Lazy Robot Control by Relaxation of Motion and Force Constraints

Djordje Vukcevic

Publisher: Dean Prof. Dr. Wolfgang Heiden

Hochschule Bonn-Rhein-Sieg – University of Applied Sciences,
Department of Computer Science

Sankt Augustin, Germany

September 2020

Technical Report 03-2020

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

ISBN 978-3-96043-084-1

This work was supervised by
Paul G. Plöger
Herman Bruyninckx
Sven Schneider

Digital Object Identifier https://doi.org/10.18418/978-3-96043-084-1

Acknowledgements

I would like to express my sincere gratitude to Prof. Dr. Paul Plöger for not just valuable feedback and remarks, but also his continuous support during the work.

I would like to express my sincere gratitude to Prof. Dr. Herman Bruyninckx for valuable insights and ideas.

I would like to especially thank M.Sc. Sven Schneider for encouraging and motivating me throughout the project. He was not only a great advisor but also a great teacher and friend who helped me to understand and learn a lot about this field of research.

Finally and most importantly, I would like to thank my family and friends for their endless love and support in every work I do.

Abstract

Human and robot tasks in household environments include actions such as carrying an object, cleaning a surface, etc. These tasks are performed by means of dexterous manipulation, and for humans, they are straightforward to accomplish. Moreover, humans perform these actions with reasonable accuracy and precision but with much less energy and stress on the actuators (muscles) than the robots do. The high agility in controlling their forces and motions is actually due to "laziness", i.e. humans exploit the existing natural forces and constraints to execute the tasks.

The above-mentioned properties of the human lazy strategy motivate us to relax the problem of controlling robot motions and forces, and solve it with the help of the environment. Therefore, in this work, we developed a lazy control strategy, i.e. task specification models and control architectures that relax several aspects of robot control by exploiting prior knowledge about the task and environment. The developed control strategy is realized in four different robotics use cases. In this work, the Popov-Vereshchagin hybrid dynamics solver is used as one of the building blocks in the proposed control architectures. An extension of the solver's interface with the *artificial* Cartesian force and feed-forward joint torque task-drivers is proposed in this thesis.

To validate the proposed lazy control approach, an experimental evaluation was performed in a simulation environment and on a real robot platform.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

Abbreviation		Meaning
ABA	=	Articulated-Body Algorithm
ABAG	=	Adaptive Bias Adaptive Gain
ACHD	=	Acceleration-Constrained Hybrid Dynamics
ASMC	=	Adaptive Sliding Mode Control
COM	=	Center Of Mass
DMP	=	Dynamic Movement Primitives
DOF	=	Degrees Of Freedom
FD	=	Forward Dynamics
FSM	=	Finite-State Machine
HD	=	Hybrid Dynamics
HQP	=	Hierarchical Quadratic Problem
ID	=	Inverse Dynamics
iTaSC	=	instantaneous Task Specification and Control
KDL	=	Kinematics and Dynamics Library
MPC	=	Model Predictive Control
OCP	=	Optimal Control Problem
ODE	=	Ordinary Differential Equation
OSF	=	Operational Space Formulation
PID	=	Proportional-Integral-Derivative
PRM	=	Probabilistic Roadmap
RBDL	=	Rigid Body Dynamics Library
RNE	=	Recursive Newton-Euler
ROS	=	Robot Operating System
RRT	=	Rapidly-Exploring Random Tree
SMC	=	Sliding Mode Control
SoT	=	Stack of Tasks
TFF	=	Task Frame Formalism
URDF	=	Unified Robot Description Format
WBC	=	Whole-Body Control
WBOSC	=	Whole-Body Operational Space Control

List of Symbols

$\ddot{X}$	Cartesian acceleration vector - Spatial vector in Plücker coordinates
$\dot{X}$	Cartesian velocity vector - Spatial vector in Plücker coordinates
$\hat{\omega}, \theta$	Axis and angle of rotation in the exponential coordinate representation, repectively
$\mathcal{L}$	Acceleration energy contribution in the Popov-Vereshchagin solver, induced by the unit constraint forces
ν	Lagrange multiplier - Vector of magnitudes of constraint forces
σ	Tube tolerance (bound)
τ	Vector of force variables in joint space
$\times, \times^*$	Spatial cross-product operators, for motion and force respectively - Plücker coordinates
φ	Rotation angle about a single frame axis
A_N	$6 \times m$ matrix of 6×1 spatial unit-column vectors of Cartesian constraint forces imposed on segment N - Unit-column vectors defined in Plücker coordinates
b_N	6×1 vector of acceleration energy setpoint for segment N
$C(q, \dot{q})$	Bias force vector consisting of Coriolis, centrifugal and gravity forces in joint space
D	Combined inertia matrix of a segment inertia and its associated joint rotor inertia
d	Joint rotor inertia
f	Linear force acting on a body
F_c	Vector of contact forces - Spatial vector in Plücker coordinates
$H(q)$	Joint space inertia matrix
I	Spatial rigid-body inertia
I^{rot}	Rotational inertia of a rigid body
k	Path section index

m	Mass of a rigid body
n	Angular force (moment) acting on a body
P^A	Projection operator for articulated body inertias and forces
q, $\dot{q}$, $\ddot{q}$	Position, velocity and acceleration vectors in joint space respectively
R	3×3 rotation matrix
r	3×1 position vector
S	Matrix for defining subspace (freedom) of a motion vector
t_p	Time horizon in prediction calculations
U	Acceleration energy contribution in the Popov-Vereshchagin solver, induced by the spatial bias forces and feed-forward joint torques
Z	Acceleration energy induced in the system - Gauss function
$(\cdot)^T$	Transpose of a matrix
$(\cdot)^{-1}$	Inverse of a matrix
${}^{i+1}X_i$, ${}^{i+1}X_i^*$	Matrices for transformation of coordinates for motion and force vectors respectively, from coordinate $\{i\}$ to coordinate $\{i+1\}$
F^{ext}	External force acting on a rigid body - Spatial vector in Plücker coordinates
F_{bias}	Bias force acting on a rigid body (Coriolis, centrifugal and external forces) - Spatial vector in Plücker coordinates

1

Introduction

1.1 Motivation

In many different environments, human and robot tasks include actions such as grasping and carrying an object, cleaning a surface, etc. These tasks are performed by means of dexterous manipulation, and for humans, they are straightforward to accomplish. Moreover, the accuracy and precision of their motion and force execution are, at the specific time instance, automatically adapted based on the task requirements. Nevertheless, humans perform these tasks with reasonable accuracy and precision but with much less energy and damage to the actuators (muscles) than the robots do. The high agility in controlling these forces and motions is actually due to "laziness", i.e. humans exploit the existing natural forces and constraints to execute tasks with the least energy, time and planning required [7]. While performing the motions and producing forces, humans make mistakes and they are constantly performing goal-directed corrections [8] until they reach the goal [9]. This *lazy* strategy allows for high performance even without burdening the task control with hard and artificial constraints.

The justification and motivation for creating this type of control, in the context of robotics, comes from the fact that model parameters of a robot and its environment are not completely accurate. For example, inertial parameters of a robot, pose and size of objects in our environment always come with an error [10], [11]. Some parameters are even unknown and time-varying, e.g. friction parameters [12]. It is only possible to have an approximation of these models. Moreover, in the sensors used for providing feedback information, noise and uncertainties always exist [13], [14]. The above-mentioned properties of the human lazy control strategy, motivate us to relax the problem of controlling robot motions and forces and solve it with the help of the environment. In other words, relax force and motion constraints, and not look for optimal control solutions but rather for ones that are good enough.

1.2 Use Cases and Challenges

In order to understand how to relax the control of robot motions and forces, we can first consider several robotic use cases and their related challenges.

Pre-Grasp Motion: The first use case is represented by a task that requires the robot to perform a pre-grasp motion before grasping an object. A robot can complete this task in different ways, for example, *i*) in a traditional way, by first estimating the pose of the object and then performing a "blind" motion towards the goal state or, *ii*) in a more intelligent way, by performing the motion in a visual servoing manner, i.e. by constantly adjusting its motion based on the feedback received from a camera or any other sensor. The challenges in this use case are:

- Accurate guiding of the robot towards the goal state while satisfying the imposed constraints, i.e. bounds on the robot's motion. These constraints are usually derived from the accuracy and precision tolerances or even environmental limits such as, for example, the minimum distance to an obstacle, and so on.
- Reactivity of the system to the effects which cannot be estimated in a single time instance but require a longer time horizon to be visible [15]–[17]. More specifically, early and smooth robot reaction to future events, such as motion deviations from the imposed bounds, collisions with obstacles, reaching joint limits, etc.

Following a Pre-defined Path: The second use case considers the task of welding metal components. Here, the robot is requested to follow a user-defined Cartesian path. The challenge in this use case is the accurate tracking of predefined position *waypoints* along the path. In addition, the same challenges that are necessary to be addressed in the first use case hold in this scenario as well.

Cleaning a Surface: The third use case considers the task of cleaning a surface (a table for instance). This task involves explicit control of the robot's Cartesian forces, together with control of its motion variables. For that reason, the challenge in this use case is sufficiently accurate execution of desired forces, in directions in which contacts with the environment are maintained. In addition, the same challenges that are necessary to be addressed in the second use case hold in this scenario as well.

Transporting an Unfamiliar Package: In the fourth use case, a robot arm is transporting an unfamiliar package without knowing its mass and inertia. When the robot's end-effector grasps an object, inertia and mass of the controlled system change. Thus, the challenge in this use case is the adaptation to the aforementioned changes in model parameters, i.e. compensation for an unknown load on the robot's end-effector. Additionally, the same challenges that are necessary to be addressed in the first use case hold in this scenario as well.

Common challenges: Lastly, the common challenges for all the above-mentioned use cases are:

- Minimized energy consumption throughout the execution of the desired robot task. In other words, low electrical energy to power the system's actuators.
- Ability for resolving robot motions even in the case of incomplete task specifications [2], i.e. in the case when some of the DOFs are left unspecified in the task definition [18].

1.3 Contributions

The contributions of this master's thesis are:

- A lazy control strategy, i.e. task specification models and control architectures that relax several aspects of robot control.
- Application of the Adaptive Bias Adaptive Gain (ABAG) controller and Popov-Vereshchagin hybrid dynamics solver in different real-world use cases.
- Extension of the Popov-Vereshchagin solver's interface with the *artificial* Cartesian force and feed-forward joint torque drivers.
- An evaluation of the proposed lazy control strategy in a simulation environment and on a real robot platform.

1.4 Thesis Outline

The structure of this master's thesis is the following:

Chapter 2 presents the state of the art in the fields of robot dynamics, motion control and planning.

Chapter 3 describes the identified problems and the task of this thesis.

Chapter 4 presents a detailed review of the original Popov-Vereshchagin hybrid dynamics solver and the proposed solver's extension.

Chapter 5 presents a detailed review of the Adaptive Bias Adaptive Gain (ABAG) controller.

Chapter 6 presents the realization of the lazy control strategy in four different use cases.

Chapter 7 presents an experimental evaluation of the proposed control strategy in three different use cases.

Chapter 8 concludes the work of this thesis and outlines the future work directions.

2

State of the Art

2.1 Robot Dynamics

The interactions of forces and motions of rigid bodies are studied by the field of dynamics. Equations that describe these quantities are evaluated by dynamics algorithms, which perform calculations for the variables of interest. In robotics, two main types of the problem for which these calculations are performed involve the *forward* and *inverse* dynamics. Here, the *forward* dynamics (FD) deals with the computation of accelerations that result from applied joint and Cartesian forces. On the other side, the *inverse* dynamics (ID) deals with the computation of forces that are required, to be generated in the system's actuators, for a robot to move with desired joint or Cartesian accelerations. An additional problem for which dynamics algorithms are used include *hybrid* dynamics (HD) computations, that deal with the combined problem of inverse and forward dynamics [19]. More specifically, the goal is to compute both unknown forces and unknown accelerations in the system, in the case when for certain joints/segments forces are known (specified) and for other joints/segments accelerations are known (specified). An additional aspect in the field of dynamics, to be considered, is the resolution of constraints imposed on the rigid body motions. These constraints can be *artificial* (e.g. desired motion specified by the task definition) and *physical* (e.g. contact between two bodies) [19].

Different types of dynamics algorithms are used for finding solutions to the aforementioned problems:

- The *Recursive Newton-Euler algorithm* (RNE) used for solving the inverse dynamics problem, for specified joint space accelerations.
- The *Articulated-Body Algorithm* (ABA) used for solving the forward dynamics problem, for specified joint space forces.
- The *Articulated-Body Hybrid Dynamics Algorithm* used for solving the hybrid dynamics problem; it represents an adaptation/extension of the aforementioned ABA [19].
- The *Popov-Vereshchagin* algorithm used for solving the hybrid dynamics problem, by resolving Cartesian acceleration constraints imposed on robot's segments and additionally computing the resulting motion from the required joint forces that realize the above-mentioned constraints [2]–[5].

Commonly used implementations of the above-described dynamics solvers can be found in the following open source software solutions:

- Kinematics and Dynamics Library (KDL): used for constructing robot kinematic chains and computation of robot motions [20]. Beside the kinematics algorithms, the library also includes an implementation of two dynamics algorithms: *Recursive Newton-Euler* and *Popov-Vereshchagin* solvers. Moreover, this library can be used as a standalone software tool or even in connection with the Robot Operating System (ROS) [21] framework.

- Rigid Body Dynamics Library (RBDL) [22]: contains implementation of three dynamics solvers, *Recursive Newton Euler Algorithm*, *Composite Rigid Body Algorithm* and *Articulated Body Algorithm*. Additionally, the library includes algorithms for solving forward and inverse kinematics problems, as well as algorithms for contact handling problems.

- RobCoGen: a code generator software library developed to generate efficient code for computations of the robot's dynamics and kinematics [23]. The code is optimized for a specific robot model that is given as an input.

- Other common examples include: Pinocchio [24] and JRL-RBDyn [25] libraries.

The above-mentioned software solutions are used as part of both robot simulators and control pipelines.

