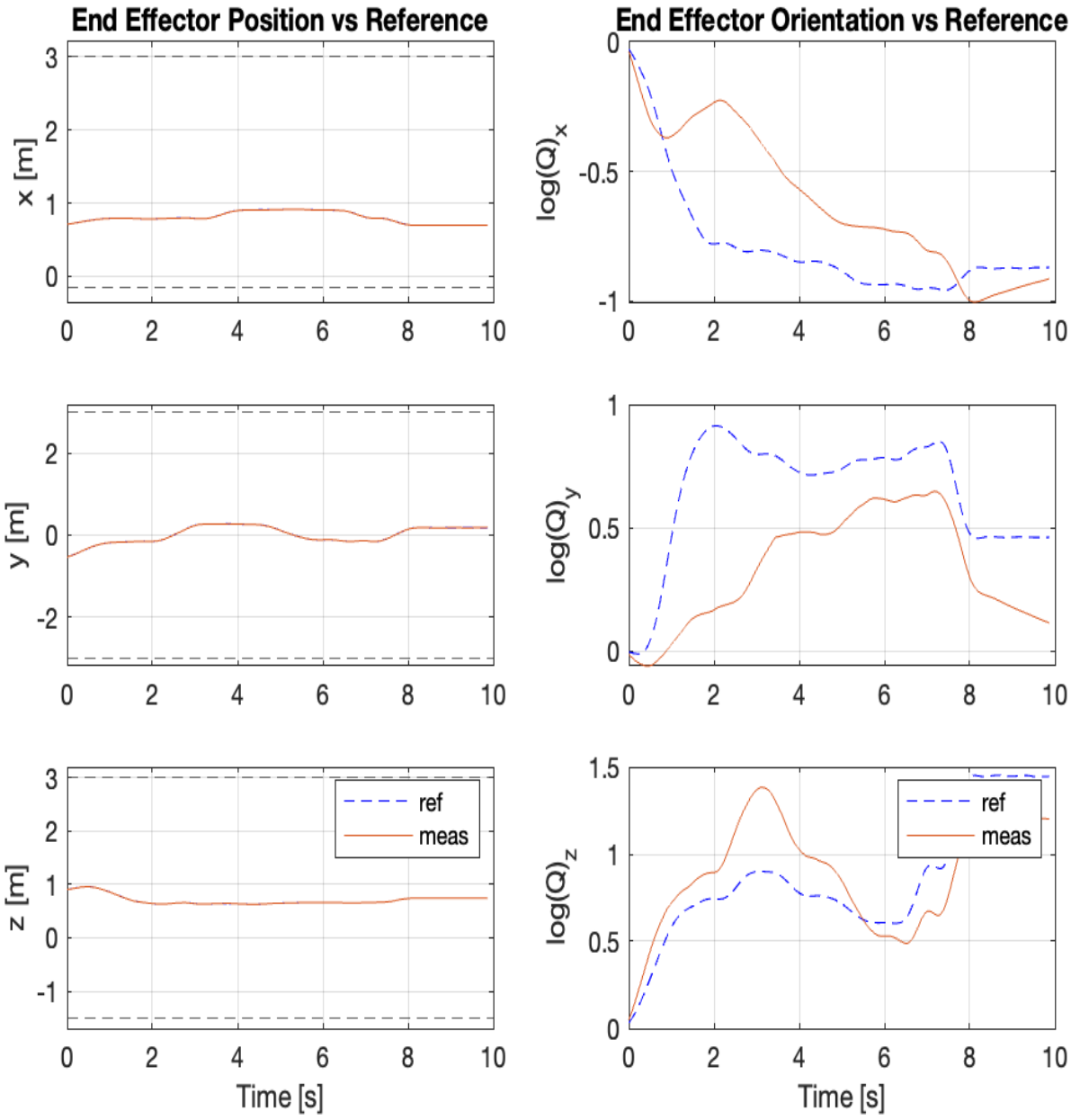


Figure B.2: The task specification is given by Table. 3.2. The end-effector position $\mathbf{p}_{ee}^{w} \in \mathbb{R}^3$ has the highest priority and orientation control is not considered. The graphs in the left column depict the DMP-MPC-generated position reference (dashed blue), and the measured position (red). The robot shows a correct tracking performance. The upper and lower dashed lines in the right column graphs denote the upper and lower position limits, Eqs. (B.4), (B.5). The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured orientation (red). The reference is, as expected, not tracked, compare this with the corresponding graphs in Figs.3.15, 3.19.

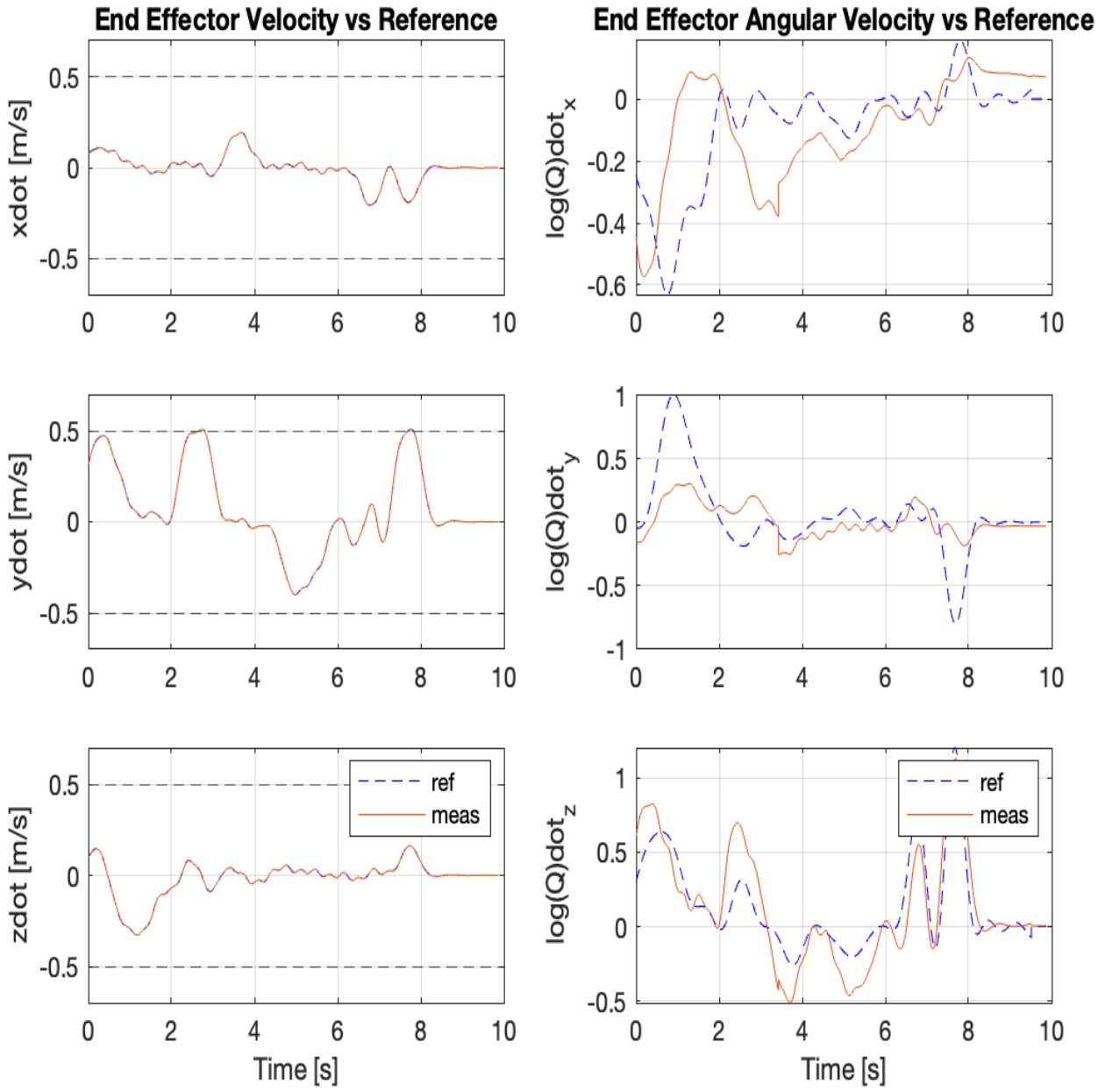


Figure B.3: The task specification is given by Table. 3.2. The end-effector position $\mathbf{p}_{ee}^{w} \in \mathbb{R}^3$ has the highest priority and orientation control is not considered. The graphs in the left column depict the DMP-MPC-generated velocity reference (dashed blue), and the measured velocity (red). The robot shows a good tracking performance. The upper and lower dashed lines in the right column graphs denote the upper and lower velocity limits Eqs. (B.4), (B.5). The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured angular velocity (red). Since the orientation is not part of the task description, the reference is not tracked, compare this with the corresponding graphs in Figs.3.16, 3.20.

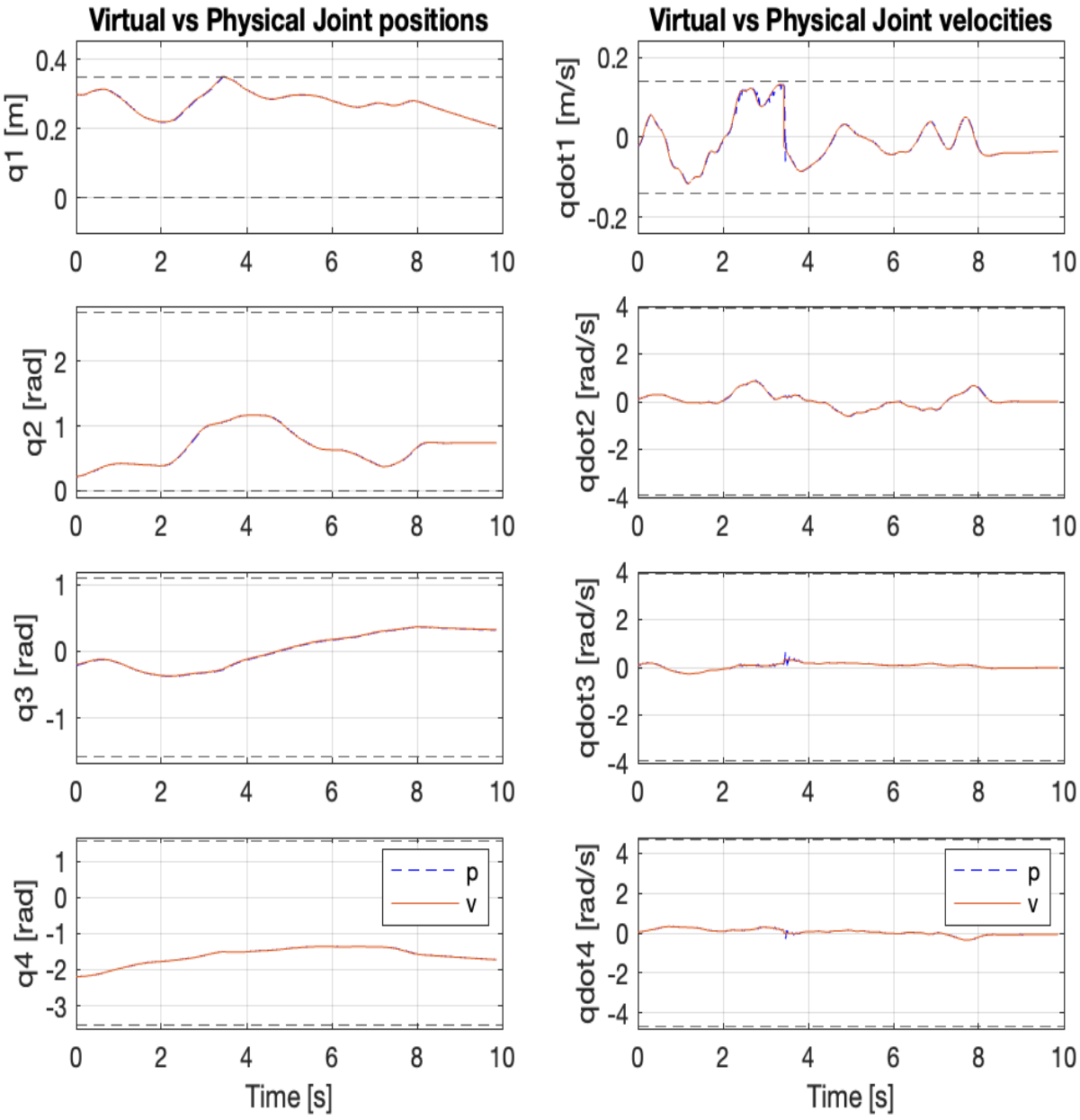


Figure B.4: The task specification is given by Table. 3.2. The graphs in the right column show virtual and physical joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec.2.1.2). The corresponding graphs for the last four joints are seen in Fig.3.14.