2.2 Motion Control and Planning

In practice, when it comes to executing the computed motions (i.e. resolved motions via some of the above-described techniques), many problems exist due to uncertainties in the robot's system and environment. These uncertainties include an inaccurate model of the robot's dynamics, sensor noise, unknown model of the environment, etc. [14]. A research field that is addressing challenges in controlling and planning robot motions, aims to provide solutions for smooth, accurate, stable and reactive task execution [6], [10]–[14], [16], [26]–[30]. The best-known representatives are the following.

2.2.1 PID Control

For enabling feedback closure and adjusting of the control commands in runtime, instances of the PID controller family are frequently used [12], [26], [31]. The reason why it is so frequently applied, especially in industrial applications [14], is the fact that this control technique is simple to implement and interpret. This method is used in many aspects of the robot motion control; starting from the regulation of a single joint motion, up to controlling a motion execution of the complete robot systems. The controller is very often used in combination with a joint space inverse dynamic solver [13], [19], to enable performant joint space trajectory tracking [32]. However, it is mainly designed for controlling linear systems and when it comes

to applying this type of control on nonlinear systems, many nonlinearities, such as friction, saturation, hysteresis, etc. need to be ignored [33]. Additionally, the system's parameters are often inaccurate and some of them are even unknown. For the aforementioned reasons, when a nonlinear system leaves the operational range around which was linearized, the performance of this type of controller reduces and it may even become unstable [33]. Thus, to provide a better performance in the motion execution of the nonlinear and complex systems such as robots, we require *nonlinear* control techniques [13].

2.2.2 Sliding Mode Control

One of the most frequently used *nonlinear* control methods for controlling mechanical systems is the *Sliding Mode Control* (SMC) [10]. Mainly because of its robustness to system's uncertainties (incorrect and unknown model parameters) and external disturbances [10], [34]. Here, the robustness is enabled by a *switching function* that defines a change of the control law, i.e. a change of the feedback controller that is used in a particular time instant during the control operation [34], [35]. The switching function, together with its predefined control laws, is forcing a system to 1) reach a predefined *sliding surface* (reaching phase), and 2) continue "sliding" within boundaries of this manifold (sliding phase). Here, a sliding surface defines a state subspace in which the system's behaviour is normal. A decision on which particular controller should be used in the current time instant is based on the current error in the system's motion. Here, an error value defines how much the system deviates from its normal behaviour (sliding surface). In comparison to this control strategy, the PID controller differs in a way that it defines only a single type of control law and it is used regardless of the current system's error [15]. However, the downside of the SMC technique is that it produces a discontinuous control signal [34] which may induce chattering in the system's motion. Nevertheless, several approaches were proposed to remedy this problem, and the most successful ones are the low-pass filtering, higher-order control, and adaptive control [13]. These approaches, especially Adaptive Sliding Mode Control (ASMC), have shown high-performance results in controlling complex robot systems, such as robot manipulators and humanoid robots [10], [13].

2.2.3 Adaptive Bias Adaptive Gain Control

In [6], the authors have presented a novel technique for controlling the thrust of a multi-rotor aerial platform, called *Adaptive Bias Adaptive Gain* (ABAG) controller. However, this control method is suitable for providing a reliable task execution on various other types of mechanical systems, as well. The main feature of this algorithm is that it approximates dynamics of the controlled system based on a *trend* in the instantaneous motion error or, in other words, by looking in the *past-time* error horizon; even without the necessity for a model of the system [6]. The information about the approximated system's model is resembled by the controller's *bias* term, which is constantly updated together with the controller's *gain* term, by following a trend in the motion error. In the original publication [6], the waveforms of both bias and gain terms are represented by fixed functions that

authors refer to as *decision maps* and these functions are defined beforehand by a control designer. This enables better predictability and stability of the system's behaviour, compared to, for example, the PID controller, where its control signal depends exclusively on the error. Additionally, a control designer chooses maximum values for the bias and gain terms by limiting its respective decision maps. This feature enables the possibility of limiting the controller's output, such that a wind-up situation does not occur [15].

2.2.4 Impedance Control

Robot tasks that involve interaction with the environment, such as, for example grasping an object, polishing a surface, etc. require a controller that is capable of properly handling contact forces. More specifically, used control approach must ensure a *compliant* behaviour of a robot. This capability is important for enabling safe interaction, i.e. avoiding breakage of the robot and environmental parts that are involved in a contact [14]. The importance of this type of control increases when humans are involved in the interaction [29]. Today, the most frequently used type of control for meeting this requirement is the *Impedance Control* [29], [36]. Other types include admittance control, hybrid force/position control, etc. [14]. In the context of impedance control methodology, the controlled robot is described as a spring-damper-mass system, on which a desired dynamic behaviour is imposed, while moving freely or interacting with the environment [29].

The robot impedance can be enabled in two ways, passively and actively. The passive impedance control can be enabled by equipping the robot system with flexible parts (joints and/or links). In this case, the mechanical impedance of the robot can only be adjusted physically, by making hardware changes directly on the robot. On the other side, an active impedance of the robot is enabled via software solutions and this type of control can work even if a robot consists completely of rigid body parts. Nevertheless, when combined, passive and active impedance control can produce very performant robot behaviour [37].

Different impedance controllers can be classified in several ways and the most important classifications define Cartesian space and joint space impedance control. The (active) Cartesian space impedance control is the most often used technique. Here, the robot end-effector's impedance is being regulated, while the robot is following a pre-defined Cartesian space trajectory and being subject to the forces originating from a contact with the environment. In the case of joint space impedance control, compliance in robot joints is enabled while the robot is following a joint space trajectory [38]. However, this type of control can be derived for each joint separately, and this approach is particularly useful for tasks of avoiding joint limits in a smooth manner. Nevertheless, combined, these two types of impedance control approaches can enable the whole-body control of a robot, i.e. end-effector and null space control in a single approach [38]. The rationale for this combined approach is not only improved safety but also improved overall stability and precision, in the case when the robot consists of redundant degrees of freedom (DOFs) [29].

The most important feature of this type of control is that an accurate model of the environment is not required for limiting contact forces, i.e. ensuring desired

compliance of the robot. However, this controller also has its drawbacks. The procedure of tuning the impedance controller depends on many factors, such as task requirements, hardware capabilities and availability of feedback. Moreover, a control designer very often needs to make a trade-off between precision and stiffness of its robot [29]. For instance, while the robot is in a contract with the environment, the preferred dynamic behaviour is to have a low stiffness of the robot, i.e. a small contact force to ensure sufficient safety. However, while ensuring a low stiffness of the robot, relatively large motion errors are likely to occur [14].

2.2.5 Model Predictive Control

The major challenge in applying robots in real-world scenarios are dynamic environments [11]. To cope with all uncertainties and unpredicted disturbances, a robot must be reactive while performing its motions. For providing a solution to the aforementioned requirements, currently, in the literature, the most popular method is the *Model Predictive Control* (MPC) technique [39]. Various MPC approaches were applied for planning and controlling motions of many types of robots, including robot manipulators [40], legged systems [28], [41], unmanned aerial vehicles [11], etc. [16]. This model-based control methodology encompasses two aspects of robot motion in a single approach: continuously (re)planning trajectories and computing the control commands that enable accurate tracking of the derived motion plans [11].

An MPC approach is (online and repeatedly) resolving a constrained *Optimal Control Problem* (OCP) [39]. Here, an OCP is defined by a cost function that is subject to equality and inequality constraints. In robotics applications, the most frequent setting of the OCP includes a cost function that is defined by the weighted sum of three quantities: terminal, integral and action costs [28]. Here, the terminal value represents a cost of the robot's final state, the integral value defines a cumulative cost of the intermediate states that are supposed to be covered towards the final state, and finally, the action cost represents a distance between the previous and current vector of control commands, considered throughout the optimization. The reasoning for including the last part of the cost function is a desire to enable a smoother transition between two consecutive robot states [27]. On the other side, the constraints are usually defined based on a model of the contact forces, joint limits, obstacles in the environment, etc. [41].

The future robot states, i.e. effects from applying the considered control commands, are predicted by evaluating the robot's dynamics model. The optimization result, at each step of the control loop, represents a locally optimal trajectory that is defined by a set of optimal future states and a set of optimal control commands required for governing the robot towards these states [42]. However, the feedback control is enabled by applying only the first control command, from the aforementioned set, on the real robot system, in each iteration of the MPC control loop [28]. Very often, the time horizon used in predictions is finite and short, in order to enable the real-time capability of the computations. The algorithms considered for optimizing a cost function in the MPC techniques include methods such as single and multiple shooting, direct collocation, differential dynamic

programming, interior-point methods, etc. [39], [41]. Due to high computational expenses, which are common in the MPC approaches, optimization algorithms are very often time-constrained in searching for the optimal solution [16]. For that reason, the computed optimum is in most cases a suboptimal solution [41], [42]. Additional approaches for improving the efficiency of MPC techniques are based on the derivatives of dynamics [43] and they are mostly used for system linearization and estimation of the gradient [40], [42].

For performing the MPC computations online, i.e. enabling a real-time implementation of this control strategy, in many applications, powerful processing hardware components are required to be integrated on a real robot system [16]. These requirements come from the fact that throughout the optimization (in every iteration of the control loop and for each step in prediction horizon) the whole dynamics of the system needs to be evaluated [28], [41].

2.2.6 Other Approaches

In the literature, many other approaches for motion planning and control exist [14]. However, in this section, the ones that should also be mentioned are 1) the *Sampling-Based* algorithms [14] such as, for example, *Rapidly-Exploring Random Tree* (RRT) [44] and *Probabilistic Roadmap* (PRM) [45], and 2) a strategy called *Dynamic Movement Primitives* (DMP) [46].

Sampling-Based strategies are based on random sampling of robot configurations in a feasible subspace of motion, that is free of joint limits and singularities. The sampling is biased towards the goal state; all neighbouring states in the configuration space are connected if the transition between two states does not involve a contact with an obstacle. This procedure is repeated until the complete robot trajectory, from initial to the desired state, is generated. The aforementioned algorithms are usually computationally expensive because, in this case, it is required to cover a large motion subspace until a solution is found [14].

The DMP refers to a strategy for generating reactive robot motion plans. This motion planning technique is very often used in the combination with a PID controller, to ensure accurate execution of the planned motions [46]. In this learning-based approach, the idea is to use a *learning from demonstration* method such that the robot learns a set of basic (primitive) motion plans towards the predefined goal states. After the learning procedure is completed, for every new goal state that is given as input to the algorithm, this approach is generating new robot trajectories by generalizing to the already learned (stored) trajectories. Once the "offline" procedure of training is completed, performing the "online" generalization procedures is not computationally expensive. However, it is time-consuming to enable this type of motion planning on a real robot. This is due to requirements for performing, i.e repeating the training procedure for every larger modification of desired robot tasks [47].

These approaches are mostly used for generating motion plans. Nevertheless, some extensions consider aspects of motion control as well. The aforementioned techniques are frequently used because they are simple to implement. However, due to their stochastic properties, both of the aforementioned approaches share

the same drawback. Unpredictability in the results from trajectory computations exist and thus, while using these methods for performing tasks, unforeseen robot behaviours may occur. Additionally, they do not make use of any model-based knowledge to resolve the motion planning problem. These properties make the above techniques not well suited for robot tasks that are subject to real-world settings.

2.3 Whole-Body Task Driven Robot Dynamics and Control

Highly redundant robot mechanisms with multiple end-effectors, such as, e.g. humanoid robots, require specification of the control tasks that are based on, not only the motions of a single end-effector but also, the motions of other parts of the robot, such as, legs, head, torso, etc. [47]. Several different approaches were considered for both, resolving and controlling, in a single framework, the imposed constraints on the whole-body *instantaneous* motions [48]–[50]. The most important representatives are the following.

2.3.1 The Operational Space Formulation (OSF)

To model and enable task-oriented whole-body control of robots, the *Operational Space Formulation* (OSF) approach was introduced in [48] and later on extended to Whole-Body Operational Space Control (WBOSC) framework [47], [51]–[54]. The framework establishes the prioritized (instantaneous motion) control hierarchy among three different control categories: 1) joint-limits and self-collisions, operational tasks (such as manipulation and locomotion) and posture tasks (e.g. maintaining balance) [53]; 2) avoiding obstacles; 3) environmental constraints handling tasks (e.g. contacts). However, this control technique can only account for *equality* type of constraints [52]. Nevertheless, the methodology formulates tasks dynamics in such way that it prevents violation (conflicting) of the higher priority tasks by the lower priority tasks and additionally, enables runtime monitoring of the robot's behaviour feasibilities [47], [52], [53]. For example, if unexpected obstacle occurs in the preplanned trajectory of the robot, the controller will detect a task in-feasibility before the collision occurs, and based on user input or a predefined control policy, the task can be modified to avoid the obstacle (without violating the higher-level constraints) or stop the motion. This characteristic of the prioritized hierarchy enables for the controller to detect an in-feasibility of a certain task by monitoring if the *Jacobian* of the respective task has become singular or not, in the computations of the instantaneous motions [52].

2.3.2 *Stack of Tasks* (SoT)

This instantaneous whole-body task-based control scheme was introduced in [49], [55] and extended in [56]–[60]. The framework is used for hierarchical (instantaneous motion) control of redundant manipulators and humanoid robots, where the motion task is specified with both *equality* and *inequality* constraints [61]. To enable a hierarchical *stack of tasks*, authors have defined a methodology where the lower priority tasks are recursively projected in the remaining motion null-space

of higher priority tasks. Here, the task specification is defined as a hierarchical quadratic problem (HQP), with the multiple quadratic programs in priority order sequence [57]. A typical quadratic program consists of a cost function that is being minimized and it is subject to artificial and/or physical constraints [59]. To solve recursively each minimization problem they use a domain-independent HQP active search algorithm [60], [62]. Additionally, for resolving the motions authors introduced slack variables in the constraint equations to transform from inequality to equality form of constraints.