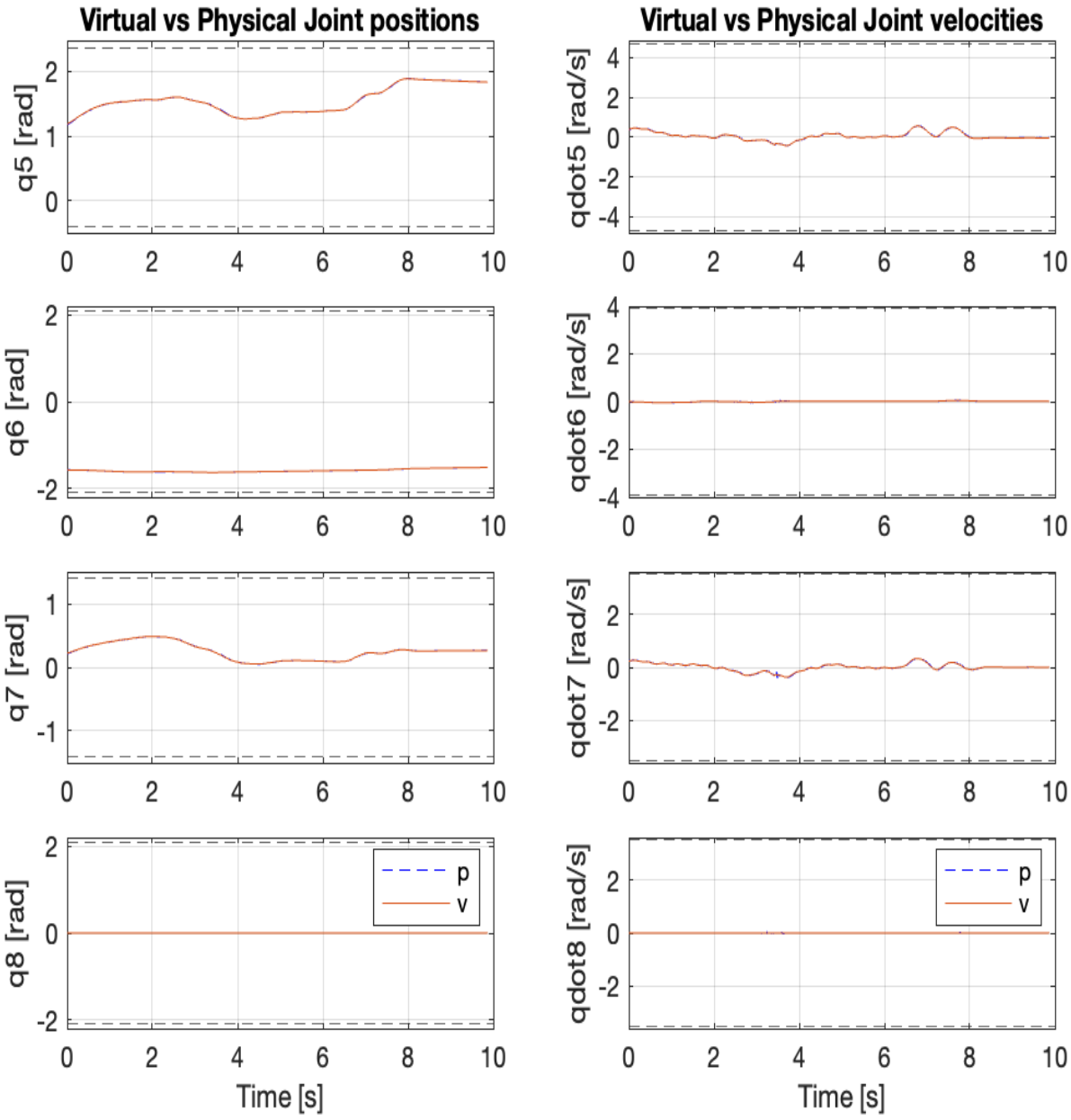


Figure B.5: The task specification is given by Table. 3.2. The graphs in the right column show virtual and physical joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec.2.1.2). The corresponding graphs for the first four joints are seen in Fig.3.13.

B.1.2 Pose Task

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pose	DMP-MPC DMP	(1,80,300)	$\mathcal{F}_1 = \boldsymbol{I}_6$ $(m = 6)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	joint pos	$\mathbf{0}_{8\times1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table B.2: Pose Task Specification. A DMP-MPC generates the position reference and an ordinary DMP for orientation; the frame indicates that it's a full pose task. In this case, there are no orientation constraints (limits). The secondary task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

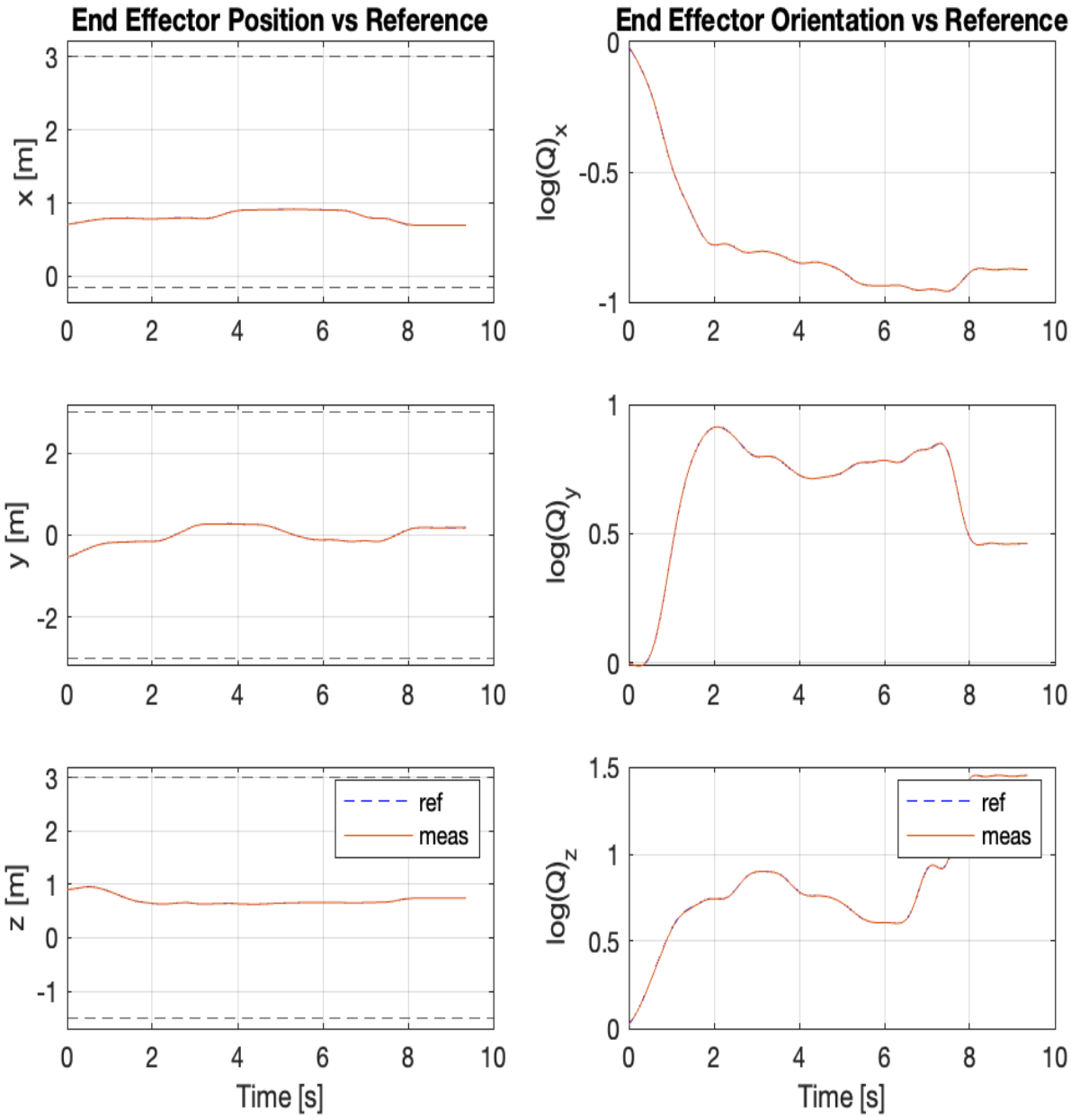


Figure B.6: Task specification is given by Table. 3.3. The graphs in the left column depict the DMP-MPC-generated position reference (blue dashed line) and the measured position (red); upper and lower dashed lines denote the corresponding position limits. The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured orientation (red). In contrast with the position task, the orientation reference is tracked well, see Figs.3.11, 3.19.

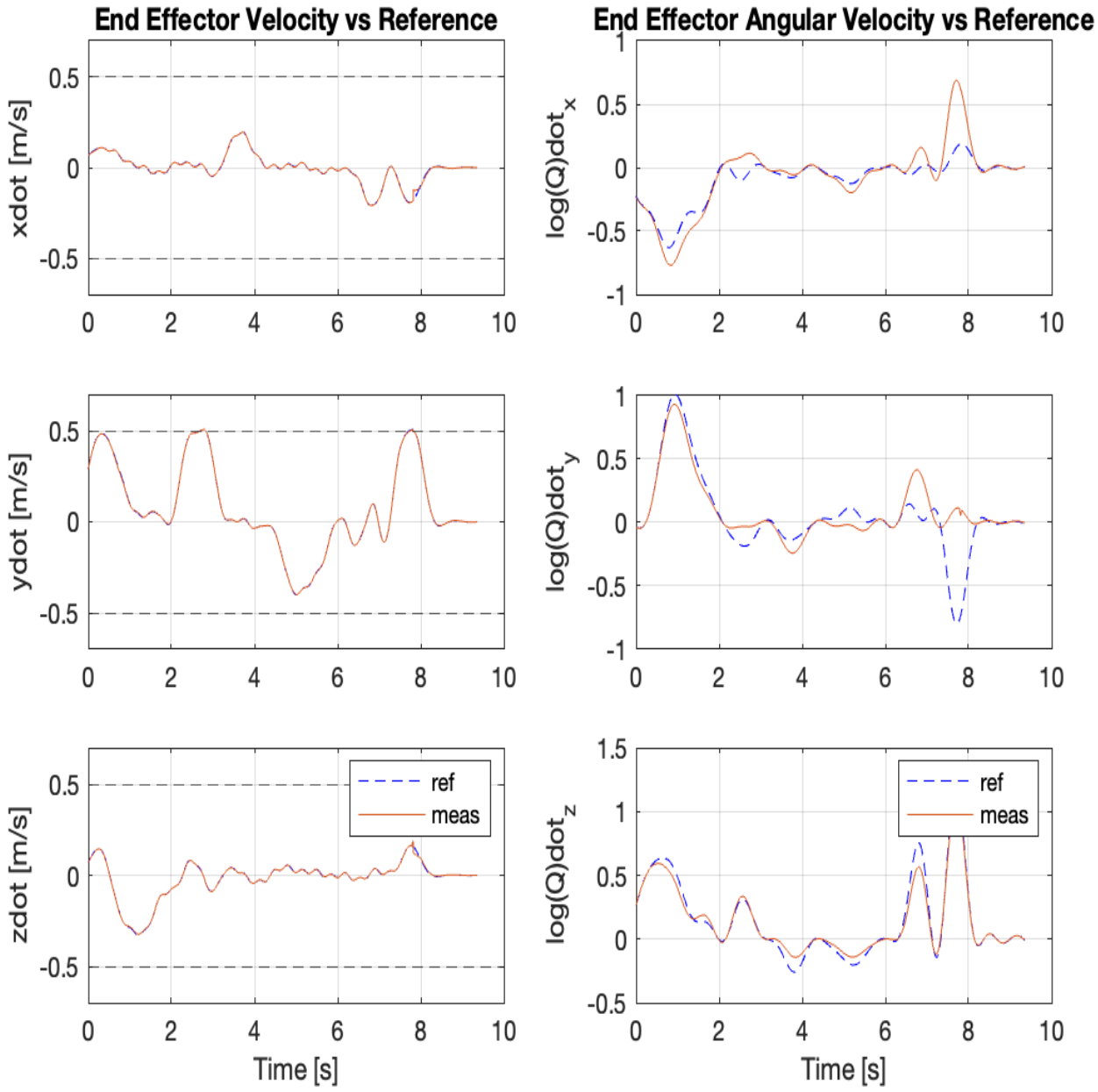


Figure B.7: Task specification is given by Table. 3.3. The graphs in the left column depict the DMP-MPC-generated velocity reference (blue dashed line) and the measured velocity (red); upper and lower dashed lines denote the corresponding velocity limits. The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured angular velocity (red). The orientation reference is tracked well, compare this with the right graphs in Figs.3.12, 3.20.

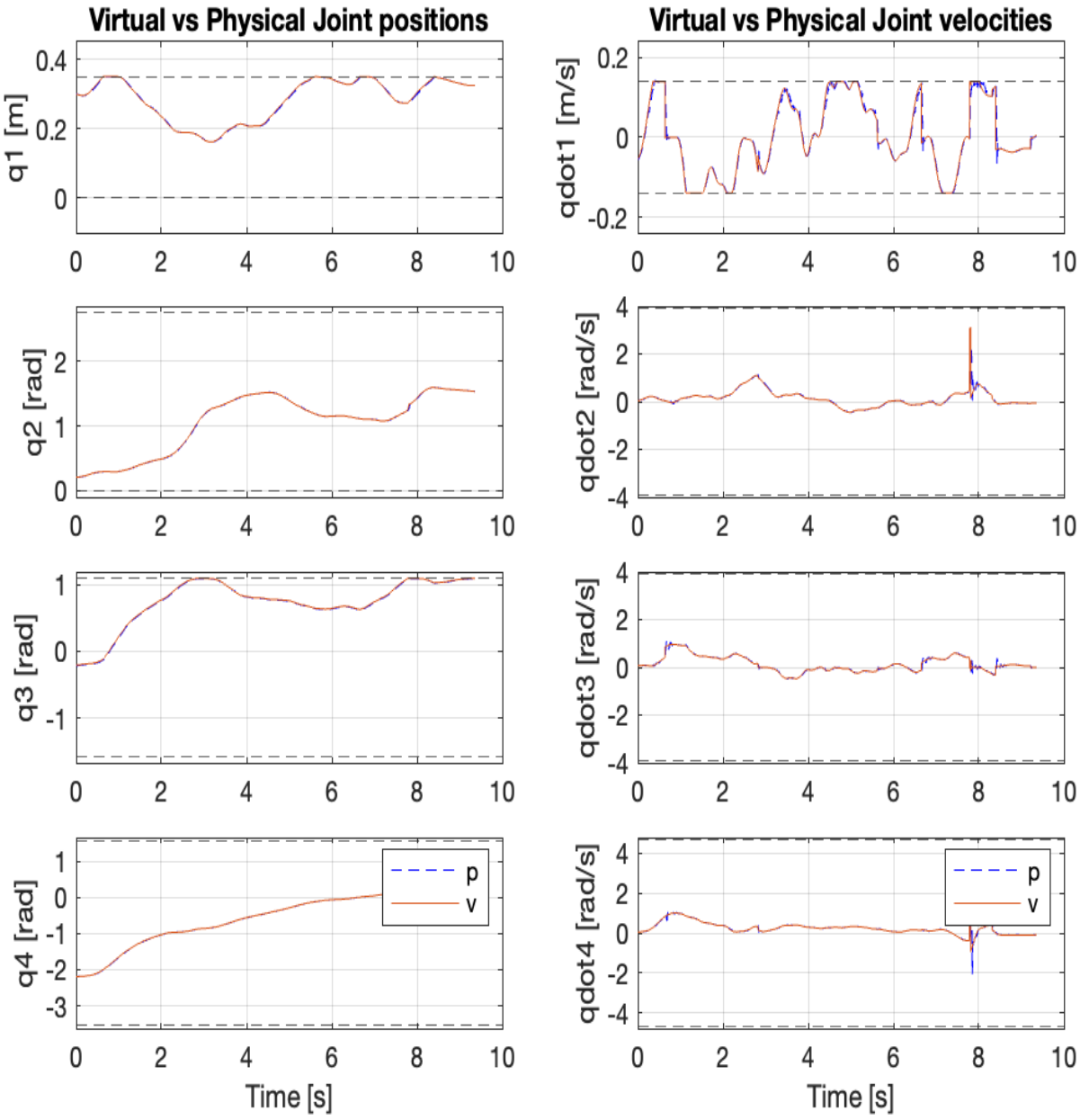


Figure B.8: The task specification is given by Table. 3.3. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec.2.1.2). The corresponding graphs for the last four joints are seen in Fig.B.9.