2.3.3 *iTaSC*

This instantaneous motion control framework was introduced in [50], [63] and the name stands for "instantaneous Task Specification and Control". In order to control robot systems with multiple sensors, the authors have defined an approach where the task is represented by a relative motion with possible dynamic interactions among objects that are part of robot system or the environment. With this framework, a task programmer describes the task by specifying constraints on forces and relative motions between objects. Examples for such variables are: pose of the robot's end-effector with respect to a laser scanner, or the orientation of an object with respect to a camera mounted on the robot. For expressing tasks/constraints, the authors have introduced two types of coordinates in their framework, feature and uncertainty coordinates, both with respect to object and feature frames. Here, the feature coordinates are used to represent relative motions or interactions between features on specific objects. On the other side, uncertainty coordinates are used for representing geometric uncertainties, which can be involved due to disturbances in the environment, calibration errors in the robot arm, etc. The uncertainties included in this approach, represent the major difference in comparisons with other approaches. By using the aforementioned coordinates, authors have reduced errors in the task execution. In the initial *iTaSC* approach, inequality constraints were not included in computations of robot motions, but later on, researches have reformulated the approach as a convex optimization problem [64], [65], to account for this type of constraints, as well.

3

Problem Formulation and Task Description

In order to enable correct execution of robot tasks, such as the ones presented in section 1.2, the current state of the art approaches such as MPC [11], [28], [41] or, a group of sampling-based motion planning algorithms [45], [46] are mainly concentrated on planning robot trajectories and applying a controller for their accurate tracking. Explicit constraints are imposed on various variables in the control of robot motions and forces, such as constraints on the state variables, force increments, a maximum value of the system's output, etc. However, the resulting robot motions are not characterized by high agility since the motion plans are over-constraining the system. Additionally, the constraints are resolved by computing a solution to an optimization problem [14], [28], [44], [46], therefore, the computations of motion plans usually become expensive. Moreover, these strategies are resulting in the setpoint tracking control for each step along the path, which is not required for most applications.

When it comes to addressing the challenge of reactivity to the effects which cannot be estimated in a single time instance, but require a longer time horizon to be visible [15]–[17], the above-mentioned MPC strategies [11], [28], [41] perform expensive predictions of future robot states. The reason why these techniques lead to high computational expenses is the fact that whole robot dynamics is evaluated for each step considered in the future time horizon and this procedure is repeated in every iteration of the control loop.

For solving the control problem presented in the third use case (section 1.2), i.e. enabling explicit control of robot's Cartesian forces together with the control of its motion variables, current control strategies, such as, for example, [66], [67], are designed to maintain predefined force *setpoints*. However, in this way, the approaches are over-constraining the control of robots, which is not necessary for most applications.

For addressing the problem presented in the fourth use case, i.e. the problem of compensating for an unknown load on the robot's end-effector, current approaches, such as [68], [69] are aiming for the exact estimation of load parameters by introducing additional computations in their control loops. However, these approaches are not exploiting the already existing data - processed in computations of control signals and dynamics - to extract information about the load exerted on the robot.

Finally, the aim of this project is to develop task specification models and control architectures that relax the control of robot's motions and forces, i.e. enable a lazy robot control, by exploiting prior knowledge about the tasks and environment. Furthermore, the task of this work includes an evaluation of the proposed approach in order to show its feasibility.

4

Operational Space Hybrid Dynamics

In the related work sections 2.1 and 2.3, the importance of robot dynamics properties and calculations has been presented. To control a robot in the operational space, it is necessary to compute required joint space commands which will achieve the motion and force constraints imposed on the robot. However, in this work, the Popov-Vereshchagin acceleration-constrained hybrid dynamics (ACHD) solver is chosen as a building block for resolving robot dynamics, given the operational-space control commands. Additionally, here, the ACHD algorithm's interface is extended to also account for specifications of *artificial* Cartesian force and *artificial* feed-forward joint torque task drivers. The original solver and its extension will be reviewed in more details, in the following.

4.1 Description

In 1970', researchers [1], [2], [70], [71] have developed a hybrid dynamics algorithm for computing the required joint control commands based on the defined Cartesian acceleration constraints, feed-forward joint torques and the external Cartesian forces from the environment. The solver is derived from a well-known principle of mechanics - *Gauss' principle of least constraint* [72] and provides a solution to the hybrid dynamics problem with linear-time, $O(n)$ complexity [3].

In general, the Gauss' principle states that the true motion (acceleration) of a system/body is defined by the minimum of a quadratic function that is subject to linear geometric motion constraints [72], [73]. The result of this Gauss function represents the *acceleration energy* of a body, which is defined by the product of its mass and the squared distance between its allowed (constrained) acceleration and its free (unconstrained) acceleration [74]. In the case of originally derived Popov-Vereshchagin algorithm [1], geometric motion constraints are Cartesian acceleration constraints imposed on the robot's end-effector. This domain-specific solver minimizes the acceleration energy by performing computational (outward and inward) sweeps along robot kinematic chain [3]. Furthermore, by computing the minimum of Gauss function, the Popov-Vereshchagin solver resolves the kinematic redundancy of the robot, when a partial motion (task) specification is provided [2].

The Gauss function that is minimized by the solver is formulated as [3], [4]:

$$\min_{\ddot{q}} \quad Z(\ddot{q}) = \sum_{i=0}^{N} \{\frac{1}{2} \ddot{X}_i{}^T I_i \ddot{X}_i + F_{bias,i}^T \ddot{X}_i\} + \sum_{j=1}^{N} \{\frac{1}{2} d_j \ddot{q}_j{}^2 - \tau_{ff,j} \ddot{q}_j\} \tag{4.1}$$

$$\begin{aligned} \textit{subject to}: \qquad & \ddot{X}_{i+1} = {}^{i+1}X_i \ddot{X}_i + \ddot{q}_{i+1} S_{i+1} + \ddot{X}_{bias,i+1} \\ & A_N^T \ \ddot{X}_N \ = \ b_N \end{aligned}$$

where term Z represents acceleration energy of the complete mechanical system [72], and the unit of this quantity is *acceleration times force* $[\frac{Nm}{s^2}]$. Here, $\ddot{X}$ denotes a spatial 6×1 vector of acceleration for particular robot segment (link). This vector contains both linear and angular components defined in Plücker coordinates [19]. For a more detailed description of spatial vector representation in Plücker coordinates we refer the reader to [4], [19]. Here, a kinematic chain is structured in such a way that the indexing of segments starts from 0 and ends with N, while the joints are numbered from 1 to N. Additionally, the kinematic chain structure defines that the joint i connects two adjacent segments, proximal $i-1$ and distal i [2], [4], [5]. Matrix I represents spatial rigid-body inertia and F_{bias} defines a spatial bias force vector for a particular segment. Joint accelerations are defined via term $\ddot{q}$. Symbol d denotes joint rotor inertia, while variable τ_{ff} represents feed-forward joint torque. Here, the matrix ${}^{i+1}X_i$ performs the transformation of the coordinates for the previously defined motion vector $\ddot{X}$, from coordinate $\{i\}$ to coordinate $\{i+1\}$ [19]. The term S represents the motion subspace matrix, that defines the motion freedom of a particular joint, based on its physical constraints. Term A_N is a $6 \times m$ matrix consisting of spatial 6×1 unit-column vectors, such that each column defines the *direction* of a single constraint force imposed on segment N (i.e. end-effector) [3]. Each unit-column vector contains both linear and angular components defined in Plücker coordinates [19]. Finally, b_N denotes the $m \times 1$ vector of desired acceleration energy for segment N.

The linear constraints presented in equation 4.1, together with the quadratic objective function, define a constrained optimization problem that requires two optimization methods for solving it, i.e. for deriving the Popov-Vereshchagin algorithm [1]. Using those two approaches, the constrained optimization problem is transformed into an unconstrained optimization problem and solved analytically. In the first step of algorithm derivation, the original formulation (equation 4.1) of the optimization problem was translated to the following form [2], [3]:

$$\min_{\ddot{q},\ \nu} \quad Z(\ddot{q}) + \nu^T (A_N^T \ddot{X}_N - b_N), \tag{4.2}$$

using the method of Lagrange multipliers [75]. Namely, the Cartesian acceleration constraints defined for the end-effector segment ($A_N^T \ \ddot{X}_N \ = \ b_N$) are integrated into the Gauss function. Here, the unknown term ν represents a variable called *Lagrange multiplier*.

The formulation of this optimization problem resembles a linear-quadratic problem (LQP). The approach of Vereshchagin and Popov used for deriving the

hybrid dynamics algorithm considers solving this LQP via dynamic programming technique [76]. Namely, based on Bellman's principle of optimality [76], the optimal solution is found recursively by propagating the objective function in a step-wise manner. More specifically, this procedure is performed by iterating through the kinematic chain and in each step considering only the dynamics relation between two consecutive segments of the robot.

A necessary condition that enables this type of closed-form algorithm (an analytical solution to the above-described optimization problem) defines that the robot's kinematic chain does not consist of closed loops, i.e. the robot's kinematic chain must be constructed in a *serial* or *tree* structure. However, it is always possible to cut these loops and introduce explicit constrains.

Representation

For evaluating robot dynamics, i.e. resolving its constrained motion, the Popov-Vereshchagin solver is performing three computational sweeps (recursions), along the kinematic chain [3], [4]. More specifically, two sweeps in *outward* and one sweep in *inward* direction. In the case of robot dynamics algorithms, the outward sweep refers to a recursion that is covering a kinematic chain from proximal to distal segments, while the inward sweep is covering a kinematic chain from distal to proximal segments [5]. In more detail:

1. The first, namely *outward* sweep is governing the computations of motion transformations X (functions of joint angles q), Cartesian velocities $\dot{X}$ (functions of joint angles q and rates $\dot{q}$), *bias* accelerations $\ddot{X}_{bias}$ and *rigid body bias*[1] forces F_{bias}, for each robot segment. Here, the existence of *bias* accelerations is influenced only by *gyroscopic* effects [3]. On the other side, the *bias force* vector is defined by Coriolis, centrifugal and external forces imposed on each segment, without accounting for gravitational forces.

2. The second, namely *inward* sweep is governing the computations of *articulated body*[1] quantities, namely inertias I^A and bias forces F^A_{bias}. Additionally, the computations are also covering *acceleration energy* contributions that are induced by i) bias forces and feed-forward joint torques, and ii) unit constraint forces. These two contributions are denoted by U and $\mathcal{L}$ terms, respectively. Here, the articulated body inertia and articulated body bias force of each segment are computed by summing (composing) its rigid body values with the *apparent*[1] inertias I^a and *apparent* bias forces F^a_{bias} of its children segments, along the kinematic chain, respectively [19].

3. Additionally, after completing the recursion in the second sweep and before starting the recursion in the last sweep, the solver is computing magnitudes of constraint forces, i.e. ν. More specifically, this operation is performed when the algorithm reaches link $\{0\}$, namely the base segment.

4. The last, also *outward* sweep is governing the computations of resulting joint accelerations $\ddot{q}$ and necessary joint torques τ for achieving those accelerations. Additionally, in this sweep, the computations of resulting spatial accelerations $\ddot{X}$, for each segment, are performed. Moreover, the results of these computations represent *final outputs* of the Popov-Vereshchagin solver.

In **Algorithm** 1, the original formulation of the Popov-Vereshchagin solver is presented. Here, the matrix ${}^{d_i}_{i}X$ represents transformation between proximal and distal frames of segment $\{i\}$. On the other side, the matrix ${}^{i+1}_{d_i}X$ represents transformation between distal frame of link $\{i\}$ and proximal frame of link $\{i+1\}$ (see figure 4.1). Here, for a pose of a particular segment, we refer to pose of its proximal frame. In this algorithm, the term X^* represents a matrix that performs the transformation of coordinates for spatial force vectors. Symbols $\times$ and $\times^*$

[1]For more detailed explanation on articulated, apparent and rigid body quantities, reader can refer to *Rigid body dynamics algorithms* book, by Roy Featherstone, 2008. [19]

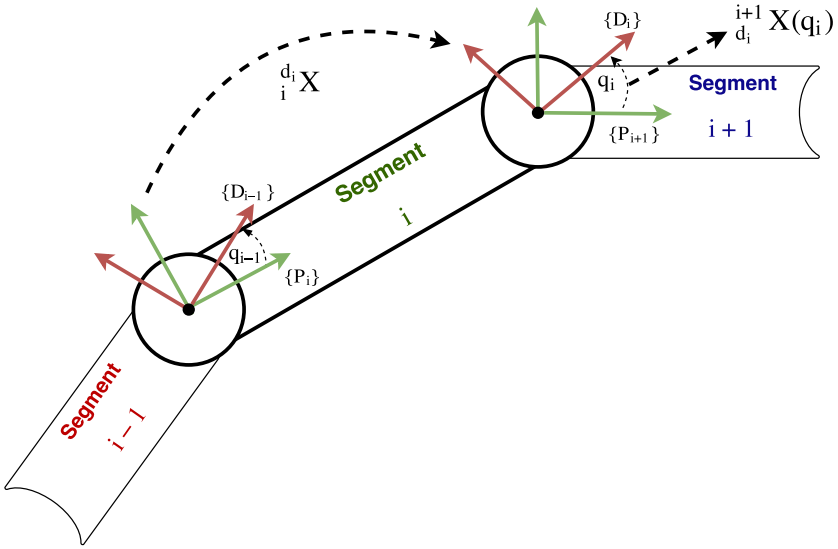

Figure 4.1: Visualization of segment frame assignments and transformations between them. Source [4].

represent spatial cross-product operators for motions and forces, respectively. Symbol D denotes the total inertia on the joint axis, which also covers the previously mentioned joint rotor inertia d [5]. Finally, term P represents a projection operator for articulated body inertias and forces [3], [5]. In this formulation of the solver, the gravity effects are taken into account by setting the acceleration of the base link ($\ddot{X}_0$) equal to gravitational acceleration [3].

Algorithm 1: Original Popov-Vereshchagin ACHD Solver [1]–[5]

Input : **Robot model**, $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{X}}_{\mathbf{0}}$, $\tau_{\mathbf{ff}}$, $\mathbf{F^{ext}}$, $\mathbf{A_N}$, $\mathbf{b_N}$

Output: $\tau_{\mathbf{ctrl}}$, $\tau_{\mathbf{total}}$, $\ddot{\mathbf{q}}$, $\ddot{\mathbf{X}}$

begin

  // Outward sweep of position, velocity and bias components

  for i = 0 to N − 1 do

    ${}^{i+1}_{i}X = ({}^{d_i}_{i}X \;\; {}^{i+1}_{d_i}X(q_i))$;

    $\dot{X}_{i+1} = {}^{i+1}X_i\dot{X}_i + S_{i+1}\dot{q}_{i+1}$;

    $\ddot{X}_{bias,i+1} = \dot{X}_{i+1} \times S_{i+1}\dot{q}_{i+1}$;

    $F_{bias,i+1} = \dot{X}_{i+1} \times^* I_{i+1}\dot{X}_{i+1} - {}^{i+1}X_0^* \; F^{ext}_{0,i+1}$;

    $F^A_{bias,i+1} = F_{bias,i+1}$;

    $I^A_{i+1} = I_{i+1}$;

  end

  // Inward sweep of inertia, force and acceleration energy contributions

  for i = N − 1 to 0 do

    $D_{i+1} = d_{i+1} + S^T_{i+1}I^A_{i+1}S_{i+1}$;

    $P^A_{i+1} = 1 - I^A_{i+1}S_{i+1}D^{-1}_{i+1}S^T_{i+1}$;

    $I^a_{i+1} = P^A_{i+1}I^A_{i+1}$;

    $I^A_i = I^A_i + \; {}^iX^T_{i+1}I^a_{i+1}\,{}^iX_{i+1}$;

    $F^a_{bias,i+1} = P^A_{i+1}F^A_{bias,i+1} + I^A_{i+1}S_{i+1}D^{-1}_{i+1}\tau_{ff,i+1} + I^a_{i+1}\ddot{X}_{bias,i+1}$;

    $F^A_{bias,i} = F^A_{bias,i} + \; {}^iX^*_{i+1}F^a_{bias,i+1}$;

    $A_i = {}^iX^T_{i+1}P^A_{i+1}A_{i+1}$;

    $U_i = U_{i+1} + A^T_{i+1}\{\ddot{X}_{bias,i+1} + S_iD^{-1}_i \cdot [\tau_{ff,i+1} - S^T_i(F^A_{bias,i+1} + I^a_{i+1}\ddot{X}_{bias,i+1})]\}; \quad U_N = A^T_N\ddot{X}_0$;

    $\mathcal{L}_i = \mathcal{L}_{i+1} - A^T_{i+1}S_{i+1}D^{-1}_{i+1}S^T_{i+1}A_{i+1}; \quad \mathcal{L}_N = 0$;

  end

  // Balance of acceleration energy at the base ({0} link)

  $\nu = \mathcal{L}^{-1}_0(b_N - A^T_0\ddot{X}_0 - U_0)$;

  // Outward sweep of resulting torque and acceleration

  for i = 0 to N − 1 do

    $\tau_{ctrl,i+1} = -S^T_{i+1}A_{i+1}\nu$;

    $\tau_{total,i+1} = \tau_{ff,i+1} + \tau_{ctrl,i+1} - S^T_{i+1}\{F^A_{bias,i+1} + I^A_{i+1}({}^{i+1}X_i\ddot{X}_i + \ddot{X}_{bias,i+1})\}$;

    $\ddot{q}_{i+1} = D^{-1}_{i+1}\tau_{total,i+1}$;

    $\ddot{X}_{i+1} = {}^{i+1}X_i\ddot{X}_i + \ddot{q}_{i+1}S_{i+1} + \ddot{X}_{bias,i+1}$;

  end

end

For more detailed description of the algorithm and its representation, the reader can refer to [3]–[5].

4.2 Interfaces

4.2.1 Input

For computing solutions to the constrained hybrid dynamics problem, the original Popov-Vereshchagin solver [2] takes into account the following inputs:

- A *robot model* that is defined by: kinematic parameters of the robot chain, mass and rigid body inertia of each segment, and inertia of each joint rotor.
- *Gravitational acceleration* of the robot base segment: $\ddot{X}_0$.
- *Joint angles* at the current time instant: q.
- *Joint velocities* at the current time instant: $\dot{q}$.
- *Motion drivers*:
  - *Acceleration constraints* imposed on the end-effector: $A_N^T \ddot{X}_N = b_N$.
  - *External forces* acting on each segment: F_{ext}.
  - *Feed-forward joint torques*: τ_{ff}.

The important feature of the Popov-Vereshchagin hybrid dynamics solver is its ability to take a task specification that is defined by three different motion drivers [5]. The following outlines the above-listed task interfaces in more detail.

Cartesian Acceleration Constraints: $\mathbf{A_N^T \ddot{X}_N = b_N}$ This first type of motion driver can be used for specifying *physical* constraints such as contacts with environment [3], or *artificial* constraints defined by the operational space task definition for the end-effector. To use this interface, a user should define i) the active constraint directions via A_N, a $6 \times m$ matrix of spatial unit constraint forces, and ii) acceleration energy setpoints via b_N, a $m \times 1$ vector. Here, the number of constraints m, or in another words number of spatial unit constraint forces is not required to always be equal to 6, which means that a human programmer can leave some of the degrees of freedom unspecified [2] for this motion driver, and still produce valid control commands [4], [5]. For example, if we want to constrain the motion of the end-effector in only one direction, namely linear x-direction, we can define the constraint as [3]:

$$A_N = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \end{bmatrix}, \qquad b_N = \begin{bmatrix} 0 \end{bmatrix} \tag{4.3}$$

Note that here, the first three rows of matrix A_N represent linear elements and the last three rows represent angular elements, of the spatial unit force defined in Plücker coordinates [19]. By giving zero value to acceleration energy setpoint, we are defining that the end-effector is not allowed to have linear acceleration in

x direction. Or in other words, we are restricting the robot from producing any acceleration energy in the specified direction.

Another example includes the specification of constraints in 5 DOFs. We can constrain the motion of robot's gripper, such that it is only allowed to move in the linear z-direction, without performing angular motions:

$$A_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad b_N = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

For both above-described task examples, the Popov-Vereshchagin solver will compute valid control torques, even though some of the Cartesian DOFs are left unspecified. More specifically: "Underconstrained motion specifications are naturally resolved using Gauss' principle of least constraint" [5]. For instance, in the second example, natural resolution of the robot motion would define that the end-effector "falls" due to effects of gravity, with the assumption that gravity forces are acting along z-direction.

The last example involves the specification of desired motion (in this case, non-zero accelerations) in all 6 $DOFs$:

$$A_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad b_N = A_N^T \ \ddot{X}_N \tag{4.5}$$

The reader should note that we are can directly assign values (magnitudes) of spatial acceleration 6×1 vector $\ddot{X}_N$ to the 6×1 vector of acceleration energy [3]. Even though physical dimensions (units) of these two vectors are not the same, the property of matrix A_N (it contains *unit* vectors), permits that we can assign values of desired accelerations to acceleration energy setpoints, in respective directions. Namely, each column of matrix A_N has the value of 1 in the respective direction in which constraint force works, thus it follows that the value of acceleration energy setpoint is the same as the value of Cartesian acceleration, in the respective direction.

External Forces: $\mathbf{F_{ext}}$ This type of driver can be used for specifying *physical* Cartesian forces acting on each robot segment. Examples for a *physical* force on a segment can be: *i*) a known weight at the robot's gripper, for instance, a grasped cup or *ii*) a force from a human pushing the robot [5].

Feed-Forward Joint Torques: τ_{ff} This type of motion driver can be used for specifying *physical* joint torques, such as e.g., including static friction torques in computations of motion [4], as it will be described in section 4.3.1.

4.2.2 Output

This recursive dynamics solver is computing several quantities that represent solutions to both, inverse and forward dynamics problems, or in other words solutions to the constrained hybrid dynamics problem. More specifically, the output interface of the original Popov-Vereshchagin algorithm consists of of [3]–[5]:

- Magnitudes of constraint forces that act on the end-effector, denoted by the previously-mentioned Lagrange multiplier: ν.
- Joint space control commands, i.e. joint torques required for achieving the desired (task-defined) behaviour of the robot: τ_{ctrl}.
- Argument that defines the solution to the originally formulated optimization problem in equation 4.1. In other words, a set of joint accelerations, resulting from the aforementioned control torques and all natural forces acting on the system: $\ddot{q}(\tau_{total})$.
- The resulting and complete spatial accelerations of each segment in the kinematic chain: $\ddot{X}(\ddot{q})$.

From line 28 in **Algorithm** 1, we can infer four different joint torque contributions which (combined) represent the *total* torque τ_{total}, *responsible* for producing the computed (resolved) motion of the system. Namely, we can see that the *total* torque consists of *i*) the physical feed-forward torque that is given as an input to the solver, *ii*) the control torque τ_{ctrl} defined in line 27, *iii*) gravity torque produced by gravitational forces acting on robot segments and *iv*) bias torque produced by all bias forces acting on the system. In the case of the original ACHD algorithm, the control torque consists of, purely, the constraint torque that is produced by the constraint forces acting on the end-effector.

The reader should note that this *control* torque is the (*necessary*) control command that a user is supposed to send to robot's actuators, to achieve the motion that is computed (resolved) by the Popov-Vereshchagin solver. Nevertheless, the reason why a user is not supposed to use the *total* torque values as the control commands for robot's actuators, is the fact that the torque contributions that represent the difference between τ_{total} and τ_{ctrl} already exist (act) on robot joints. More specifically, these *residual* contributions are produced on the joints by the already existing natural forces that act on the system.

The Popov-Vereshchagin hybrid dynamics solver enables a user to achieve many types of operational space tasks. In other words, various controllers can be implemented *around* the aforementioned interfaces of the algorithm. Examples can be controllers for hybrid force/position control [77], impedance control [29], [30], etc [14].

4.3 Extensions

4.3.1 Existing Extensions

To account for multiple motion constraints imposed on a *tree* robot structure, and not only on an end-effector segment but also on more proximal segments, the original solver (**Algorithm** 1) has been extended by Shakhimardanov in [3]. Moreover, an additional extension to the original algorithm was outlined in [2] and derived in [4]. This extension provides a solution for including dissipative forces in dynamics computations, i.e. reaction forces from joint static friction and joint position limits [4].

The original Popov-Vereshchagin algorithm, published in [2], has been implemented in the already mentioned open-source *Kinematics and Dynamics Library* (KDL) [20], by Ruben Smits, Herman Bruyninckx and Azamat Shakhimardanov. The complete library is created using the C++ programming language. However, the above-mentioned extensions to the original solver are not implemented in the current library version and they will not be used in this work.

4.3.2 Extension Developed in This Work

As previously described in section 4.2, the original Popov-Vereshchagin solver exposes an *external force* interface that can be used for specifying *physical* Cartesian forces acting on each robot segment, and a *feed-forward joint torque* interface that can be used for specifying *physical* joint torques acting around each joint axis. However, this algorithm can be extended to also account for specifications of *artificial* Cartesian forces and *artificial* feed-forward joint torques acting on a robot system. Examples for *artificial* Cartesian force specifications, that motivate the aforementioned solver's extension, include *i*) a desired contact force that the robot is supposed to apply on an object in the environment (for instance, in the use case of cleaning a table) and *ii*) Cartesian commands produced by a certain type of active-impedance control approach that enables compliant robot motions [14], [29], [30]. Examples for *artificial* feed-forward joint torque specifications, that motivate the aforementioned solver's extension, include joint commands that are produced by null-space control approaches [3], [53], used for keeping a joint distant from its position limits [5].

In **Algorithm** 2, the extended solver with the above-described *artificial* drivers is presented. More specifically, the computations in lines 7, 9, 19 and 21 represent additional and adjusted equations that enable the solver to account for specifications of *artificial* Cartesian forces. On the other side, the computations in lines 17 and 21 represent adjusted equations that enable the solver to also account for specifications of *artificial* feed-forward joint torques.

The requirement for completing the above-mentioned extension defines that the torque contributions, originating from the *artificial* Cartesian forces and *artificial* feed-forward joint torques, must be included in the τ_{ctrl} term. More specifically, the computed *artificial* torque contributions should also constitute the motor commands since these forces do not already exist in the environment, i.e. these forces should be produced by the system's actuators. For that reason, the τ_{ctrl} term is extended with additional contributions in line 26.

An additional feature introduced in this extended solver is motivated by a practical insight. More specifically, the control commands computed by the solver for satisfying the desired task drivers may force the joint actuators to reach their saturations [5]. Such event could be caused in different ways, but most common examples include a task definition that drives a manipulator close to a singular configuration while performing a motion or, a task definition that specifies too strong forces or accelerations [5]. For that reason, a function that saturates the computed control torques has been introduced in line 26 and it is defined as follows:

$$saturate(\tau) = \begin{cases} \tau_{max}, & \tau > \tau_{max} \\ \tau, & \tau_{min} \leq \tau \leq \tau_{max} \\ \tau_{min}, & \tau < \tau_{min} \end{cases} \tag{4.6}$$

Here, τ_{max} and τ_{min} represent maximum and minimum torque saturation limits, respectively, defined by the robot's actuator model. With this modification, if a saturation event occurs in computations, the resulting control commands will not achieve the exact robot behaviour that was demanded by the task definition.

The saturation function that limits the computed control torques could be implemented in a different part of an application that uses this solver for evaluation of robot dynamics, i.e. the torque saturations could be performed outside the solver. In our case, the rationale for performing the saturation in solver's line 26, comes from a necessity to introduce the knowledge of actuators' torque limits in the following forward dynamics computations, i.e. resultant motion computations performed in lines 28 and 29. In this way, the computed resultant joint and segment accelerations, $\ddot{q}(\tau_{total})$ and $\ddot{X}(\ddot{q})$ respectively, are closely matching the achieved accelerations of the real system that uses the saturated torque actuator commands. This feature becomes even more important in an application where a user of the solver decides to exploit the aforementioned state variables for deriving additional state-related quantities, such as, for example, predictions of future robot joint space and/or Cartesian space motions.