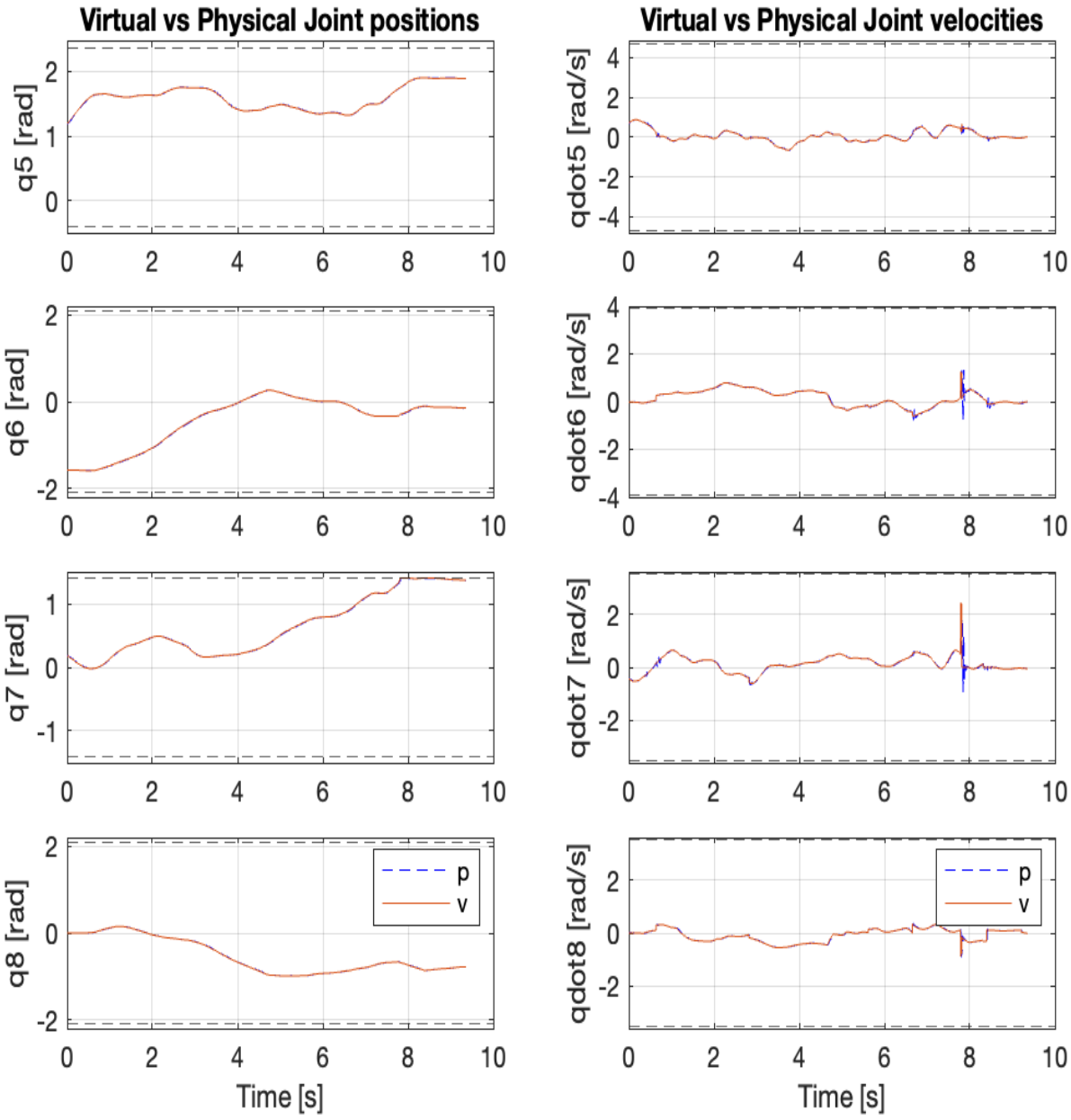


Figure B.9: The task specification is given by Table. 3.3. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec.2.1.2). The corresponding graphs for the first four joints are seen in Fig.B.8.

B.1.3 Pose Task with different hierarchies

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pos	DMP-MPC	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{I}_3 \\ \mathbf{0}_3\end{bmatrix}$ $(m=3)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	cart. or	DMP	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{0}_3 \\ \mathbf{I}_3\end{bmatrix}$ $(m=3)$	
3	joint pos	$\mathbf{0}$	(0,80,0)	$\mathcal{F}_3 = \boldsymbol{I}_8$ $(m=8)$	Eq. (2.54)

Table B.3: Task Specification with a primary position task, secondary orientations task, and thirdly, a joint task. The task dimension is indicated by the corresponding frame. A DMP-MPC generates the position reference and an ordinary DMP generates the orientation reference. The third task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

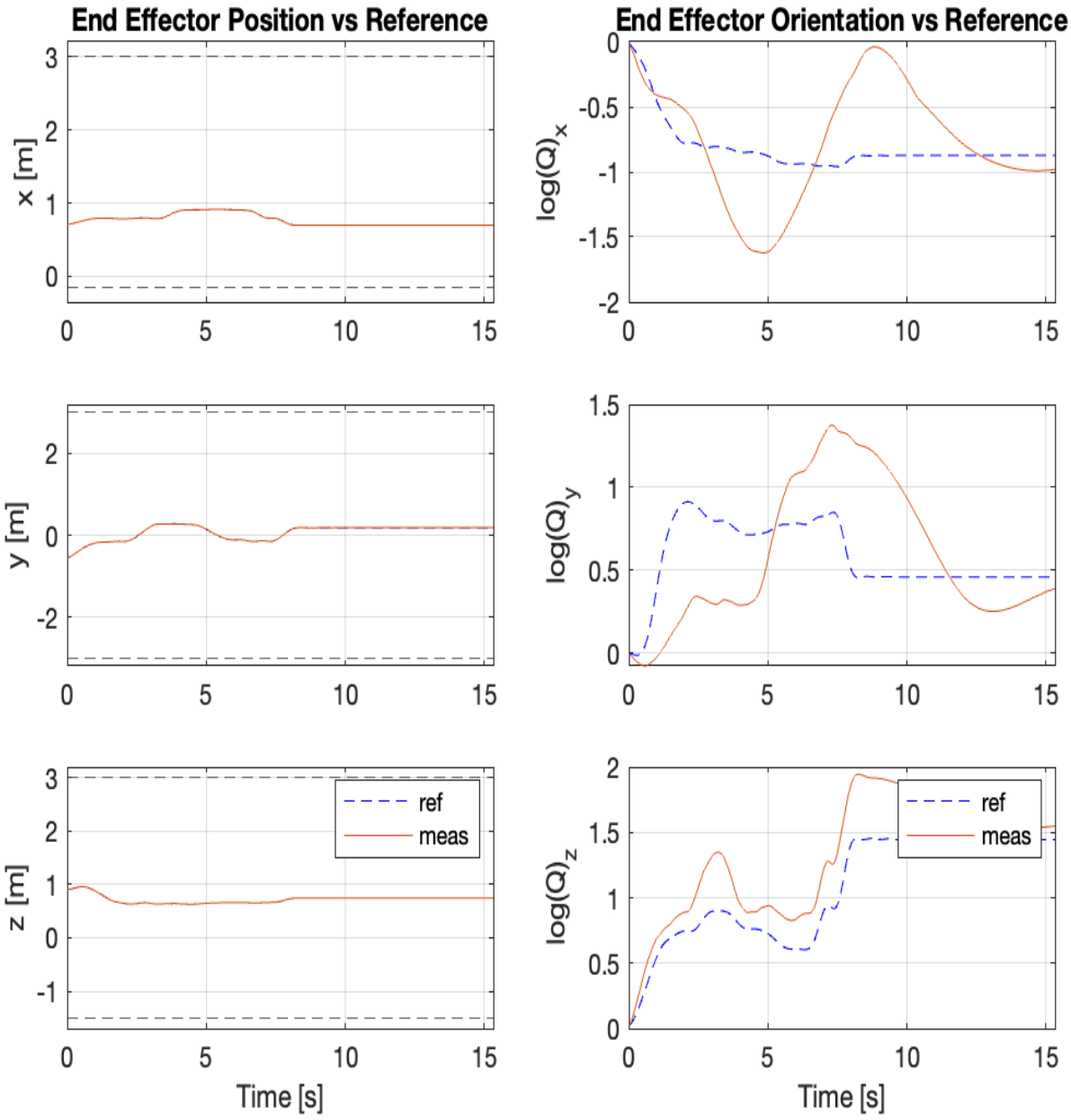


Figure B.10: Task specification is given by Table. 3.4. The graphs in the left column depict the DMP-MPC-generated position reference (blue dashed line) and the measured position (red); upper and lower dashed lines denote the corresponding position limits. The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured orientation (red). The performance of the orientation reference tracking is lower than the position tracking, compare this with the corresponding graphs in Figs.3.11, 3.15.

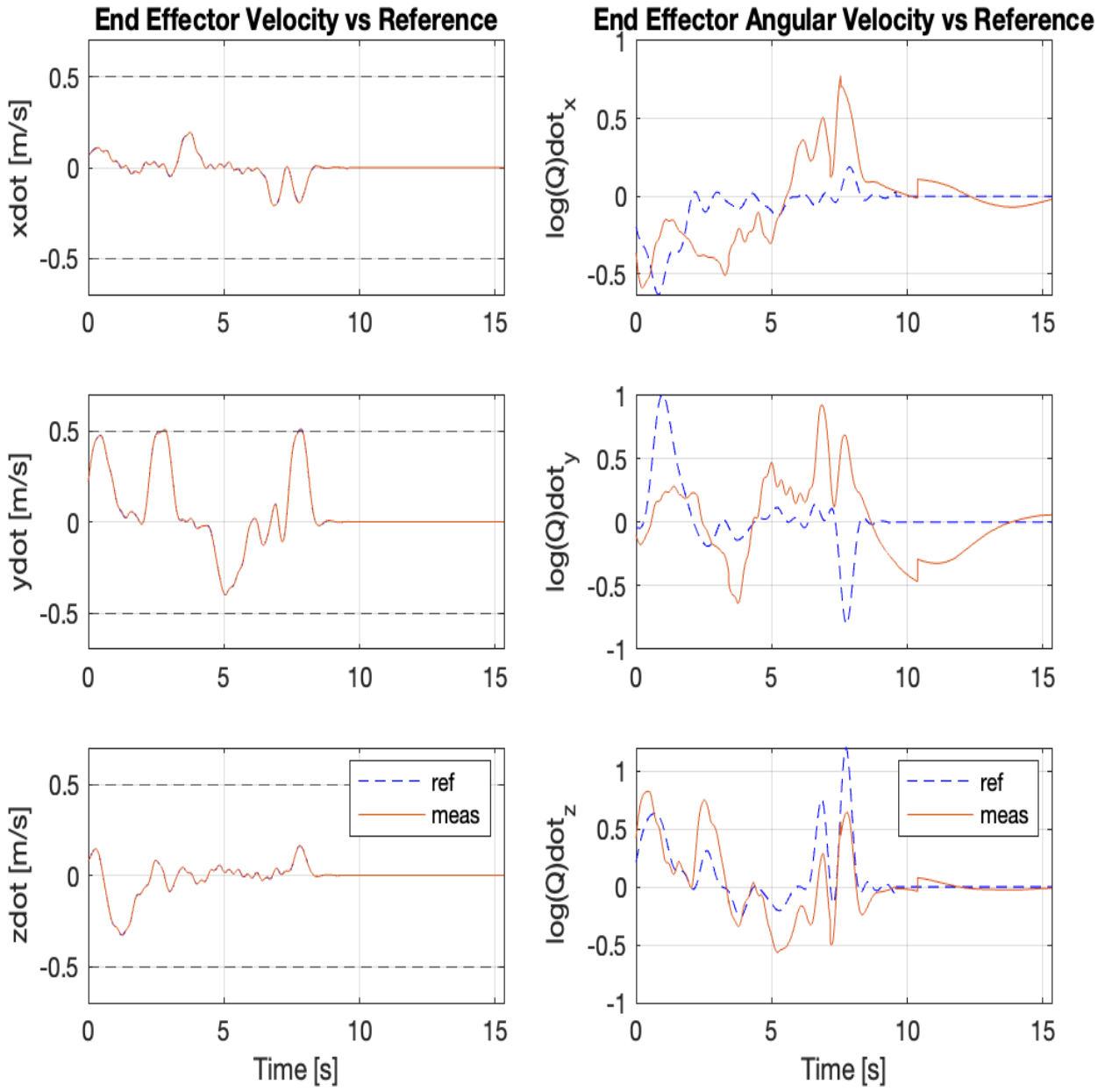


Figure B.11: Task specification is given by Table. 3.4. The graphs in the left column depict the DMP-MPC-generated velocity reference (blue dashed line) and the measured velocity (red); upper and lower dashed lines denote the corresponding velocity limits. The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured angular velocity (red). The orientation reference is tracked somewhat, compare this with the right graphs in Figs.3.12, 3.16.

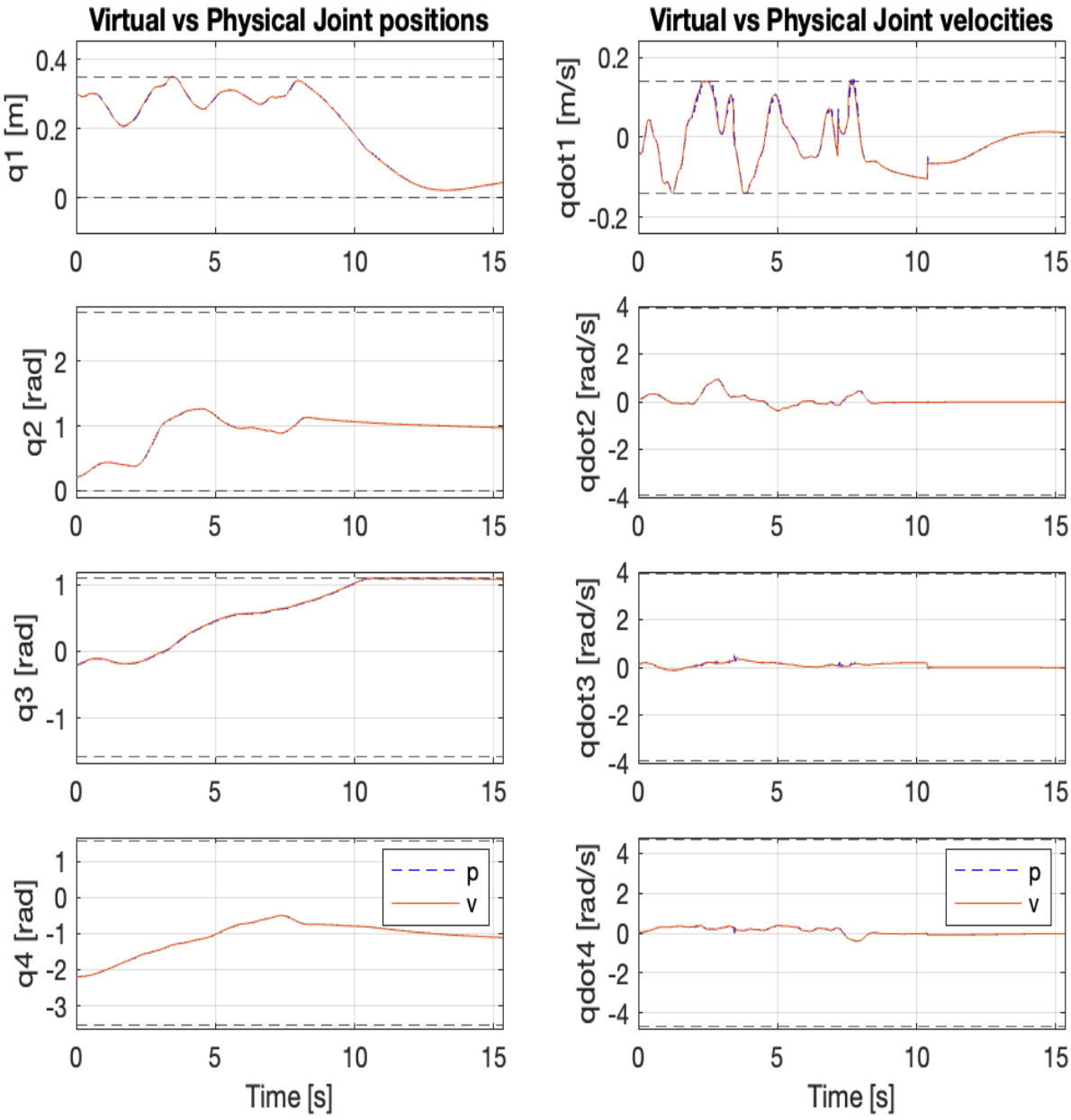


Figure B.12: The task specification is given by Table. 3.4. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec.2.1.2). The corresponding graphs for the last four joints are seen in Fig.3.22.

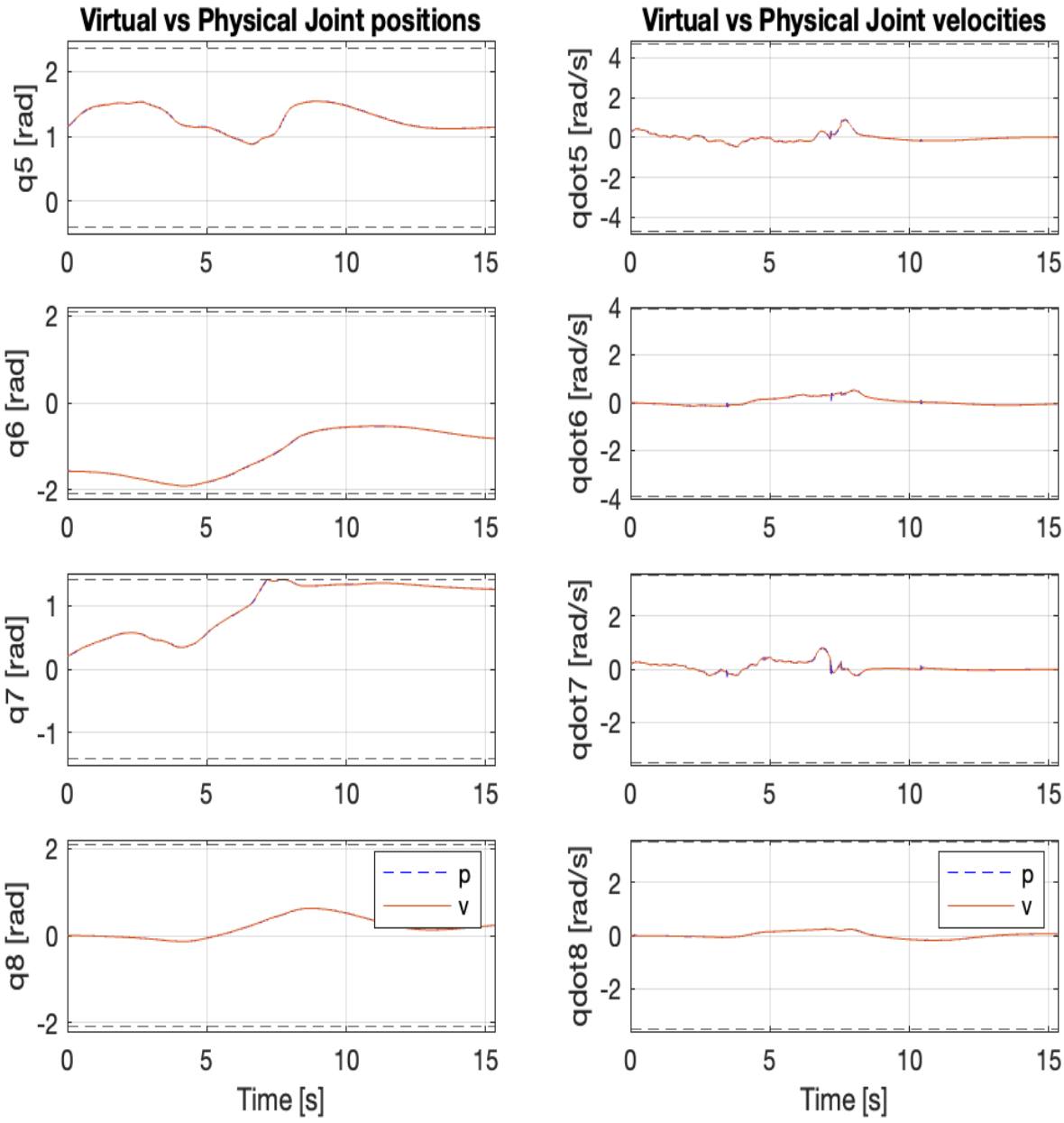


Figure B.13: The task specification is given by Table. 3.4. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec.2.1.2). The corresponding graphs for the first four joints are seen in Fig.3.21.

B.2 Trace Stack of cubes

In this section, we compare the task tracking for the same task (taught using KT) with different task specifications (hierarchies). The task is to move into a grasping pose, see Fig. B.1. The teaching and execution of this task can be seen in the video.

Figure B.14: Depiction of the trace cube stack task. The task was to teach the robot to trace the stack of cubes by first going to a grasp pose for cube 'A' and then trace the stack down to cube 'B' and up again.

Eqs. (B.4), (B.5) and (B.6) give the lower, and upper Cartesian space position and velocity limits used for the position tasks below.

$$\underline{\mathbf{p}}_{ee}^{w} = \begin{bmatrix} -0.15 & -3.0 & -1.5 \end{bmatrix}^{\top} \tag{B.4}$$

$$\overline{\mathbf{p}}_{ee}^{w} = \begin{bmatrix} 3.0 & 3.0 & 3.0 \end{bmatrix}^{\top} \tag{B.5}$$

$$\underline{\dot{\mathbf{p}}}_{ee}^{w} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \end{bmatrix}^{\top} = -\overline{\dot{\mathbf{p}}}_{ee}^{w} \tag{B.6}$$

B.2.1 Position Task

Task Specification Table					
Priority	Type	Ref. source	Ctrl.p (M,D,K)	Frame	ineq. constr
1	cart. pos	DMP-MPC	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{I}_3 \\ \mathbf{0}_3\end{bmatrix}$ $(m = 3)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	joint pos	$\mathbf{0}_{8\times 1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table B.4: The highest priority task is the Cartesian space end-effector position, $\mathbf{p}_{ee}^{w}$, indicated by the type and the frame. The secondary task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

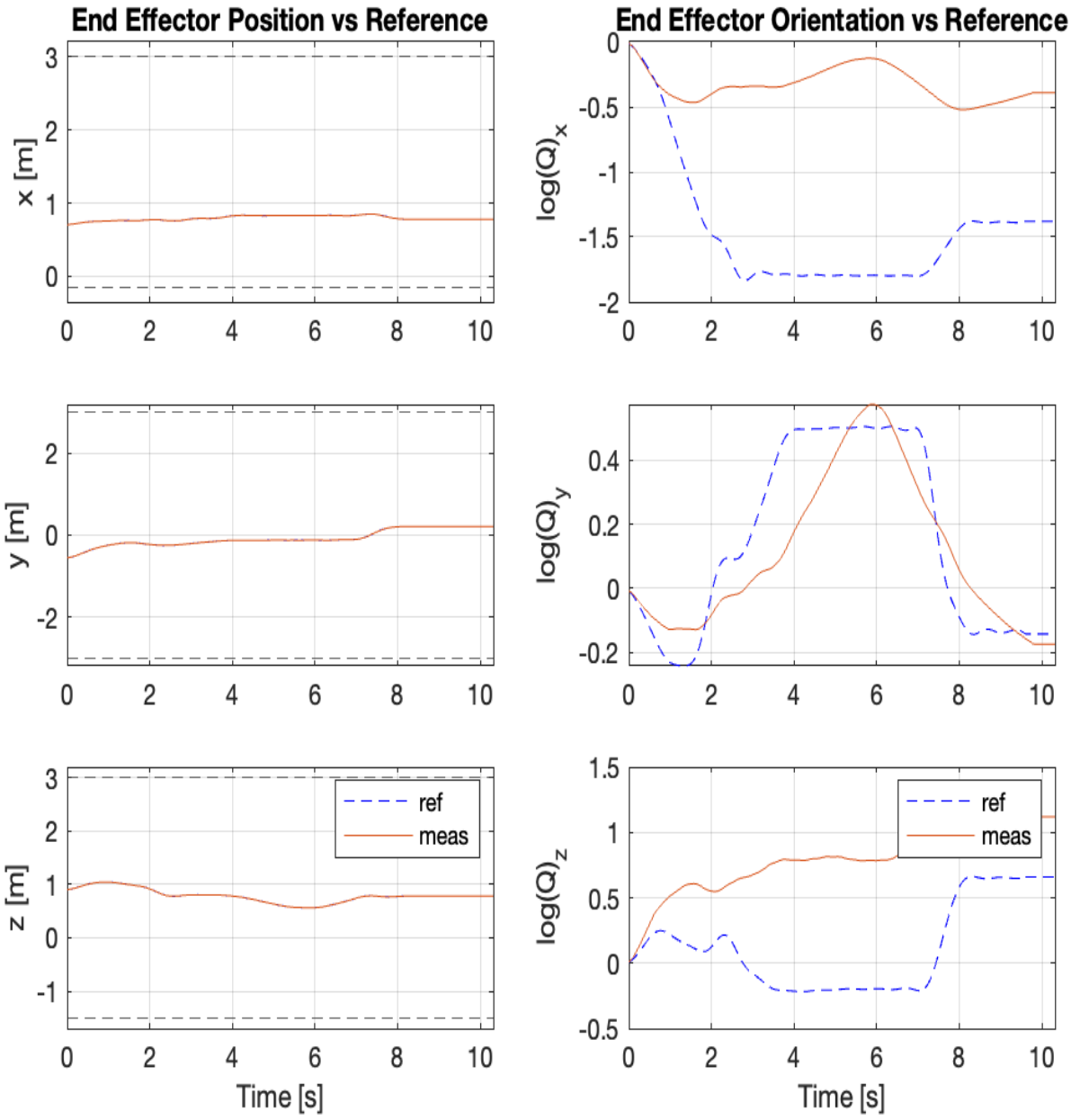


Figure B.15: The task specification is given by Table. 3.2. The end-effector position $\mathbf{p}_{ee}^{w} \in \mathbb{R}^3$ has the highest priority and orientation control is not considered. The graphs in the left column depict the DMP-MPC-generated position reference (dashed blue), and the measured position (red). The robot shows a correct tracking performance. The upper and lower dashed lines in the right column graphs denote the upper and lower position limits, Eqs. (B.4), (B.5). The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured orientation (red). The reference is, as expected, not tracked, compare this with the corresponding graphs in Figs.3.15, 3.19.