Algorithm 2: Extended Popov-Vereshchagin ACHD Solver

Input : **Robot model**, $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{X}}_{\mathbf{0}}$, $\tau_{\mathbf{ff-physical}}$, $\tau_{\mathbf{ff-artificial}}$, $\mathbf{F}^{\mathbf{ext-physical}}$, $\mathbf{F}^{\mathbf{ext-artificial}}$, $\mathbf{A_N}$, $\mathbf{b_N}$

Output : $\tau_{\mathbf{ctrl}}$, $\tau_{\mathbf{total}}$, $\ddot{\mathbf{q}}$, $\ddot{\mathbf{X}}$

begin

  for i = 0 to N − 1 do

    ${}^{i+1}_{i}X = ({}^{d_i}_{i}X \;\; {}^{i+1}_{d_i}X(q_i))$;

    $\dot{X}_{i+1} = {}^{i+1}X_i\dot{X}_i + S_{i+1}\dot{q}_{i+1}$;

    $\ddot{X}_{bias,i+1} = \dot{X}_{i+1} \times S_{i+1}\dot{q}_{i+1}$;

    $F_{bias,i+1} = \dot{X}_{i+1} \times^* I_{i+1}\dot{X}_{i+1} - {}^{i+1}X_0^T \; F^{ext-physical}_{0,i+1}$;

    $F_{ext-artificial,i+1} = -\; {}^{i+1}X_0^T \; F^{ext-artificial}_{0,i+1}$;

    $F^A_{bias,i+1} = F_{bias,i+1}$;

    $F^A_{ext-artificial,i+1} = F_{ext-artificial,i+1}$;

    $I^A_{i+1} = I_{i+1}$;

  end

  for i = N − 1 to 0 do

    $D_{i+1} = d_{i+1} + S^T_{i+1}I^A_{i+1}S_{i+1}$;

    $P^A_{i+1} = 1 - I^A_{i+1}S_{i+1}D^{-1}_{i+1}S^T_{i+1}$;

    $I^a_{i+1} = P^A_{i+1}I^A_{i+1}$;

    $I^A_i = I^A_i + \; {}^iX^T_{i+1}I^a_{i+1}\,{}^iX_{i+1}$;

    $F^a_{bias,i+1} = P^A_{i+1}F^A_{bias,i+1} +$
    $I^A_{i+1}S_{i+1}D^{-1}_{i+1}(\tau_{ff-physical,i+1} + \tau_{ff-artificial,i+1}) + I^a_{i+1}\ddot{X}_{bias,i+1}$;

    $F^A_{bias,i} = F^A_{bias,i} + \; {}^iX^T_{i+1}F^a_{bias,i+1}$;

    $F^A_{ext-artificial,i} = F^A_{ext-artificial,i} + \; {}^iX^T_{i+1}P^A_{i+1}F^A_{ext-artificial,i+1}$;

    $A_i = {}^iX^T_{i+1}P^A_{i+1}A_{i+1}$;

    $U_i = U_{i+1} + A^T_{i+1}\{\ddot{X}_{bias,i+1} + S_iD^{-1}_i[\tau_{ff-physical,i+1} + \tau_{ff-artificial,i+1} -$
    $S^T_i(F^A_{bias,i+1} + F^A_{ext-artificial,i+1} + I^a_{i+1}\ddot{X}_{bias,i+1})]\}; \quad U_N = A^T_N\ddot{X}_0$;

    $\mathcal{L}_i = \mathcal{L}_{i+1} - A^T_{i+1}S_{i+1}D^{-1}_{i+1}S^T_{i+1}A_{i+1}; \quad \mathcal{L}_N = 0$;

  end

  $\nu = \mathcal{L}^{-1}_0(b_N - A^T_0\ddot{X}_0 - U_0)$;

  for i = 0 to N − 1 do

    $\tau_{ctrl,i+1} =$
    $saturate(-S^T_{i+1}A_{i+1}\nu - S^T_{i+1}F^A_{ext-artificial,i+1} + \tau_{ff-artificial,i+1})$;

    $\tau_{total,i+1} =$
    $\tau_{ff-physical,i+1} + \tau_{ctrl,i+1} - S^T_{i+1}\{F^A_{bias,i+1} + I^A_{i+1}({}^{i+1}X_i\ddot{X}_i + \ddot{X}_{bias,i+1})\}$;

    $\ddot{q}_{i+1} = D^{-1}_{i+1}\tau_{total,i+1}$;

    $\ddot{X}_{i+1} = {}^{i+1}X_i\ddot{X}_i + \ddot{q}_{i+1}S_{i+1} + \ddot{X}_{bias,i+1}$;

  end

end

5

Adaptive Control

In the related work section 2.2, a novel type of *state of the art* adaptive controller has been mentioned, namely, the Adaptive Bias Adaptive Gain (ABAG) algorithm [6]. In the original controller's publication [6], the presented application is dealing with the 1D control for each motor independently, on a test aerial platform. However, its properties make the ABAG controller a promising candidate for controlling task executions of other multi-actuator systems, such as robot manipulators and humanoid robots. More specifically, multiple independent ABAG controllers can be employed for enabling *i*) the accurate tracking of command setpoints for each robot's joint and moreover, *ii*) the accurate task execution in each direction of robot's operational space. In this chapter, the aforementioned algorithm will be reviewed in more details.

5.1 Description

In [6], researches have developed an *adimensional* adaptive controller; in the sense that the algorithm's computations are completely based on dimensionless quantities. More specifically, the signals that constitute inner state and output of the controller represent percentages of the application-specific maximum values that the controller acts on. Additionally, each of the above-mentioned signals contains information about the direction in which its respective adaptation advances.

In **Algorithm** 3, the Adaptive Bias Adaptive Gain (ABAG) controller is presented. As previously mentioned in section 2.2, this algorithm is controlling a mechanical system based on the *trend* of the instantaneous error or, in other words, by looking in the *past-time* error-horizon. Here, the error-trend information $\overline{e}_{sign}$ is derived by applying a low-pass filter on the sign of input error e. This means that the controller does not take the magnitude of input error into account while computing control commands. This pre-processing of input error is performed in line 5. Here, $sgn()$ stands for the sign function. In the original formulation of ABAG controller [6] and in **Algorithm** 3, the error-sign is filtered using a *first-order* low-pass filter. However, this error pre-processing procedure can be adjusted (if computational hardware allows) to enable a more powerful error signal estimation, without alternating the core idea of the ABAG controller; for instance, by introducing a *higher-order* low-pass filter.

Algorithm 3: Adaptive Bias Adaptive Gain (ABAG) Controller [6]

Input : $e_i \in R$ // System's Error

Parameters : $\alpha \in (0,1)$, // Filtering Factor
$\overline{e}_b \in (0,1)$, // Bias Adaptation Threshold
$\delta_b \in (0,1)$, // Bias Adaptation Step
$\overline{e}_g \in (0,1)$, // Gain Adaptation Threshold
$\delta_g \in (0,1)$ // Gain Adaptation Step

State Signals: $\overline{e}_{sign,i} \in [-1,1]$, // Low-Pass Filtered Error Sign
$b_i \in [-1,1]$, // Adaptive Bias
$g_i \in [0,1]$ // Adaptive Gain

Output : $u_i \in [-1,1]$ // Control Command

1 **begin**
2 $i = 0$; $\overline{e}_{sign,0} = b_0 = g_0 = u_0 = 0.0$;
3 **while** $++i$ **do**
4 // Low-pass filtering of error sign.
5 $\overline{e}_{sign,i} = \alpha \cdot \overline{e}_{sign,i-1} + (1-\alpha) \cdot sgn(e_i)$;
6 // Bias signal adaptation.
7 $b_i = sat_{[-1,1]}[b_{i-1} + \delta_b \cdot hside(|\overline{e}_{sign,i}| - \overline{e}_b) \cdot sgn(\overline{e}_{sign,i} - \overline{e}_b)]$;
8 // Gain signal adaptation
9 $g_i = sat_{[0,1]}[g_{i-1} + \delta_g \cdot sgn(|\overline{e}_{sign,i}| - \overline{e}_g)]$;
10 // Command signal adaptation
11 $u_i = sat_{[-1,1]}[b_i + g_i \cdot sgn(e_i)]$;
12 **end**
13 **end**

Another valuable feature of this algorithm is that it approximates unknown (non-modelled) dynamics parameters (inertia-mass, damping, elasticity) of the controlled system [6], by adapting to their effects. More specifically, the information about approximated dynamics is resembled by the controller's *bias* term b, which is constantly adapting its value by following the trend in error (see line 7). Here, $hside()$ stands for the Heaviside step function and $sat()$ stands for the saturation function. Using this information, if natural/existing forces (originating from *steady* disturbances acting on the system) contribute positively (i.e. assist) in keeping the plant within the desired state, the ABAG controller will take advantage of these forces to control the plant. This feature defines the *lazy* property of the ABAG algorithm and its mathematical description is provided in [6]. On the other hand, if the aforementioned forces contribute negatively (i.e. interfere), the *bias* signal will serve to compensate these effects. For instance, in the case of the application presented in the original paper [6], *steady* disturbances are the ones that originate from the effects of *i*) friction in the motors and *ii*) air resistance acting on the actuator's propellers.

The inner-state of the ABAG controller consist of an additional signal, i.e. the *gain* term g which is updated in parallel with the controller's *bias* term b (see line 9). However, the role of this signal is different. More specifically, the gain term is used to act against sudden (not steady) disturbances that affect the controlled system. Examples of such disturbances include instant and relatively large changes of system's desired state, sensor noise, etc. Furthermore, the *gain* term can be seen as a signal that is responsible for providing a good response of the controller until the *bias* signal converges to a stable value, i.e. approximates the non-modelled dynamics of the system. In other words, the reactivity of these two signals (*bias* and *gain*) on changes in the error value differs, in the sense that the *gain* term reacts faster on the input error, compared to the bias term [6].

It is worth mentioning that the P and I parts in the PID controller share similarities with the above-described *gain* and *bias* signals, in terms of their role in the control of a system. More specifically, the role of the I part is similar to the role of the *bias* signal, in the sense that I in PID builds its value over time and compensates for *steady* disturbances acting on the system. On the other side, the P part has a similar role as the *gain* signal, i.e. the P in PID is responsible for an instantaneous reaction to the system's error. However, differences between these two controllers can be seen in the adaptation laws that govern the formation of *bias* and *gain* terms on one side, and the adaptation laws that govern the formation of I and P terms on the other.

In the ABAG control strategy, the output signal u is constructed from values of both *bias* and *gain* terms (see line 11). In the stage where the controlled plant is not in a steady state, both *gain* and *bias* signals contribute to the control magnitude. However, when the system reaches a steady-state, the absolute value of the *gain* signal decreases and the *bias* signal takes over as the main contributor to the control command u [6].

As previously mentioned, the algorithm's computations are based on dimensionless quantities and values of its signals represent percentages of an application-specific maximum value that the controller acts on. More specifically, an application that uses the ABAG controller gives a more concrete meaning to the final control command u outside the controller's computations. For instance, if the ABAG algorithm is applied to control the motion of a single robot joint, the aforementioned maximum value would, in this case, represent a nominal torque that the joint actuator can produce and the control command u would represent a percentage of this nominal torque.

5.2 Parameterization

In the ABAG formulation, *waveforms* of both *bias* and *gain* signals are represented by fixed functions, also referred to as *decision maps* (ξ_b and ξ_g in figure 5.1) [6]. These functions are defined (parameterized) in advance by a control designer, depending on the desired behaviour of the system and its capabilities. This feature, i.e. the explicit design of *bias* and *gain* waveforms, enables better

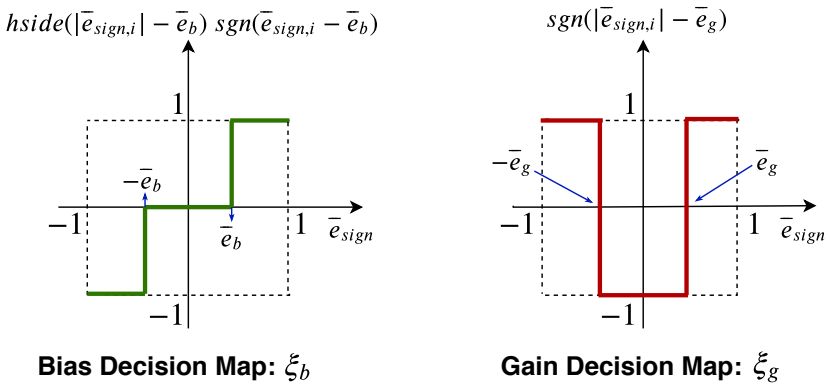

Figure 5.1: Decision maps in the ABAG controller. Here, $hside()$ stands for the Heaviside step function and $sgn()$ stands for the sign function. Figure based on [6].

predictability and *stability* of the controller's behaviour, compared to, for example, the PID controller, where formation of its control signals is highly depended on the actual error [15]. For a more detailed analysis of controller's stability, we refer the reader to [6].

An additional feature of the above-mentioned *decision maps* allows for specification of control *dead zones*, i.e., in this case, regions in the value of the filtered error for which *bias* and *gain* signals are not adapted (see figure 5.1). Only after the filtered error crosses the allowable tolerances, the controller starts alternating values of its signals. Furthermore, the controller's formulation enables a control designer to choose maximum values for the *bias* and *gain* terms by adjusting the limits of its respective saturation functions. This feature enables the possibility of limiting the controller's output, such that a wind-up situation does not occur [15].

In the following paragraph, each of the controller parameters will be discussed in more detail:

- **Filtering Factor α:** This parameter is used for adjusting the smoothness of the filtered error-sign signal $\bar{e}_{sign}$ [6]. Noisier state estimation, higher α required, or in other words, higher its value, the error sign will be more filtered. However, as the low-pass filter provides the error-trend information based on which the *bias* and *gain* signals are later on constructed, the chosen value for this filtering factor has a high influence on the choice for values of the following parameters.

- **Bias Threshold $\bar{\mathbf{e}}_\mathbf{b}$:** This parameter defines how far should the controller ignore the filtered error before it starts adapting the *bias* signal, i.e. it starts alternating the controller's current approximation of non-modelled dynamics. This tolerance represents the previously mentioned *bias* dead zone.

- **Gain Threshold $\bar{\mathbf{e}}_\mathbf{g}$:** In simple words, and given the role of *gain* signal, this parameter defines when should the controller start "panicking". More specifically, this parameter defines how far should the controller ignore the filtered error, before it starts adapting the *gain* signal. This tolerance represents the previously mentioned *gain* dead zone.

- **Bias and Gain Steps $\delta_{\mathbf{b}}$ & $\delta_{\mathbf{g}}$:** These two parameters define the adaptation speed of *bias* and *gain* signals, respectively. Additionally, they have a high impact on how smooth the signals and overall control will be.

- **Saturation Limits:** The meaning behind these parameters is twofold. First, a saturation limit defines a percentage that represents a maximum possible contribution of a signal to the overall control. Second, a saturation limit also defines in which direction a signal can contribute, positive or negative. For example, in the controller's formulation presented in **Algorithm** 3, the *bias* signal is allowed to contribute up to 100% in both negative and positive directions. On the other side, the *gain* signal is allowed to contribute up to 100% only in a positive direction, and not have any contribution in a negative direction. The reason is that in the final equation of the ABAG controller (line 11), the *gain* signal is multiplied by the sign of input error, and this operation governs if the control command u should be reduced or increased by the *gain* magnitude.

It is important to mention that the shapes of the aforementioned *decision maps* (for both *bias* and *gain* terms) must be designed carefully, such that a good trade-off between controller's responsiveness and chattering behaviour is achieved [6]. More specifically, if we choose a set of parameters in such a way that the *gain* signal is kept constantly high throughout the control, the response time would be very short. However, in this case, the control will be, at the same time, characterized by high chattering, which in turn creates a harmful effect on the system's actuators [6].

6

Realization of the Lazy Robot Control

The control strategies derived in this project should not only address challenges of the use cases presented in section 1.2 but also, relax the robot control while ensuring correct execution of these tasks. For that reason, this chapter describes the developed task specification models and control architectures for enabling a lazy robot control. Additionally, it discusses how the previously presented components, the acceleration-constrained hybrid dynamics (ACHD) solver (chapter 4) and ABAG controller (chapter 5) are used as building blocks of the derived control architectures, along with the newly developed components that are necessary for ensuring the correct task executions in the above-mentioned use cases.

6.1 Pre-Grasp Motion and Common Challenges

6.1.1 Strategy

In order to create a *lazy* control strategy that would enable execution of the task defined under this use case, several objectives are considered while meeting the imposed challenges, i.e. several aspects of robot control are subject to relaxation.

As mentioned in section 1.2, motion constraints are usually defined based on the accuracy and precision tolerances, obstacles in the environment, etc. Approaches such as MPC [11] or a group of sampling-based motion planning algorithms [44], [46] are satisfying the task specifications by performing expensive computations of explicit trajectories. Additionally, the resulting robot motions are not characterized by high agility, due to the fact that the motion plans are over-constraining the system. However, these strategies are not considering the fact that this problem can be *relaxed* by leaving some of the imposed task constraints to be resolved *instantaneously*, by a good design of the control architecture.

The first objective here is to eliminate the generation and optimization of explicit robot trajectories from the computations of motions and instead, create a less constraining motion specification. This is achieved by constraining the robot motion via a *tube*. In the practical context, a robot state is considered to be allowed if it is inside a *tube* or, in other words, an area in the state space. Boundaries of this tube are derived from the task-specified motion tolerances (e.g. the accuracy and precision requirements) [15], [78]. Furthermore, in this control approach, the

area error is taken into account instead of the *setpoint* error. The idea is to *guide* the robot through the *tube* directly towards the goal state, which is, in terms of this use case, an area around the object. As a result, the trajectory (set of exact robot poses) would be an outcome of the robot's motion, but not planned in advance or online. In this way, the control is less constrained [15], i.e. a bigger space of allowed states is defined and thus, a larger set of feasible control solutions exist.

For addressing the first challenge defined under this use case and moreover, enabling the overall *lazy* control strategy, the previously introduced Adaptive Bias Adaptive Gain (ABAG) controller and Popov-Vereshchagin constrained hybrid dynamics solver are employed. The reason for choosing the ABAG controller, as one of the building blocks of our control strategy, is the controller's ability to resolve the imposed control constraints via simple computations and at the same time provide us useful control features. As mentioned in chapter 5, the explicit design of *bias* and *gain* waveforms, enables better *predictability* and *stability* of the controller's behaviour, compared to, for example, the PID controller [15]. Additionally, the controller enables the possibility of limiting the control output, such that a wind-up situation does not occur. On the other side, the reason for choosing the Popov-Vereshchagin hybrid dynamics algorithm, as one of the building blocks of our control strategy, is its ability to resolve instantaneous robot dynamics analytically, with linear-time, $O(n)$ complexity [3]. Moreover, the solver exposes tree different task interfaces which enable specification of the significant number of different task models, i.e. this algorithm computes required joint space control commands based on defined Cartesian acceleration constraints, feed-forward joint torques and the external Cartesian forces.

For meeting the second challenge of this use case, computations performed by the aforementioned ABAG controller are not sufficient. More specifically, this algorithm is controlling a system based on a trend of the task execution error or, in other words, by looking in the past-time error-horizon. Thus, the controller solely cannot optimize effects influenced by future events in task execution. However, an additional component can be considered in this control strategy for addressing this challenge, i.e. a model prediction method that estimates future robot states. The list of possible methods to be used for providing the required state predictions is extensive [16], [17], [27]. As presented in chapter 3, for providing this feature, current approaches such as MPC [11], [28], [41] perform expensive finite-horizon predictions. However, the objective here is to relax these computations and therefore, the approach is to simply integrate the measured robot's Cartesian velocities over a particular and finite time interval. In such a way, the information about future robot states can be received via inexpensive computations. The justification for performing this type of predictions comes from the fact that model parameters of a robot and its environment are anyway not completely accurate [10], [12], [79]; it is only possible to have an approximation of these models. Additionally, in the sensor information, noise and uncertainties always exist [13], [14]. The benefit of combining this component with the ABAG controller is an early and smooth reactivity of the system to an error in the motion before

it occurs. Thus, the combined effort of these two components would keep robot states within allowed bounds while still producing very small increments in the control commands. Furthermore, this type of "cheap" future-horizon computations enable us to implement a control loop with higher-frequency, i.e. expose feedback information more frequently.

For meeting the first challenge listed as common for all use cases (see section 1.2) additional features of the already considered components, the Popov-Vereshchagin hybrid dynamics solver and ABAG controller are exploited. More specifically, as described in chapter 4, the Popov-Vereshchagin algorithm takes advantage of the existing forces acting on the system while resolving the *instantaneous* dynamics, i.e. while minimizing the acceleration energy of the system, based on the *known model parameters* [4]. On the other side, the ABAG controller has also a *lazy* property, such that it takes advantage of existing forces such as, for example, friction in the motors, air resistance, gravity, etc., while computing the *non-instantaneous* control of the system, based on the approximation of *non-modelled* parameters [6]. A mathematical description of this feature is provided in [6]. By leaving many of the task objectives to the system's natural dynamics, the combined effort of these two components improves the energy consumption of the system's actuators.

The second challenge listed as common for all four use cases has not been considered very often by the current state of the art approaches, in the field of robot dynamics and control. Many strategies are not exploiting prior knowledge about the task and environment in order to *relax* the control of robot motions and forces. More specifically, instead of leaving some of the DOFs unspecified in the task definition, many approaches are explicitly defining and resolving motion and force constraints [37], [80], [81]. The aforementioned feature is important, since different phases may exist in the task execution and for every phase, different constraints are imposed on the robot. For example, in this use case, if necessary, two phases can be defined, namely an "approach" and a "fine positioning" phase. In an "approach" phase, only the control of robot wrist's position may be required, while in a "fine positioning" phase, only the control of the wrist's orientation may be required. For addressing this challenge, the objective is to leave the unconstrained motion directions to "emerge naturally" [5], [18]. More specifically, for enabling this type of relaxation the already considered Popov-Vereshchagin solver can be exploited. The reason for choosing this component is that even in the case of incomplete tasks definitions (some of the DOFs left unspecified) this hybrid dynamics algorithm produces a solution to the inverse dynamics problem, by naturally resolving robot motions based on the "*criterion of least constraints*" [2], defined by the Gauss' principle [5], [72] (as described in section 4.1).

6.1.2 Application

Task specification

In order to develop a task specification formalism that will model the desired motions and forces that are necessary for the execution of the tasks described in section 1.2, the well-known *Task Frame Formalism* (TFF) [81], [82] approach was used as a motivation. However, in the original TFF approach, for each of the six

task frame axes, it is required to define a motion or force constraint. Furthermore, this formalism does not consider definitions of the above-presented tube bounds. Nevertheless, a *modified* version of TFF was developed here to enable specifications of tube tolerances and the possibility for leaving some of the DOFs out of the task definition. Here, an example of the task specification created for the first use case is presented in the following:

Task specification for the first use case: Example 1

move compliantly { // with task frame directions

- ${}^{T}X$: velocity v [m/s], velocity-tolerance σ_v [m/s]
- ${}^{T}Y$: 0 [m], position-tube σ_y [m]
- ${}^{T}Z$: 0 [m], position-tube σ_z [m]
- ${}^{T}aX$:
- ${}^{T}aY$:
- ${}^{T}aZ$:

} no specification

} until either:

- goal area reached, tolerance σ_x [m]
- or ${}^{T}F_X$ / ${}^{T}F_Y$/ ${}^{T}F_Z > f$ [N]
- or task time $> t$ [s]

In the above task model, all terms are specified with respect to an orthogonal user-defined *task frame* (T). These terms represent motion and force variables for the robot's end-effector (tool) segment. Here, the position tube bounds for linear Y and Z directions are noted with σ_y and σ_z respectively and the velocity tolerance is noted as σ_v. However, the term σ_x represents goal area tolerance in the linear X direction. Terms ${}^{T}F_X$, ${}^{T}F_Y$ and ${}^{T}F_Z$ represent measured contact linear forces. On the other side, terms f and t denote force and time thresholds, respectively. In this use case, the *task frame* is placed at the pre-grasp location, which is, together with the frame's orientation, determined beforehand by a certain grasping strategy.

A simple illustration of the robot action necessary for realizing the type of task defined under the this use case is presented in figure 6.1. In this illustration, three frames are depicted, namely the *task frame*, base frame and end-effector frame.

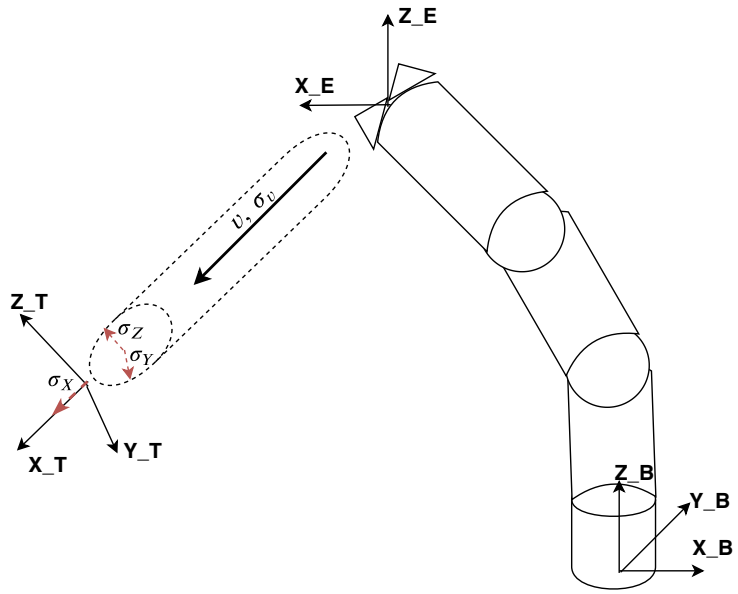

Figure 6.1: Illustration of the robot action performed in the first use case.

Another example of the task specification created for the first use case is presented in the following:

Task specification for the first use case: Example 2

move compliantly { // with task frame directions

- ${}^{T}X$: velocity v [m/s], velocity-tolerance σ_v [m/s]
- ${}^{T}Y$: no specification
- ${}^{T}Z$: 0 [m], position-tube σ_z [m]
- ${}^{T}aX$:
- ${}^{T}aY$:
- ${}^{T}aZ$:

} 0 [rad], orientation tube σ_a [rad]

} until either:

- goal area reached, tolerance σ_x [m]
- or ${}^{T}F_X$ / ${}^{T}F_Y$ / ${}^{T}F_Z > f$ [N]
- or task time $> t$ [s]

The term σ_a represents orientation-tube tolerance. The reason for heaving a single number for representing the tolerance on an orientation originates from the choice of the rotation representation, namely the *axis-angle* representation [32], [83], [84], based on which the orientation-error calculation is performed. A more detailed discussion on this topic will be presented in the following.

Control Architecture

A control architecture that enables execution of the task defined under the first use case is presented in figure 6.2. In this control loop, the task error is calculated based on *i*) the above-presented task specification given as input and *ii*) the measured and predicted state of the robot. In this case, the predicted state is taken into consideration for calculating robot deviations from the pose-tube bounds. The rationale for using the predicted state information in the pose-tube error calculations is presented in section 6.1.1. On the other hand, the current state (measured velocity) is directly used for calculating robot deviation from the velocity-bound. The complete task error defines: *i*) a 6×1 vector E associated with robot pose deviations and *ii*) a scalar value $e_{x,v}$ associated with robot velocity deviations. Every DOF defined in the task specification has its associated error element in the aforementioned task error. However, the computed robot position error along the linear X direction, will not be used in the following computations of the adaptive controller. Nevertheless, this error element will be used for monitoring purposes by the *finite state machine* (FSM) component, which will be presented in the following. Additionally, the error element that is associated with an unconstrained DOF has zero value.

Both of the aforementioned error quantities are passed as input to an adaptive controller that consists of six independent (decoupled) ABAG controllers. The result of this component is a 6×1 vector of Cartesian control commands $\vec{u}$, defined in percentages. However, as presented in chapter 5 (section 5.1), more concrete meaning to these control commands is given outside the controller's computations. For that reason, each element of the vector $\vec{u}$ is scaled with its corresponding element in a 6×1 vector of *maximum commands* that, in this case, represent the maximum allowed *acceleration energy* setpoints for the robot's end-effector. The result of this operation is a 6×1 vector of *acceleration energy* setpoints b_N and this vector is expressed w.r.t. *task frame*. However, the next component in this control loop, namely the Popov-Vereshchagin solver, expects that a Cartesian space motion driver is expressed w.r.t. the robot *base frame*. For that reason, is it necessary to perform a coordinate transformation of these values. As presented in chapter 4 (section 4.2), the *acceleration energy* setpoints are related to the A_N, in this case, a 6×6 matrix of unit constraint forces. Thus, for performing the coordinate transformation for this motion driver it is necessary to *only* transform the A_N matrix. However, in the case of an unconstrained DOF in the task specification, before the transformation operation is performed the unit column vector of the A_N matrix that is associated with this unconstrained DOF is set to a 6×1 vector of zeros.