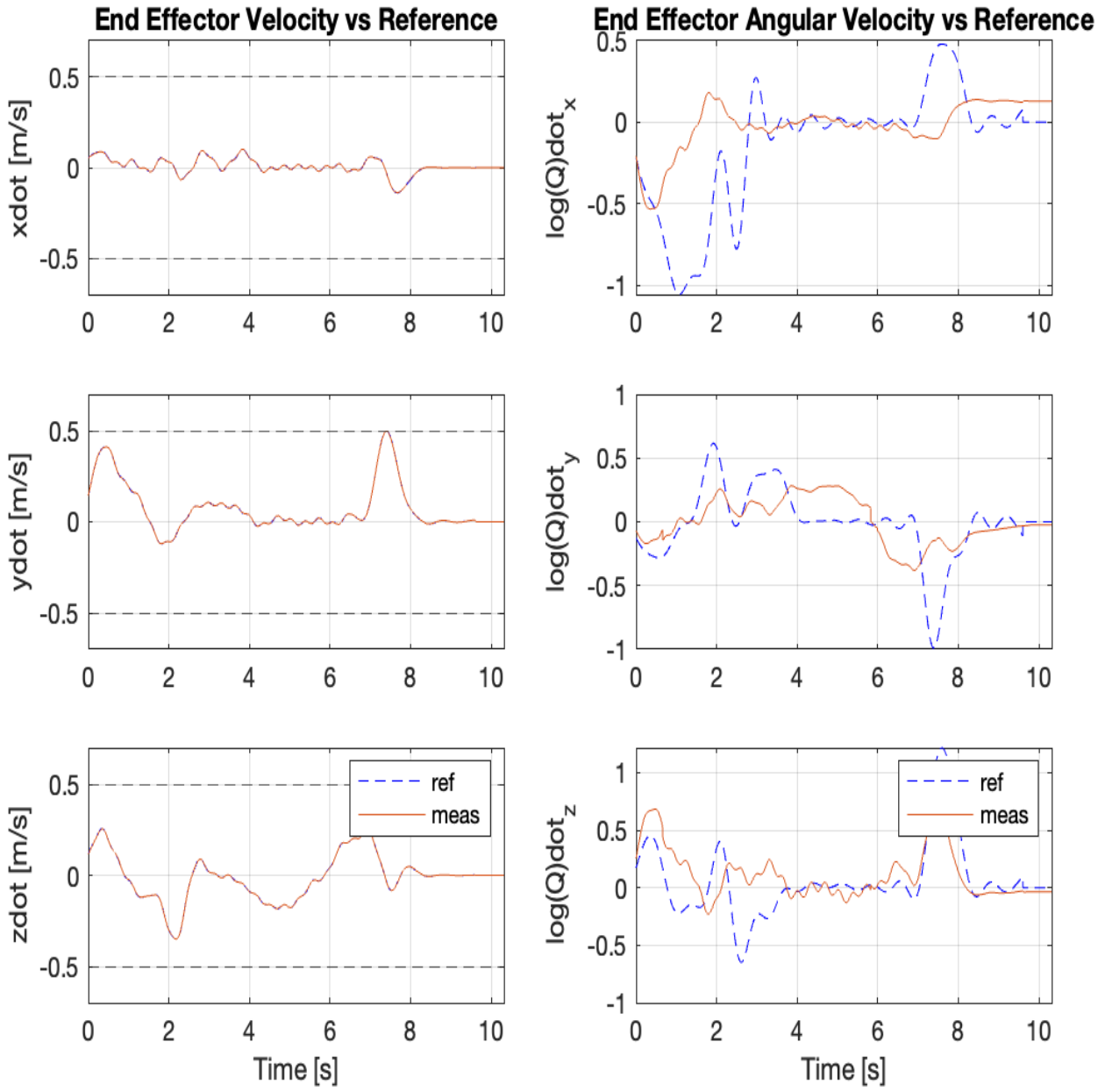


Figure B.16: The task specification is given by Table. 3.2. The end-effector position $\mathbf{p}_{ee}^{w} \in \mathbb{R}^3$ has the highest priority and orientation control is not considered. The graphs in the left column depict the DMP-MPC-generated velocity reference (dashed blue), and the measured velocity (red). The robot shows a good tracking performance. The upper and lower dashed lines in the right column graphs denote the upper and lower velocity limits Eqs. (B.4), (B.5). The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured angular velocity (red). Since the orientation is not part of the task description, the reference is not tracked, compare this with the corresponding graphs in Figs.3.16, 3.20.

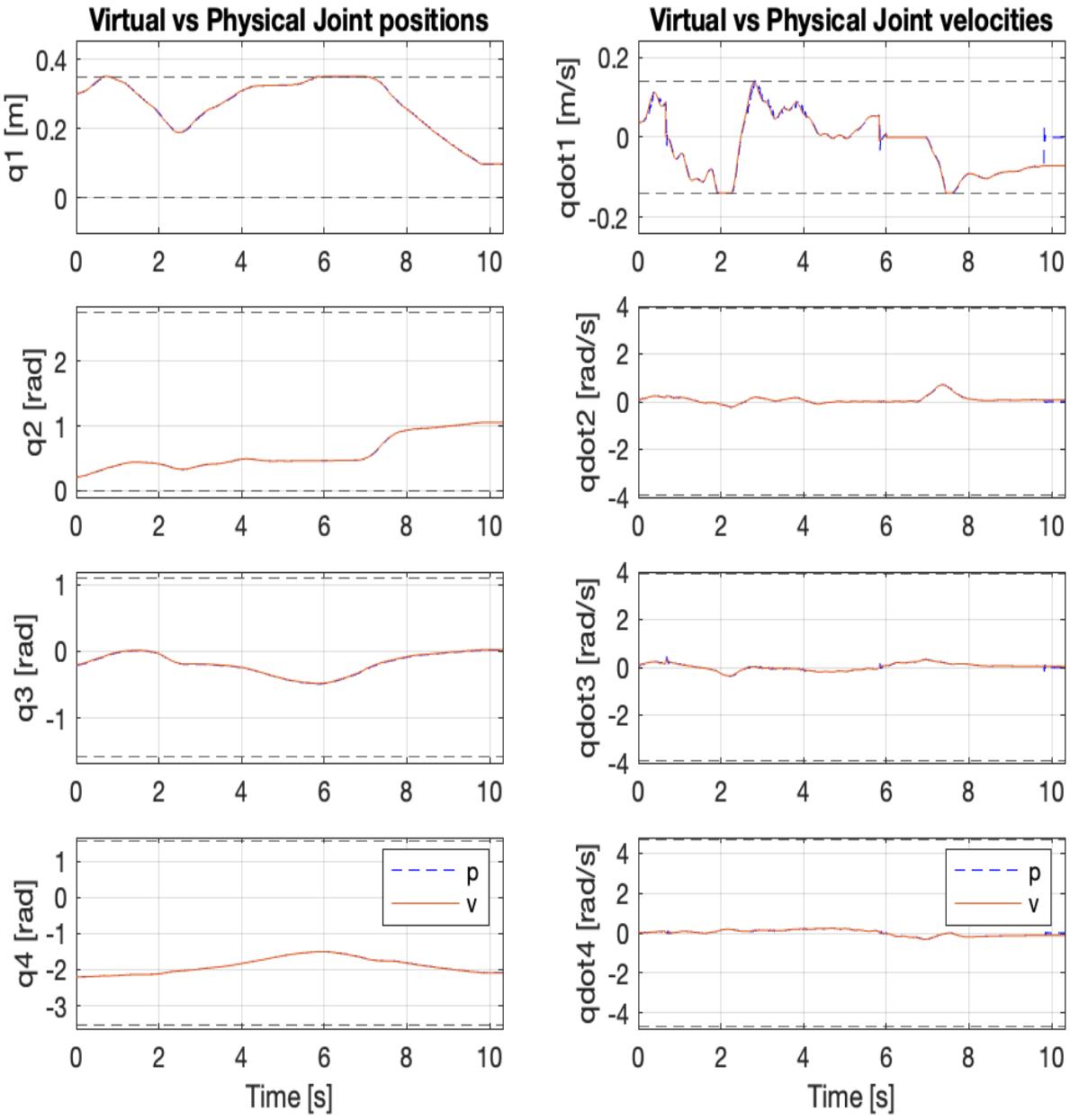


Figure B.17: The task specification is given by Table. 3.2. The graphs in the right column show virtual and physical joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec.2.1.2). The corresponding graphs for the last four joints are seen in Fig.3.14.

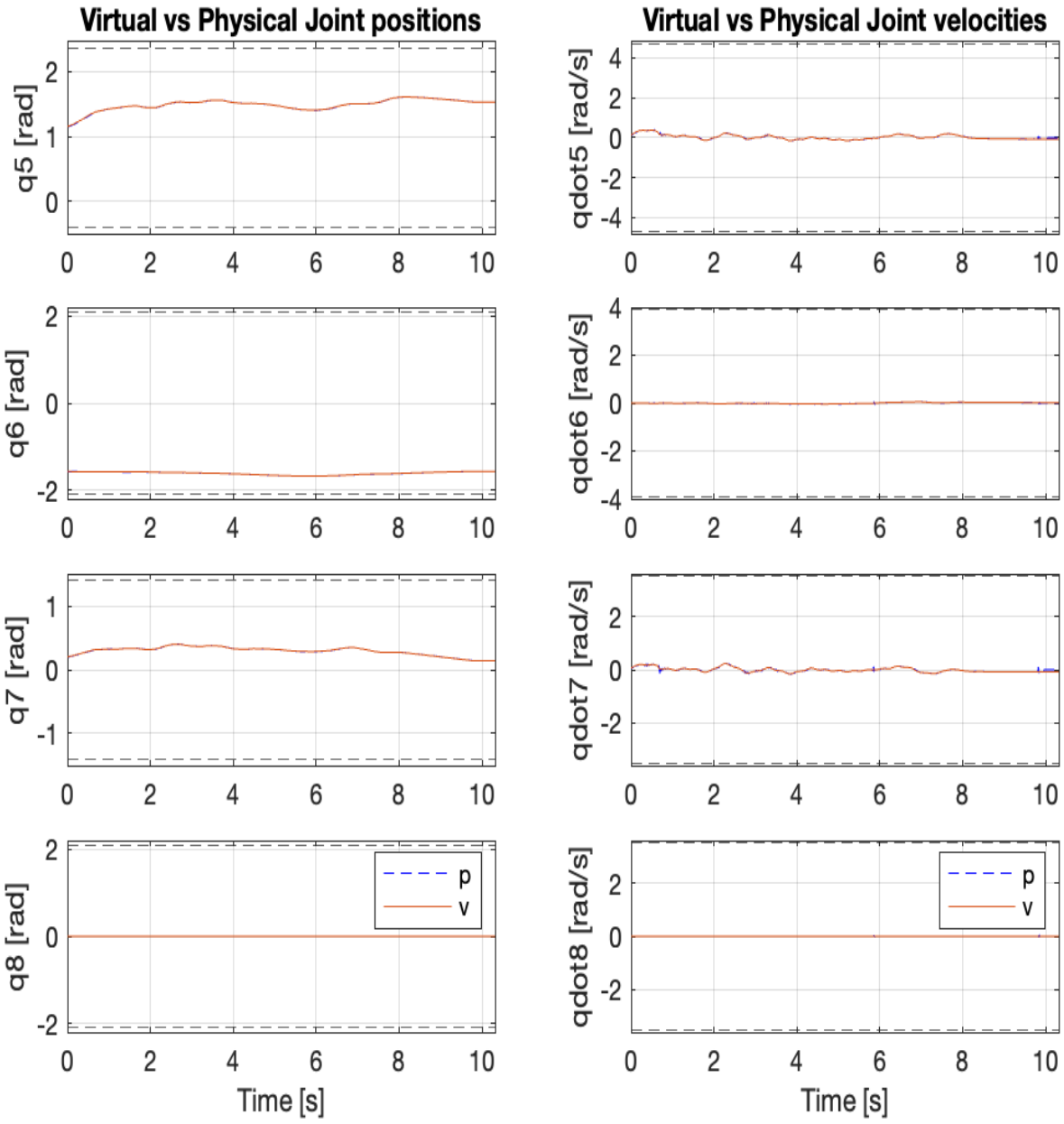


Figure B.18: The task specification is given by Table. 3.2. The graphs in the right column show virtual and physical joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec.2.1.2). The corresponding graphs for the first four joints are seen in Fig.3.13.

B.2.2 Pose Task

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pose	DMP-MPC DMP	(1,80,300)	$\mathcal{F}_1 = \boldsymbol{I}_6$ $(m = 6)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	joint pos	$\mathbf{0}_{8\times1}$	(0,80,0)	$\mathcal{F}_2 = \boldsymbol{I}_8$ $(m = 8)$	Eq. (2.54)

Table B.5: Pose Task Specification. A DMP-MPC generates the position reference and an ordinary DMP for orientation; the frame indicates that it's a full pose task. In this case, there are no orientation constraints (limits). The secondary task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

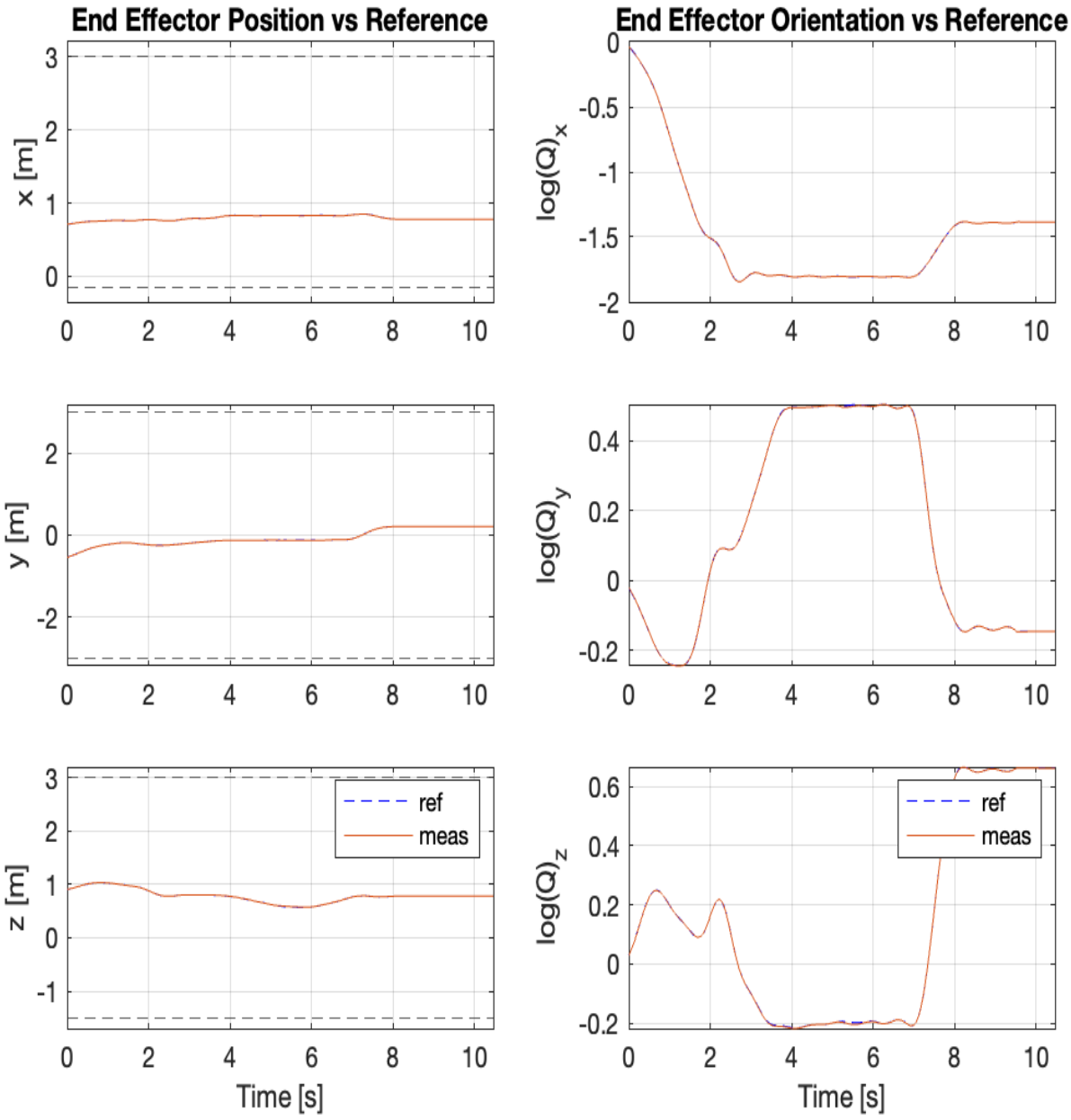


Figure B.19: Task specification is given by Table. 3.3. The graphs in the left column depict the DMP-MPC-generated position reference (blue dashed line) and the measured position (red); upper and lower dashed lines denote the corresponding position limits. The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured orientation (red). In contrast with the position task, the orientation reference is tracked well, see Figs.3.11, 3.19.