The output of the constrained hybrid dynamics solver consists of two quantities, joint control commands τ_{crtl} and resultant joint accelerations $\ddot{q}$. However, only the computed joint torques τ_{crtl} are used as control commands for robot actuators. Nevertheless, the resultant joint accelerations $\ddot{q}$ are used as a part of the robot's state, by the *finite state machine* (FSM) component which will be presented in the following.

After the control commands are applied, sensor measurements, i.e. joint

positions q and joint velocities $\dot{q}$ are used by the next component, the forward kinematics solver. This unit is computing the current robot state in Cartesian space, i.e. the pose and velocity of each robot segment. These quantities are expressed w.r.t. robot base frame, and for that reason, it is necessary to perform a coordinate transformation from the base frame to *task frame*. However, in order to complete computations of the control feedback, the next step is to perform the future Cartesian state predictions that are necessary for the task error computations.

The final component in this architecture is the *finite state machine* (FSM) which monitors the robot task execution via E term and several aspects of the robot's state, such as measured joint positions q and joint velocities $\dot{q}$, the resultant joint accelerations $\ddot{q}$ computed by the hybrid-dynamics solver and measured contact forces that are exerted on the robot's end-effector. Additionally, this unit is responsible for alternating the control modes of this loop, based on the events that occur in run-time and the predefined control policy.

More detailed discussions about many of the above-considered components will be presented in the following.

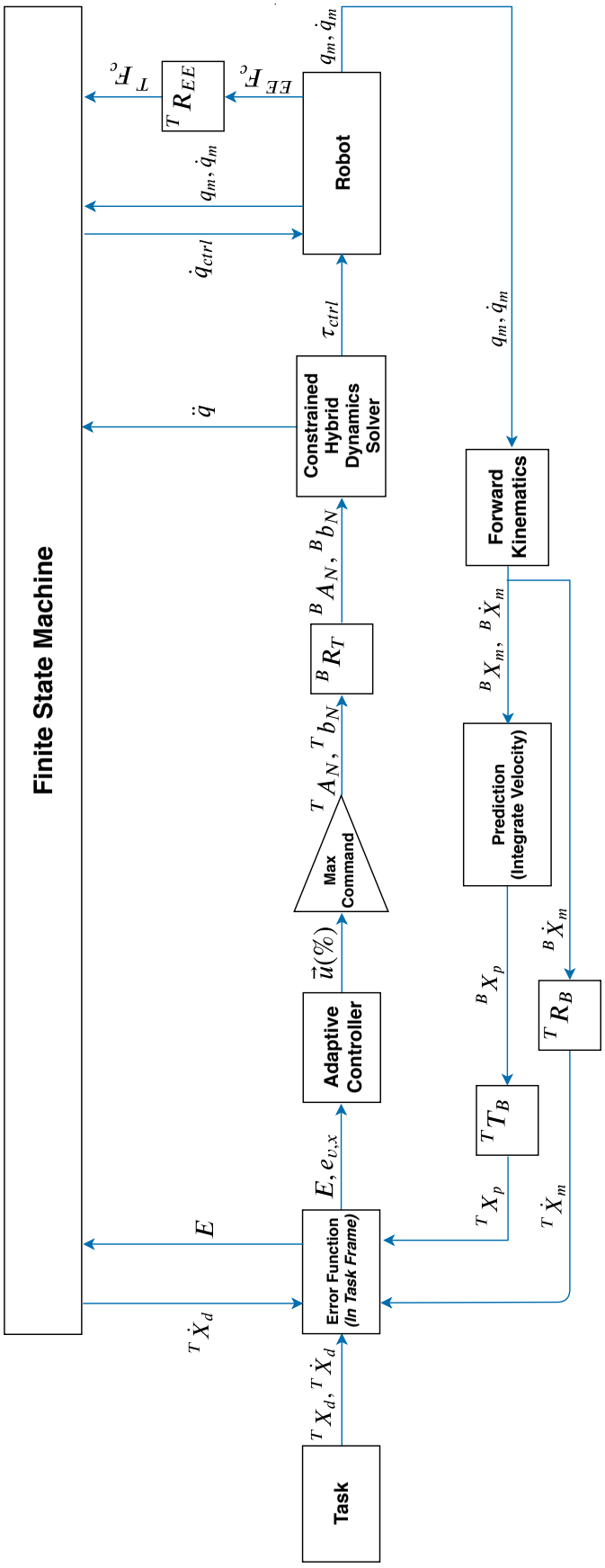

Figure 6.2: Main control loop for the first use case.

Error Function: As previously mentioned, in this use case the task error is calculated based on *i*) the task specification given as input and *ii*) the measured and predicted state of the robot. In this case, the predicted state is taken into consideration for calculating robot deviations from the pose-tube bounds. The rationale for using the predicted state information in the pose-tube error calculations is presented in section 6.1.1. On the other hand, the current state (measured velocity) is directly used for calculating robot deviation from the velocity-bound. The complete error function graph is presented in figure 6.3. Here, the subscripts l and a stand for linear and angular, respectively.

In this function, the pose error calculation is decoupled, in the sense that, the position error is calculated independently from the orientation error. The calculation of the position error is straightforward, in contrast to the calculation of the orientation error. More specifically, in the case of position error, the difference between the desired and predicted end-effector's position is defined by a 3D vector r_e which represents the relative translation between these two points, expressed w.r.t *task frame*. In other words, this difference defines the linear velocity $\dot{X}_l$ that can bring the end-effector from the predicted position to the desired one, in one unit of time [32], [83]. However, the approach taken for computation of the orientation error is motivated by the work presented in references [32], [83], [84]. Here, the term R_e represents the relative orientation between the predicted and desired end-effector state, expressed w.r.t *task frame*. However, in order to compute an error vector that the controller can interpret, it is necessary to compute the *logarithm* of this rotation matrix R_e. More specifically, the result of this operations is an angular velocity $\dot{X}_a$ that would cause the end-effector frame to rotate from the predicted to the desired orientation, in one unit of time [32]. Based on the *exponential coordinate* representation [84], this rotational motion is performed around a rotation axis, i.e. a unit vector $\hat{\omega}$ over a rotation angle θ. For more information about this type of representation, the reader can refer to [32]. Nevertheless, the *exponential coordinate* or, in other words, *axis-angle* representation of relative orientation was used for defining the orientation-tube tolerance, in the previously described task specification model. More specifically, the (single number) orientation-tube bound is directly related to the above-presented rotation angle θ.

Since the ABAG controller considers only the sign of input error, in its computations, it is necessary to additionally perform pre-processing of the control error, such that the tube boundaries can be *explicitly* taken into account in the overall control loop. This pre-processing procedure is performed for both, pose-tube and velocity-tube error calculations (see the graph in figure 6.3).

Adaptive Controller: As mentioned above, this component consists of six independent (decoupled) ABAG controllers and it is presented by the graph in figure 6.4. More specifically, a single ABAG controller is responsible for controlling the motion in only one DOF, defined in the task specification. As it can be seen from the graph, in the linear X-direction tube-speed is controlled, while in other directions robot deviations from the pose-tube are controlled.

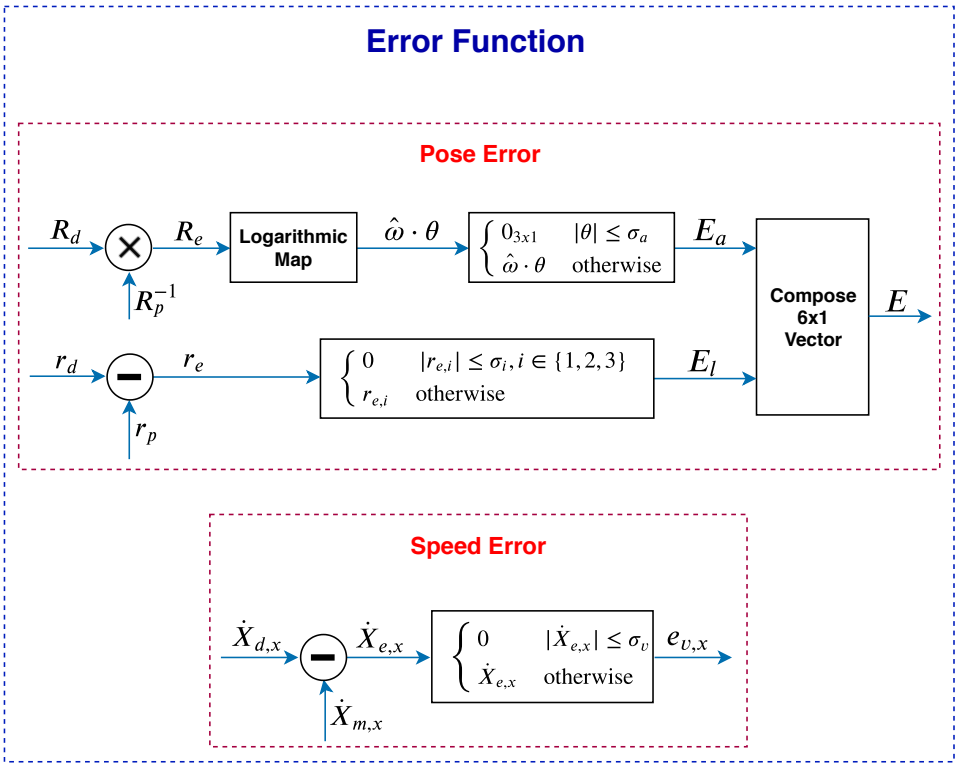

Figure 6.3: Error function for the first use case. The complete task error defines: *i*) a 6×1 vector E associated with robot's pose-tube deviations and *ii*) a scalar value $e_{x,v}$ associated with robot's velocity deviations.

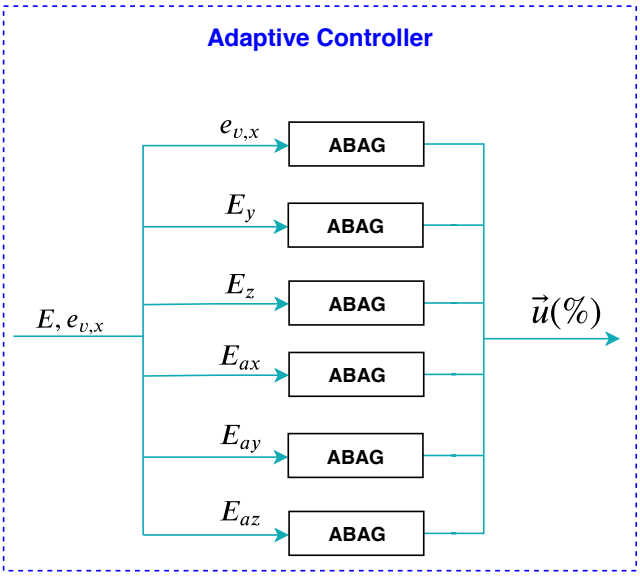

Figure 6.4: Adaptive controller for the first use case. In the linear X direction, tube-speed is controlled. In other directions, robot deviations from the pose-tube are controlled.

Constrained Hybrid Dynamics Solver: In this component, i.e. the Popov-Vereshchagin hybrid dynamics solver, the *acceleration constraint* motion driver is used for specifying the instantaneous end-effector motion constraints (see section 4.2). The algorithm is computing three output quantities, joint control commands τ_{ctrl}, resultant joint accelerations $\ddot{q}$ and resultant Cartesian accelerations $\ddot{X}$ of each robot segment. However, it is necessary to mention that a saturation function that limits the computed control torques τ_{ctrl} is implemented as a part of the solver's inner computations (for more information, see section 4.3).

Prediction: As previously mentioned, this component is performing computations of future robot Cartesian states, which are required by the above-described task error calculations. Here, the approach is to simply integrate the measured robot's Cartesian velocities, i.e. the end-effector's twist over a particular and finite time interval. In other words, the idea is to estimate what will be the future end-effector's pose, if the current velocity of this segment is maintained for a finite time horizon.

Similarly to the case of the task error function, the prediction method is also decoupling linear and angular parts. More specifically, the future end-effector's position is estimated independently from its future orientation. In order to calculate the relative rotation that is caused by the angular velocity $\dot{X}_a$ (in one unit of time), it is necessary to compute the *exponential* of this angular velocity [32], [85], [86]. As it can be seen from the graph in figure 6.5, the *body-fixed* twist is used in this integration procedure [87].

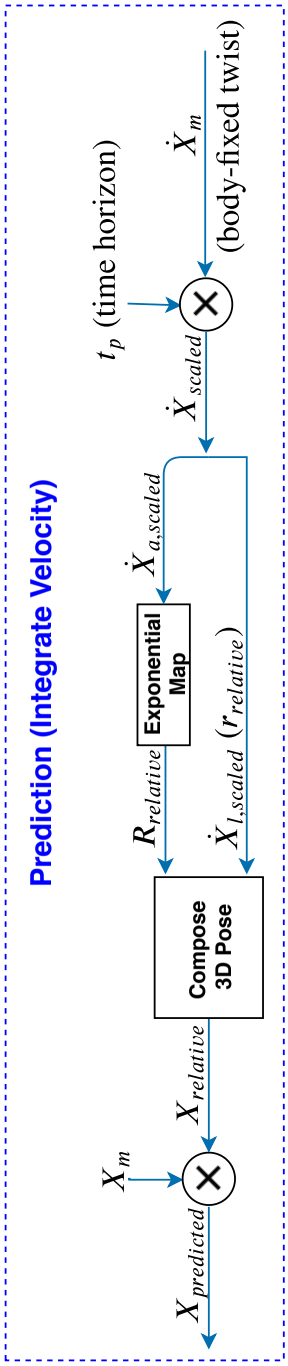

Figure 6.5: (Decoupled) Prediction calculations for estimating future end-effector poses.