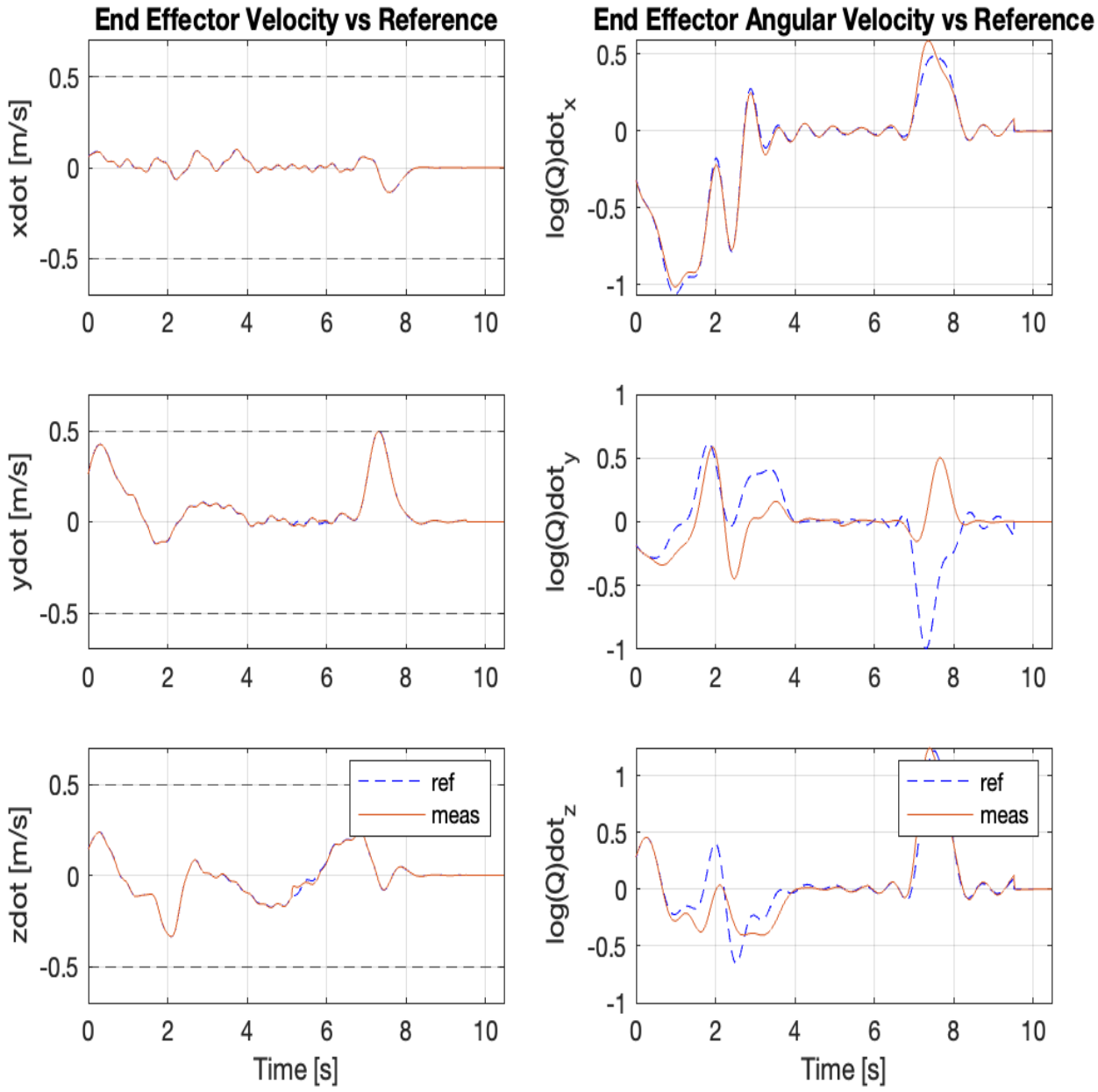


Figure B.20: Task specification is given by Table. 3.3. The graphs in the left column depict the DMP-MPC-generated velocity reference (blue dashed line) and the measured velocity (red); upper and lower dashed lines denote the corresponding velocity limits. The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured angular velocity (red). The orientation reference is tracked well, compare this with the right graphs in Figs.3.12, 3.20.

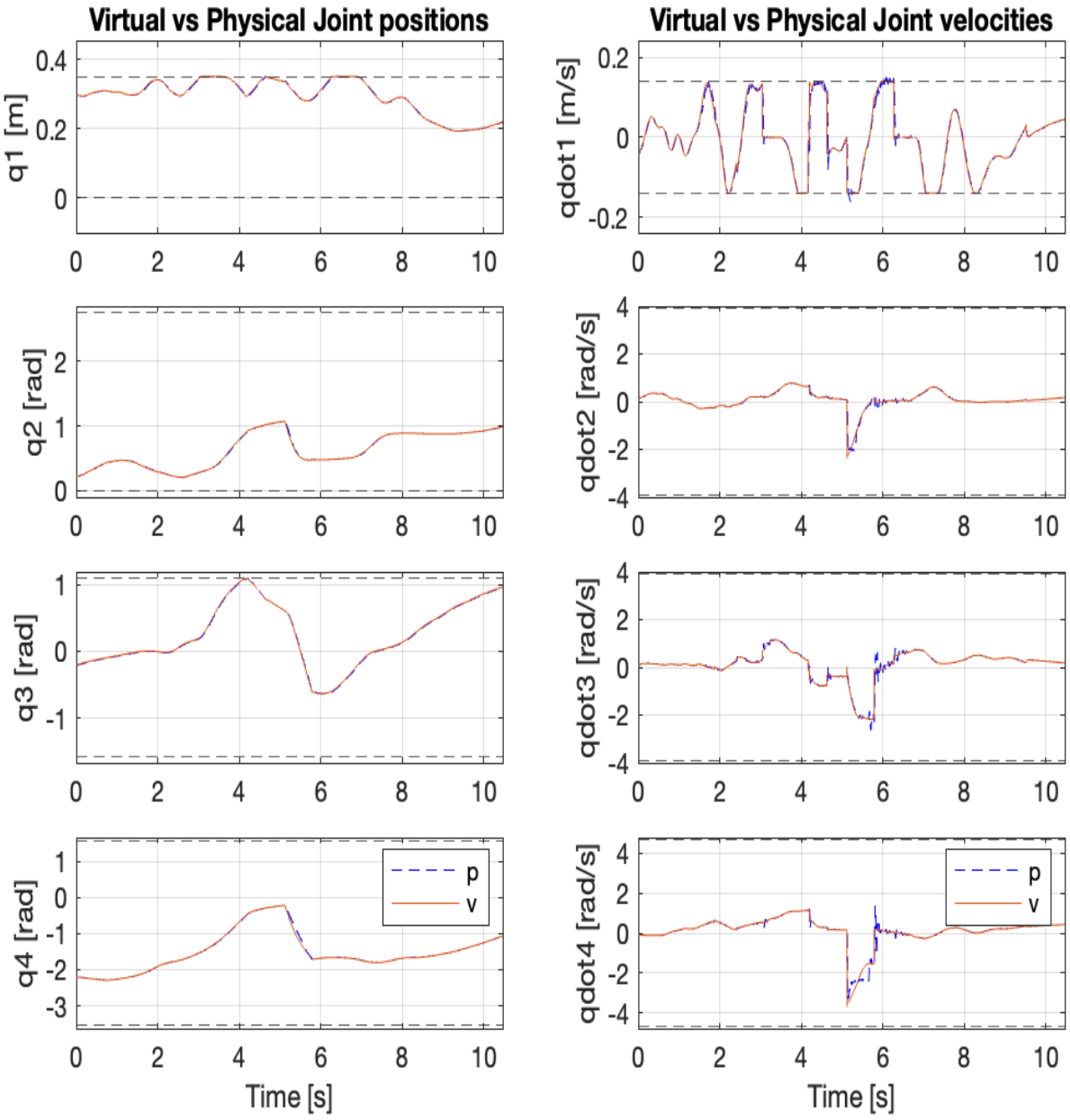


Figure B.21: The task specification is given by Table. 3.3. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec.2.1.2). The corresponding graphs for the last four joints are seen in Fig.B.9.

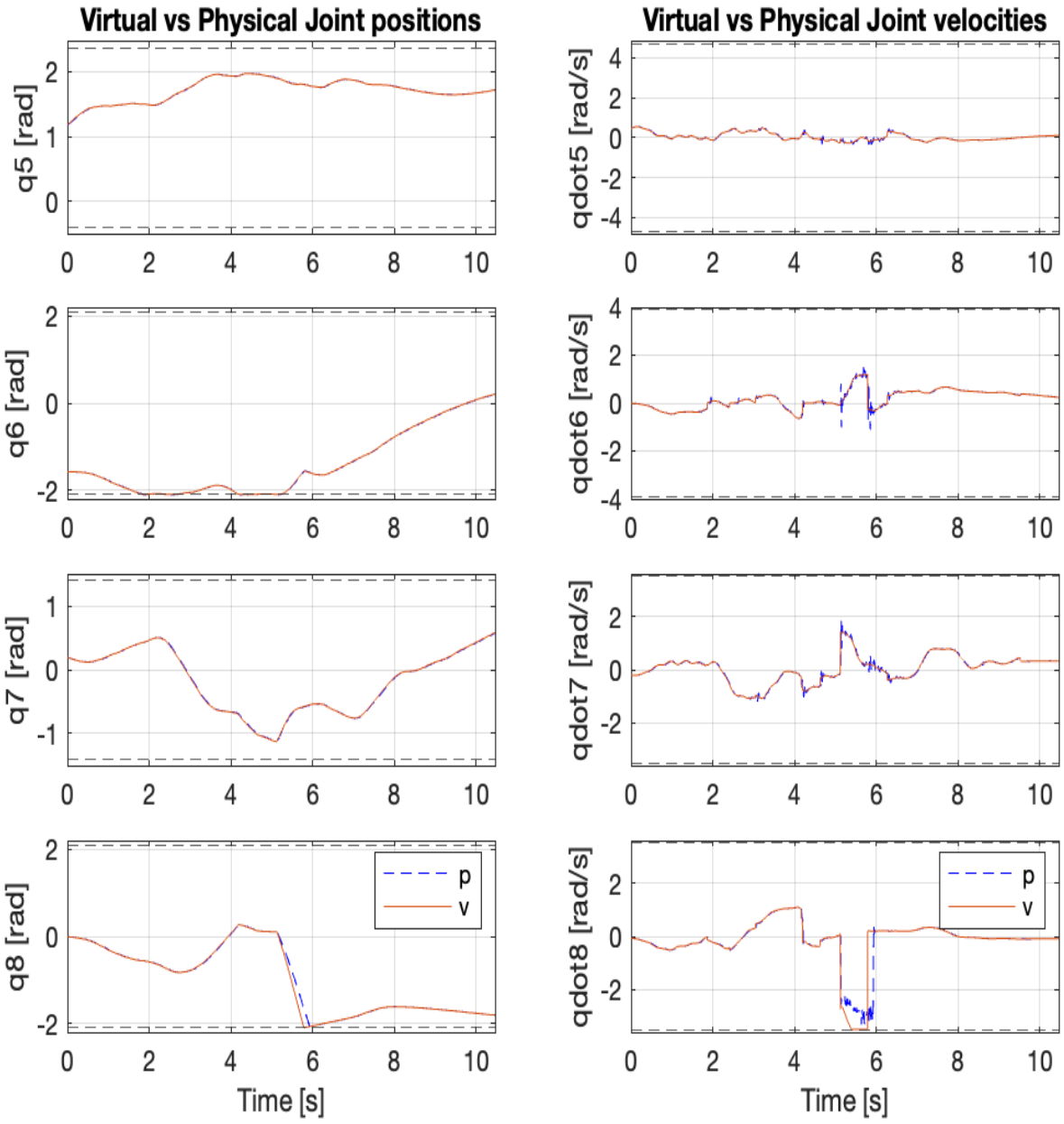


Figure B.22: The task specification is given by Table. 3.3. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec.2.1.2). The corresponding graphs for the first four joints are seen in Fig.B.8.

B.2.3 Pose Task with different hierarchies

Task Specification Table					
Priority	Type	Ref. source	Contrl.p (M,D,K)	Frame	ineq. constr
1	cart. pos	DMP-MPC	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{I}_3 \\ \mathbf{0}_3\end{bmatrix}$ $(m=3)$	$\underline{\mathbf{p}}_{ee}^{w} \leq \mathbf{p}_{ee}^{w} \leq \overline{\mathbf{p}}_{ee}^{w}$ $\underline{\dot{\mathbf{p}}}_{ee}^{w} \leq \dot{\mathbf{p}}_{ee}^{w} \leq \overline{\dot{\mathbf{p}}}_{ee}^{w}$
2	cart. or	DMP	(1,80,300)	$\mathcal{F}_1 = \begin{bmatrix}\mathbf{0}_3 \\ \mathbf{I}_3\end{bmatrix}$ $(m=3)$	
3	joint pos	$\mathbf{0}$	(0,80,0)	$\mathcal{F}_3 = \boldsymbol{I}_8$ $(m=8)$	Eq. (2.54)

Table B.6: Task Specification with a primary position task, secondary orientations task, and thirdly, a joint task. The task dimension is indicated by the corresponding frame. A DMP-MPC generates the position reference and an ordinary DMP generates the orientation reference. The third task is a joint task (indicated by the type) that considers all joints (indicated by frame). The joint task is velocity-controlled since the only non-zero control parameter is the damping D. Zero velocity is the reference for the joint task. The inequality constraints from Eq. (2.54) are active for this task.

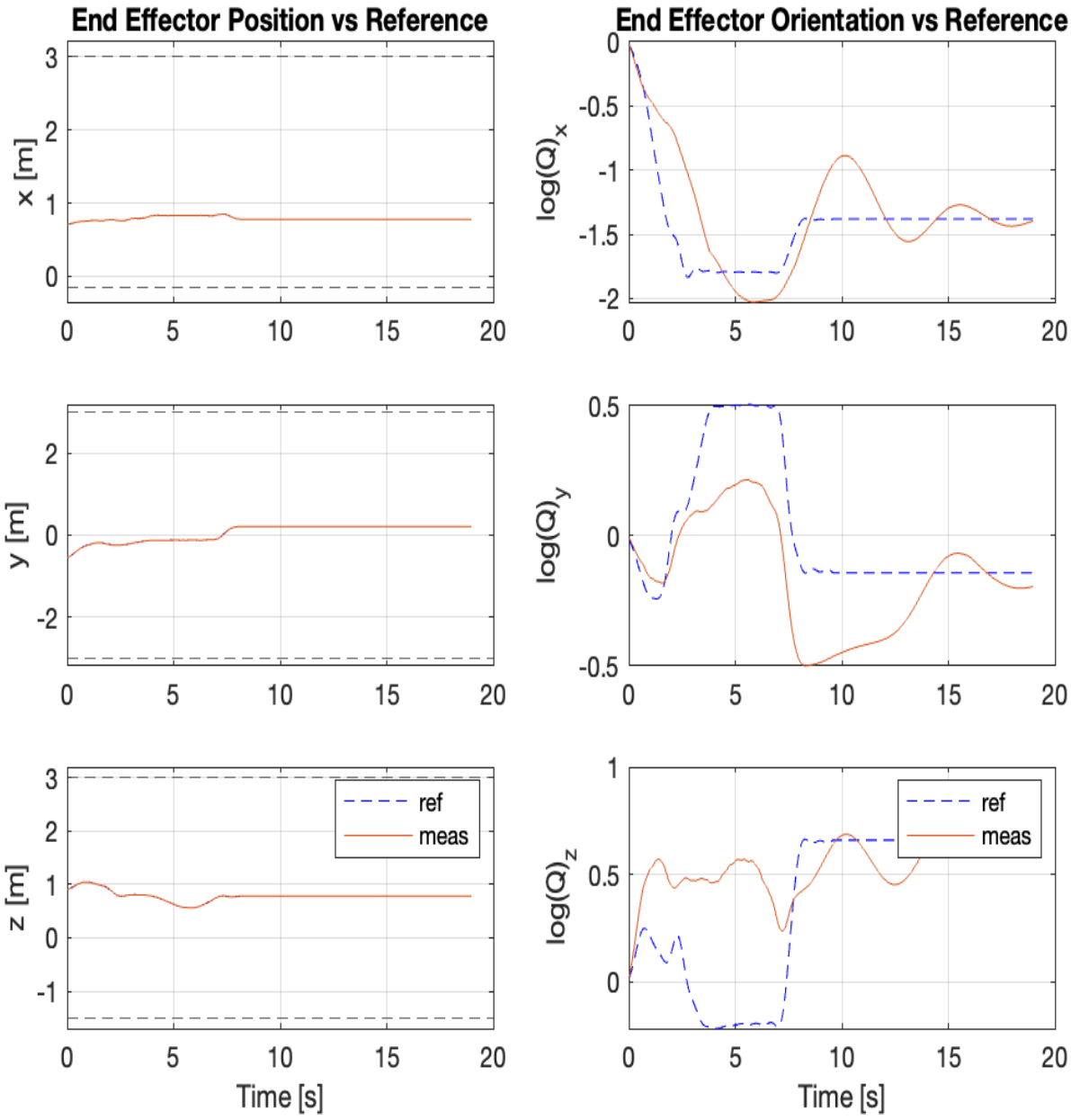


Figure B.23: Task specification is given by Table. 3.4. The graphs in the left column depict the DMP-MPC-generated position reference (blue dashed line) and the measured position (red); upper and lower dashed lines denote the corresponding position limits. The graphs in the right column show the DMP-generated orientation reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured orientation (red). The performance of the orientation reference tracking is lower than the position tracking, compare this with the corresponding graphs in Figs.3.11, 3.15.

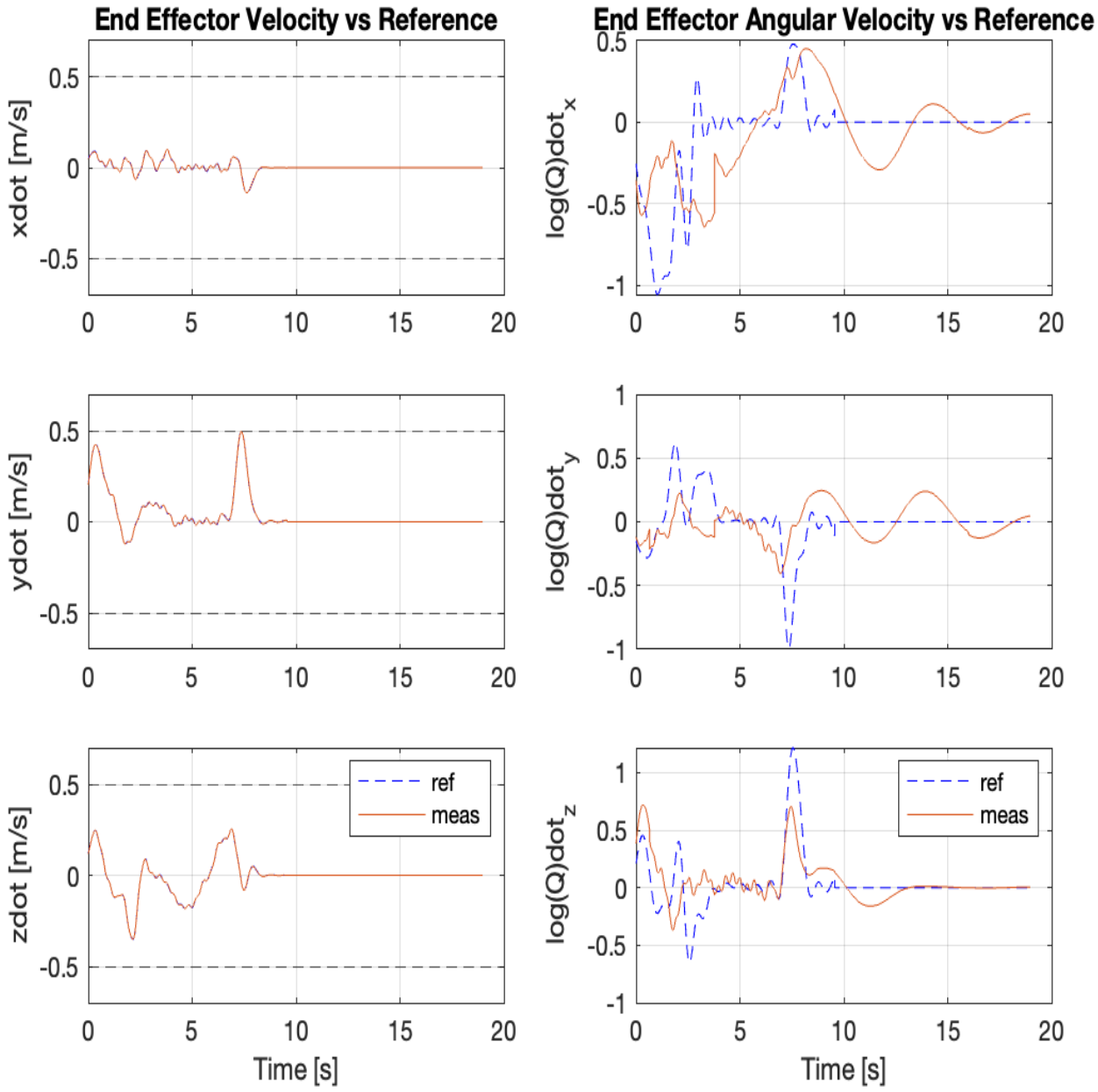


Figure B.24: Task specification is given by Table. 3.4. The graphs in the left column depict the DMP-MPC-generated velocity reference (blue dashed line) and the measured velocity (red); upper and lower dashed lines denote the corresponding velocity limits. The graphs in the right column show the DMP-generated angular velocity reference in the quaternion log space (Sec.2.1.5) (blue dashed line) and the measured angular velocity (red). The orientation reference is tracked somewhat, compare this with the right graphs in Figs.3.12, 3.16.

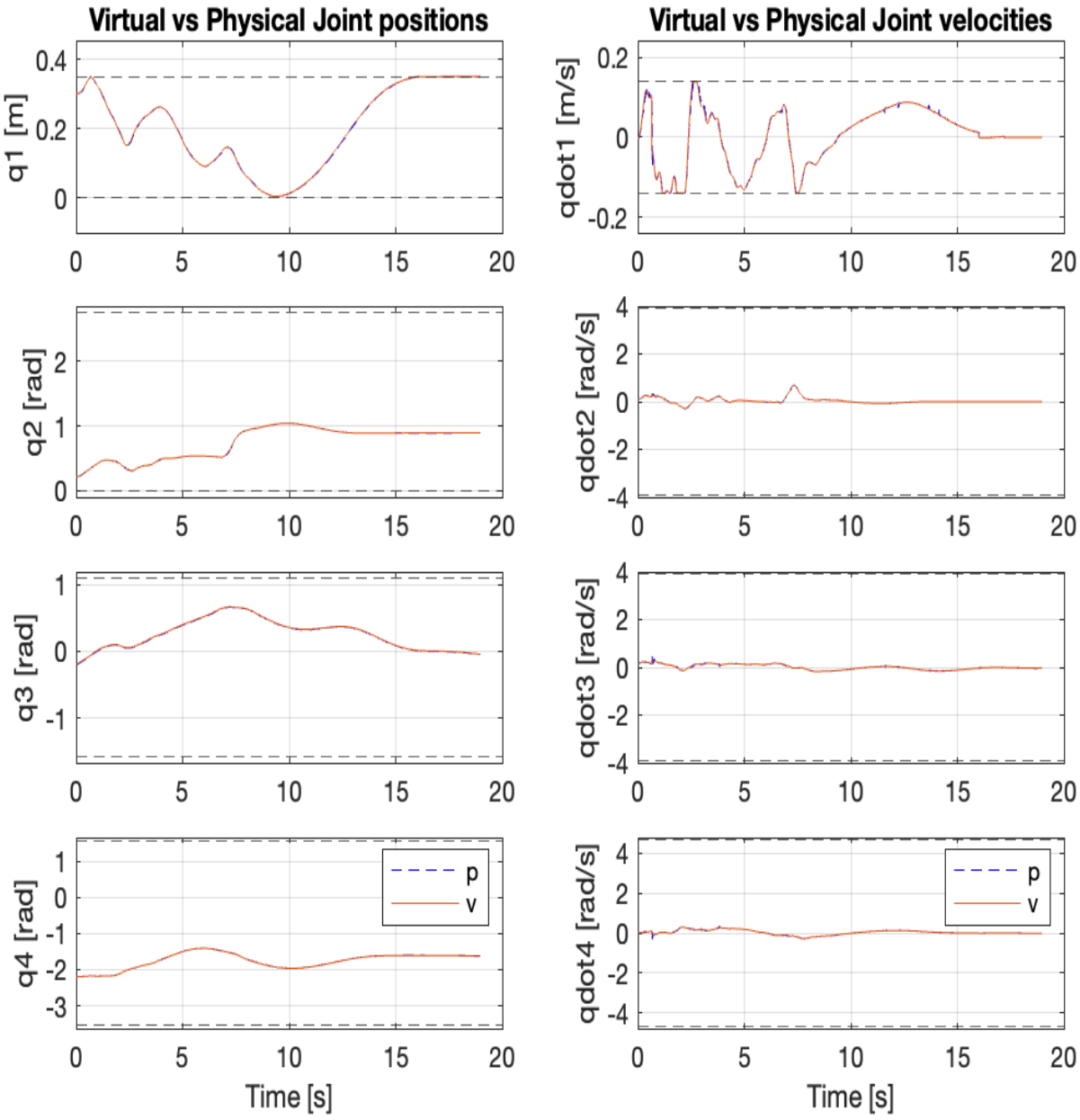


Figure B.25: The task specification is given by Table. 3.4. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the first four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The first joint is prismatic and the rest are revolute (Sec.2.1.2). The corresponding graphs for the last four joints are seen in Fig.3.22.

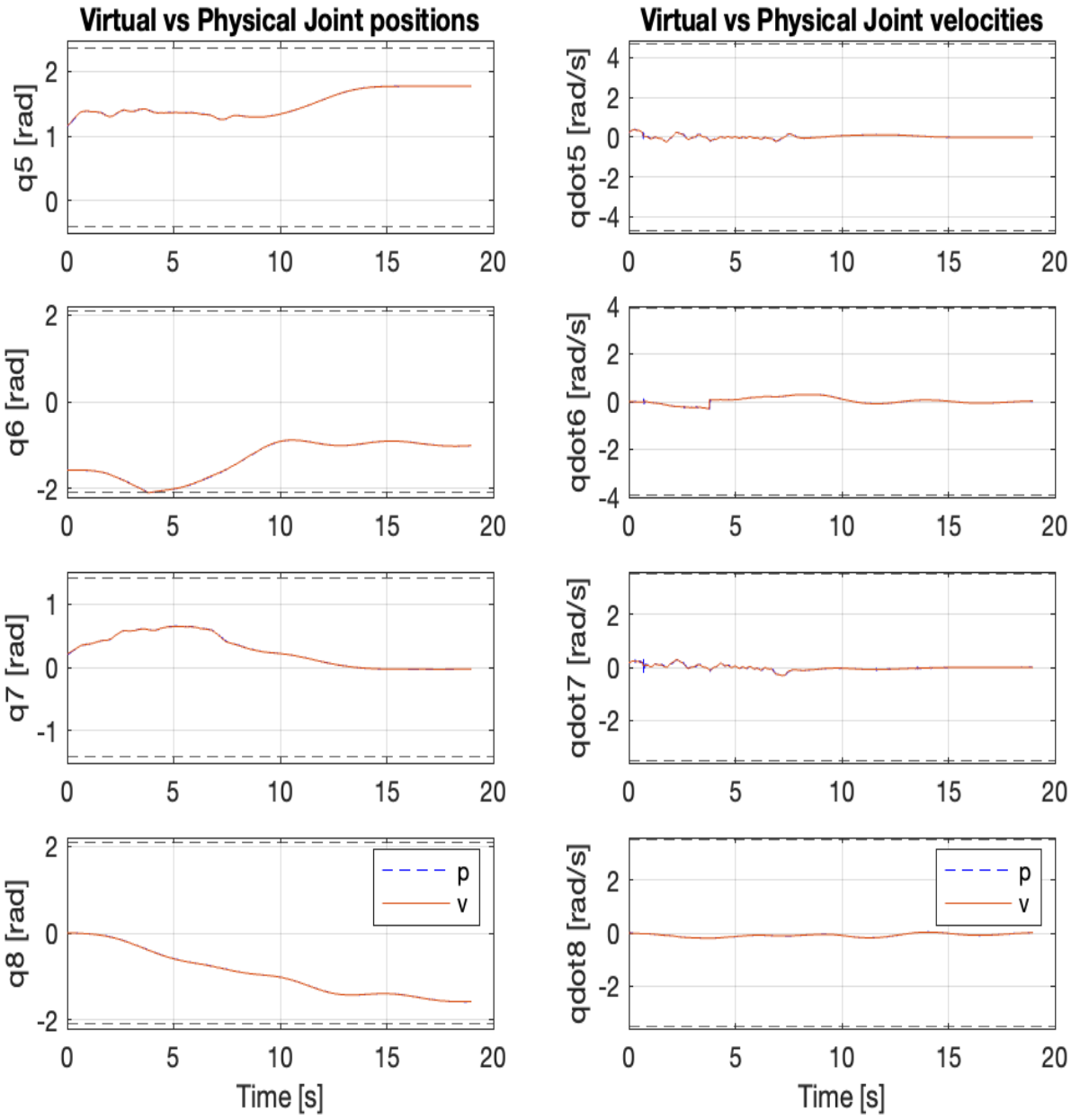


Figure B.26: The task specification is given by Table. 3.4. The graphs in the right column show virtual (red) and physical (dashed blue) joint positions for the last four joints; dashed lines denote upper and lower position limits. In the left column graphs, virtual and physical joint velocities; upper and lower dashed lines denote velocity limits. The last four joints q_5, q_6, q_7, q_8 are revolute (Sec.2.1.2). The corresponding graphs for the first four joints are seen in Fig.3.21.