Finite-State Machine (FSM): The *finite state machine* component provides functionalities that, at runtime, monitor execution of the specified task, i.e. judge if the task constraints are satisfied w.r.t. defined tolerances [15]. Additionally, the FSM monitors the robot's internal state, i.e. it is responsible for providing safety monitoring to ensure that the commanded joint torques τ_{crtl} will not drive the robot system over its joint limits. The information about robot's safety is derived based on *i*) the measured joint positions q and joint velocities $\dot{q}$, and *ii*) the resultant joint accelerations $\ddot{q}$ computed by the Popov-Vereshchagin solver. More specifically, the safety component is exploiting the existing feature of this hybrid dynamics solver, i.e. its *forward dynamics* calculations to estimate the future robot joint states. Here, the resultant joint accelerations $\ddot{q}$ are integrated over two time-steps and in each time-step, the predicted joint positions q and joint velocities $\dot{q}$ are checked for their respective limit violations. If a limit-violation event occurs, the control policy considered in this work defines an immediate stopping of all robot joint motions. Moreover, in the case of this event, the robot actuators are controlled via the *joint velocity* interface and the control commands are specified in $\dot{q}_{ctrl}$ term.

A graphical representation of this FSM is illustrated in figure 6.6. When it starts, the finite state machine first enters the *STOP_MOTION* state, in which the system is held in a static configuration i.e., the robot joints are commanded to hold their current positions. Further control states, such as *START-TO-CRUISE* and *CRUISE_THROUGH_TUBE* are motivated by the work presented in [15]. More specifically, the former control state defines a motion behaviour where the robot is moving towards the task-specified tubular region. However, in this state, zero tube-speed is maintained, i.e. the robot is not allowed to move (cruise) along the linear X-direction (defined in the task specification), until the robot reaches the tubular region. On the other side, the latter control state defines a motion behaviour where the robot is *cruising* through the tubular area, i.e. moving towards the final goal state while keeping itself within the task-specified tube bounds. Events that trigger the connecting transitions between the aforementioned states in FSM, are the following [88]:

- *e_initialized*: defines that the system is initialized [88].
- *e_timeout*: defines that the task time-limit is reached [88].
- *e_contact*: defines that the robot's sensors have sensed a contact with the environment [88].
- *e_joint_limit*: defines that the robot has reached a joint limit.
- *e_out_of_goal_area*: defines that the robot is far away from the final goal area.
- *e_goal_area*: defines that the robot has reached the final goal area and it is within the tube bounds.
- *e_out_of_tube*: defines that the robot is outside of the tubular area.

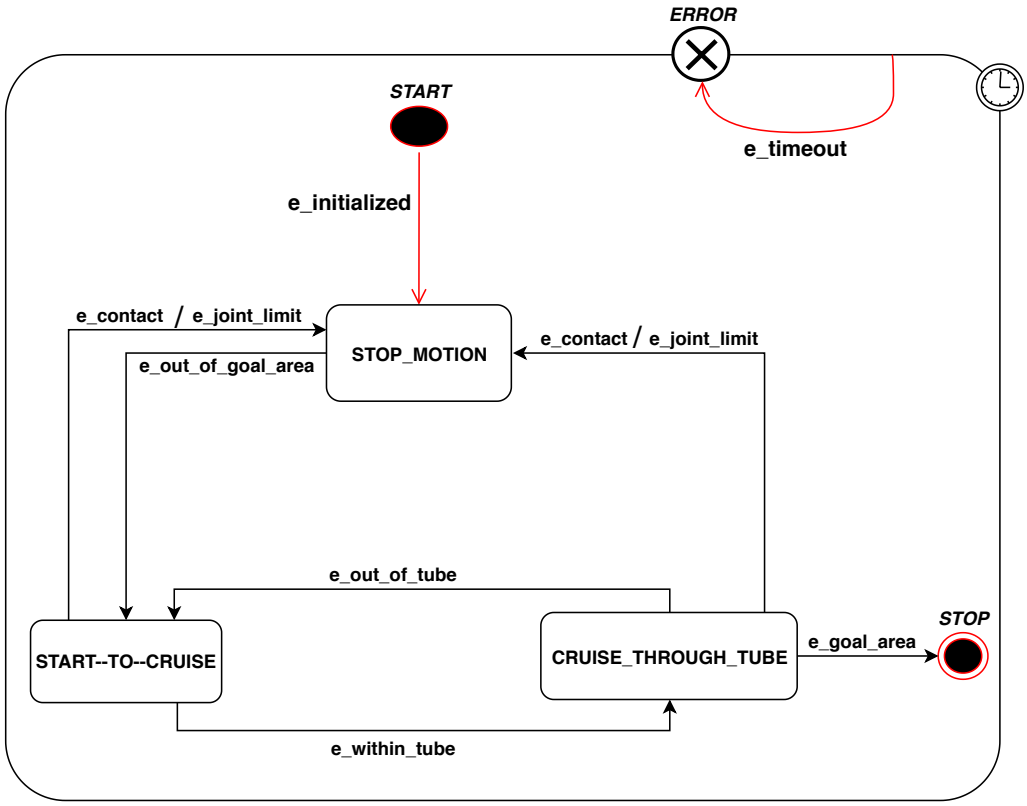

Figure 6.6: Finite state machine developed for the *pre-grasp motion* task. Figure based on [88].

- *e_within_tube*: defines that the robot is inside of the tubular area.

6.2 Following a Pre-defined Path

6.2.1 Strategy

The only difference between this use case and the previous one is that here, the task is defined with multiple goal states/areas instead of just one. More specifically, the robot is commanded to follow predefined position waypoints along the Cartesian path, instead of performing a motion towards a single goal area. Nevertheless, the same challenges that are necessary to be addressed in the first use case hold in this scenario as well.

In order to create a *lazy* control technique that would enable execution of the task and meet the challenges defined under this use case, the same aspects of the robot control that are considered in section 6.1.1 are subject to relaxation, here as well. More specifically, the idea is to use the same strategy developed for the first use case. However, the difference in this use case will be in the application of the strategy.

6.2.2 Application

Task specification

Here, the same task specification formalism that was developed in section 6.1.2, will be used for modelling the motions required for the execution of this type of task. However, in this case, each waypoint in the user-defined Cartesian path has associated its own *task frame*. This means that it is necessary to define one instance of the task specification for each waypoint in this path. Nevertheless, if required, the aforementioned (task specification) instances can share the same task parameters, such as tube bounds, contact force thresholds, goal area tolerances, etc., while being associated to different *task frames*. An example for one of the task specification instances, created for this use case, is presented in the following:

Task specification for the second use case: Example 1

move compliantly { // with task frame directions

- ${}^{T_k}X$: velocity v [m/s], velocity-tolerance σ_v [m/s]
- ${}^{T_k}Y$: 0 [m], position-tube σ_y [m]
- ${}^{T_k}Z$: 0 [m], position-tube σ_z [m]
- ${}^{T_k}aX$: } no specification
- ${}^{T_k}aY$:
- ${}^{T_k}aZ$:

} until either:

- goal area reached, tolerance σ_x [m]
- or ${}^{T_k}F_X$ / ${}^{T_k}F_Y$/ ${}^{T_k}F_Z > f$ [N]
- or task time $> t$ [s]

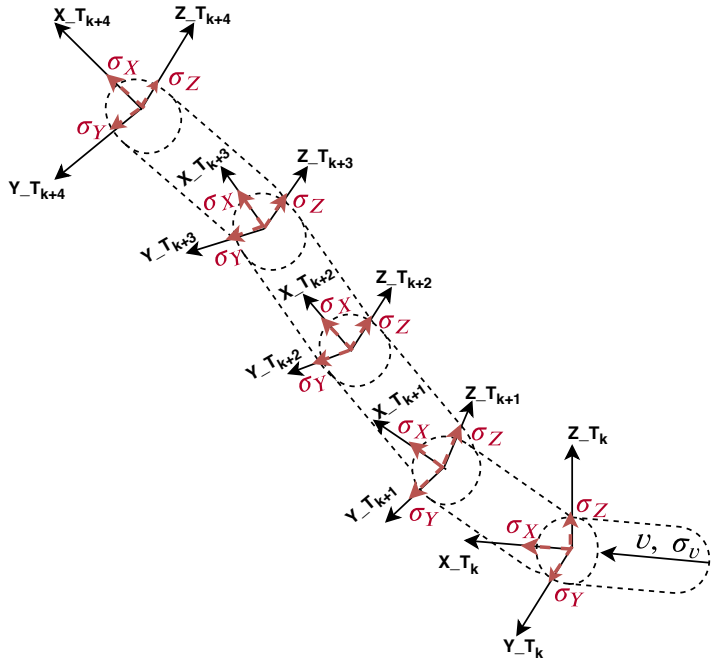

Figure 6.7: Illustration of an example path, defined for realizing the task in the second use case. Each waypoint in the user-defined Cartesian path has associated its own *task frame*. Here, T_k stands for the task frame associated with *k-th* waypoint in the defined path.

Here, T_k stands for the task frame associated with *k-th* waypoint in the defined path. A simple illustration of an example path, defined (in advance) for realizing the task specified in this use case, is presented in figure 6.7. In this example, the same tube parameters are specified for each section of the path, i.e. parameters such as position-tolerances σ_y and σ_z, tube-cruising velocity v and velocity-tolerance σ_v.

Control Architecture

A control architecture that enables execution of the task defined under this use case is presented in figure 6.8. All components constituting the control loop defined for the previous use case, except the *finite state machine* (FSM), are reused in this architecture without any modifications. More details about modifications in FSM are presented in the following.

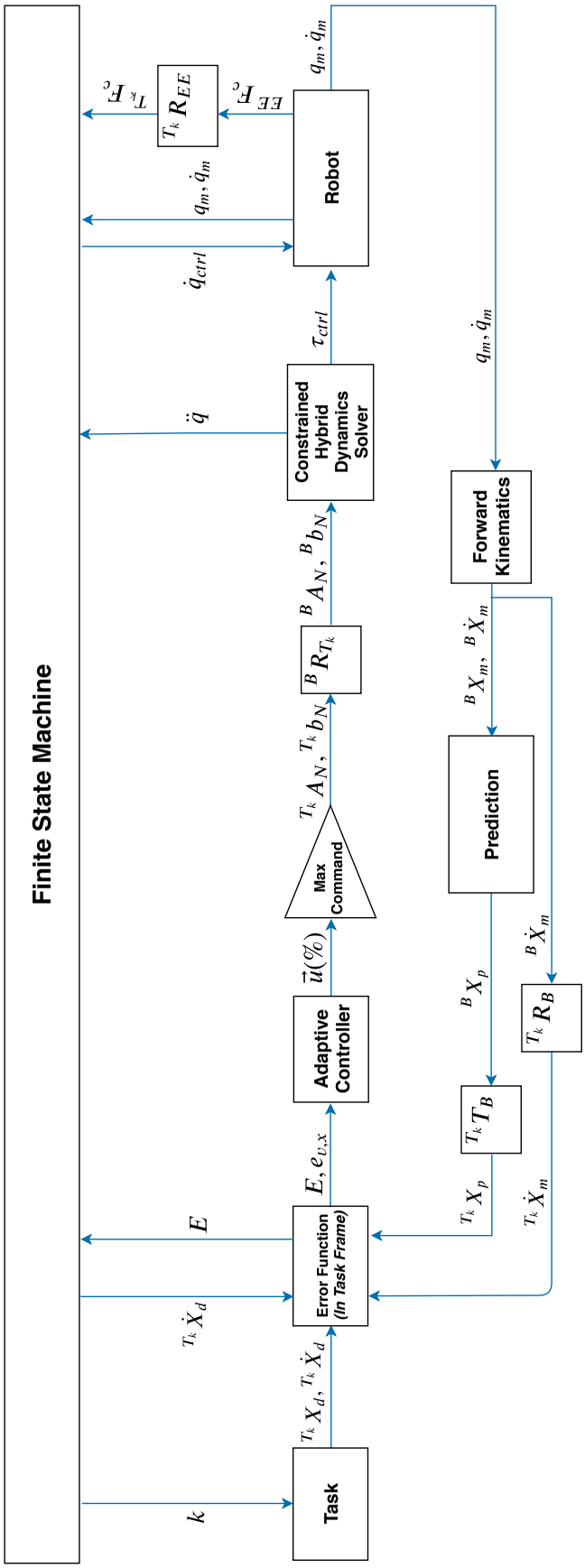

Figure 6.8: Main control loop for the second use case.

Finite-State Machine (FSM): As previously mentioned, the *finite state machine* component provides functionalities that, at runtime, monitor execution of the specified task, i.e. judge if the task constraints are satisfied w.r.t. defined tolerances [15]. However, since the task in this use case additionally defines a set of pre-defined Cartesian waypoints to be followed, the FSM is modified to additionally monitor the path traversal. For that reason, two new control states are added in this FSM logic, namely *FOLLOW_PATH* and *TRANSITION_PATH_SECTION* states (see figure 6.9). The former state defines a motion behaviour where the robot is following the predefined path while cruising through tubular areas (see figure 6.7), i.e. moving towards waypoints while keeping itself within the task-specified tube bounds. More specifically, this state encapsulates two sub-states, namely the *CRUISE_THROUGH_TUBE* state (previously defined in section 6.1.2) and the above-mentioned *TRANSITION_PATH_SECTION* state. Here, the latter sub-state does not define a different motion behaviour, but it defines transitioning of the path section through which the robot is currently *cruising*. More specifically, it defines that once the robot reaches the current sub-goal area in the path, the current instance of the task specification, that is used in the control loop computations, is replaced with an instance that corresponds to the next waypoint in the path. Additionally, this change takes effect in the next iteration of the control loop, such that the current computations are not affected by the change of the *task frame* and possibly task parameters. However, it is necessary to mention that the above-described policy for transitioning between two path sections is explicitly chosen in this work. Other policies and extra constraints in the task specification model can be derived for the aforementioned state, but development of these features is out of this work's scope.

The switching between the aforementioned *CRUISE_THROUGH_TUBE* and *TRANSITION_PATH_SECTION* states in FSM, is triggered by the following events [88]:

- *e_section_covered*: defines that the system has covered the current path section.
- *e