C

Appendix C

Data recorded during KT with joint velocity limits set equal to the physical robot, see `https://youtu.be/t0Wgd-3bS7c`. See Figs. C.1, C.2 and C.3. Compare with Sec. 3.3.2.

In Fig.C.1 we observe poor tracking of the linear velocities. The reason for the poor tracking is found in Figs. C.2 and C.2 where it's seen that the joint velocities of the virtual robot are frequently at velocity limits, hence, the optimizer was likely not able to find a joint velocity command to follow the reference.

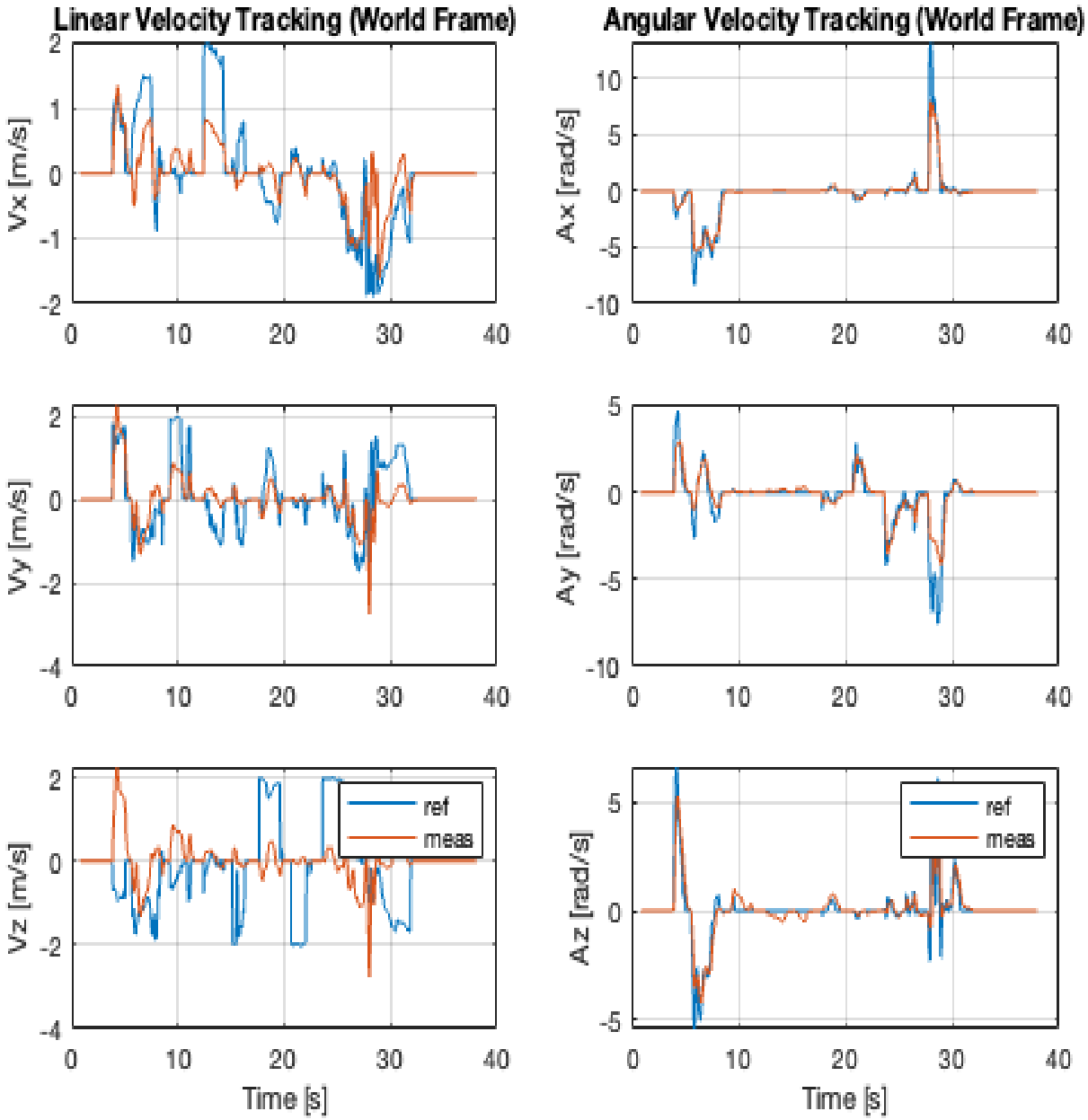


Figure C.1: Reference tracking during kinesthetic teaching with more conservative joint velocity limits than was used for Fig. 3.7. Graphs in the right column depict the tracking of linear velocity. The left column graphs depict angular velocity tracking.

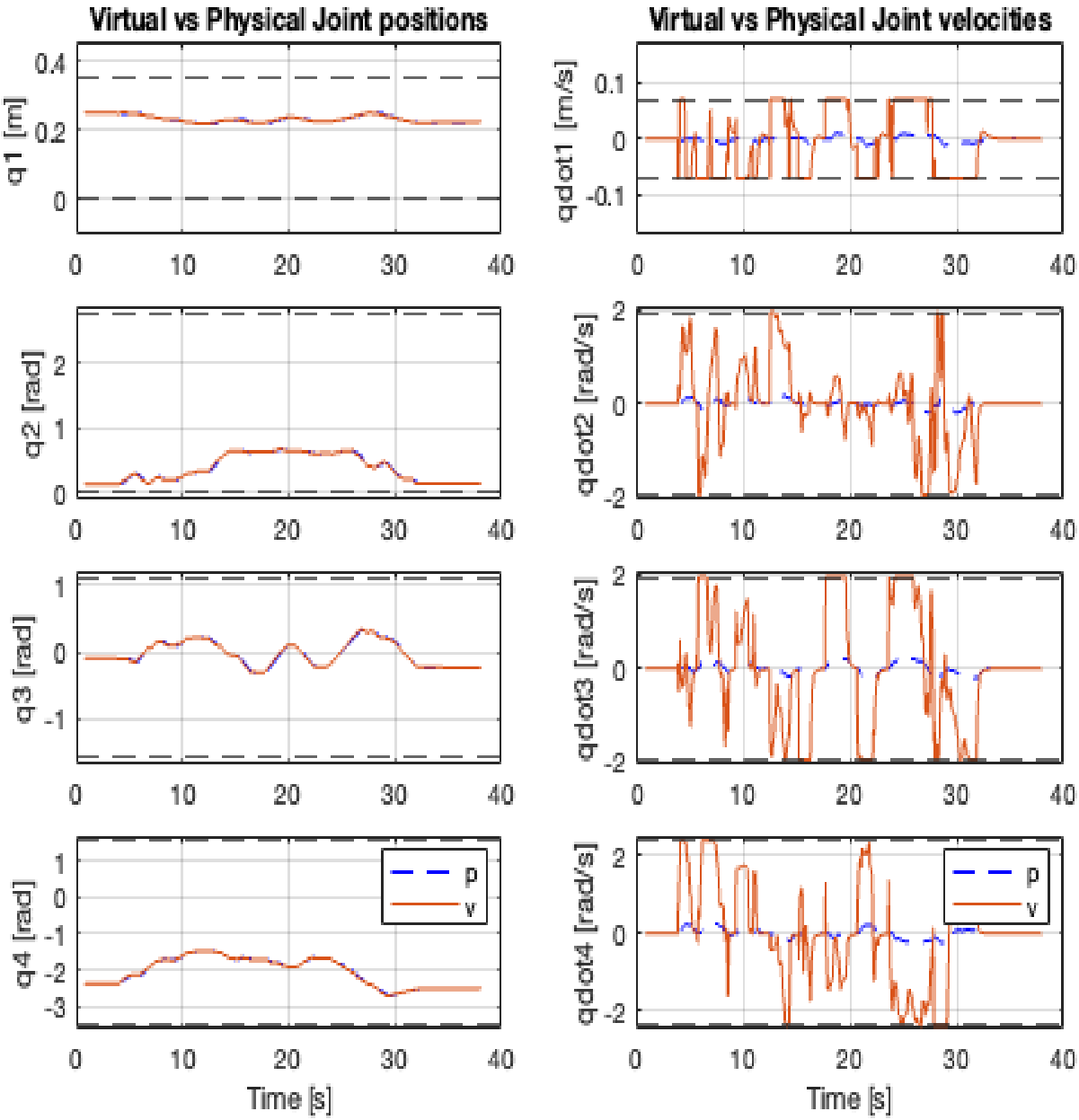


Figure C.2: In the graphs to the right, the first four virtual and physical joint positions are compared. The upper and lower black dashed lines depict upper and lower joint position limits. In the left column, virtual and physical joint velocities are compared. The upper and lower black dashed lines depict upper and lower joint velocity limits, note, that they're lower than in Fig. 3.8.

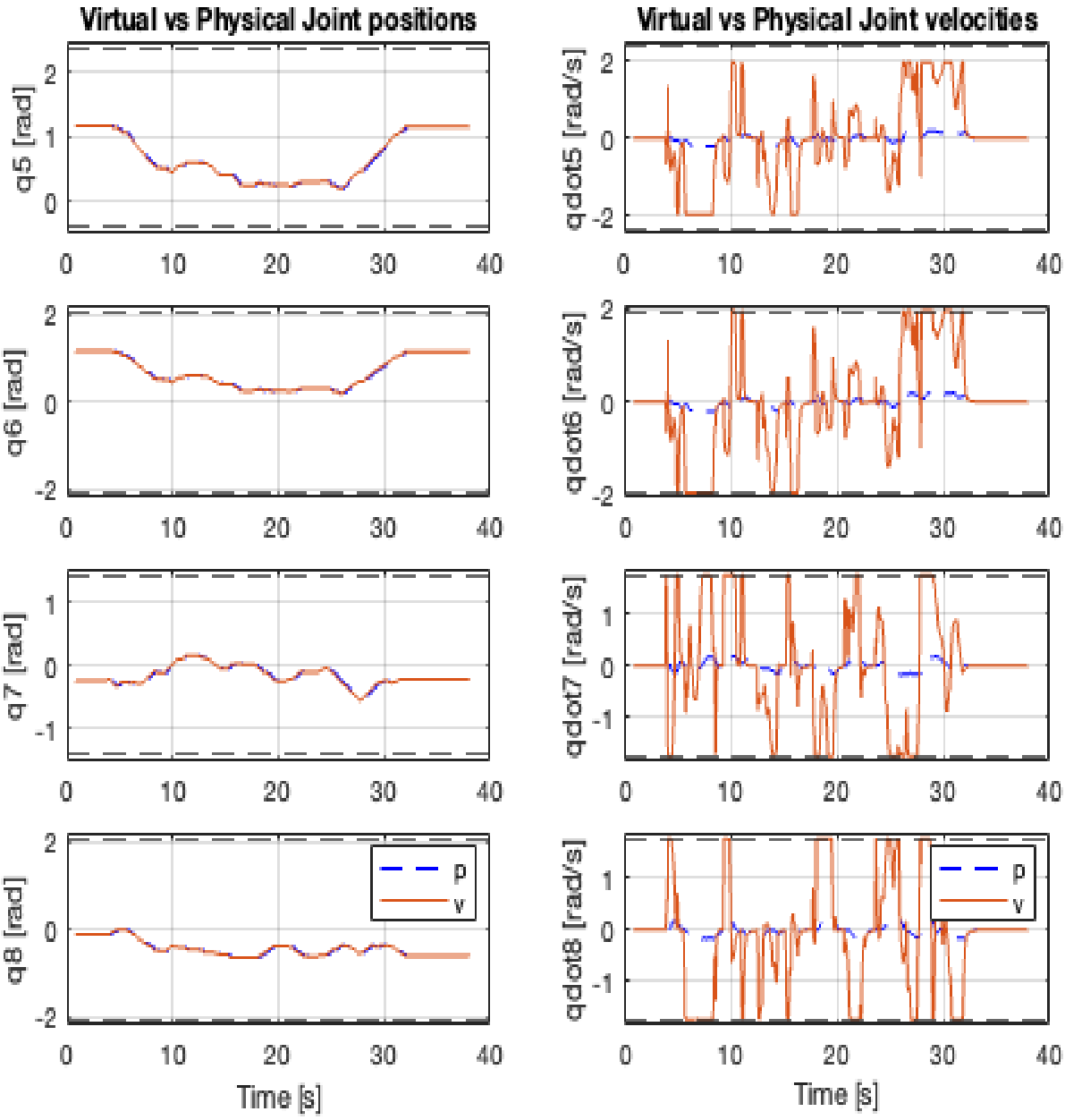


Figure C.3: In the graphs to the right, the last four virtual and physical joint positions are compared. The upper and lower black dashed lines depict the upper and lower joint position limits imposed on the virtual robot. In the left column, virtual and physical joint velocities are compared. The upper and lower black dashed lines depict upper and lower joint velocity limits, note, that they're lower than in Fig. 3.9.

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY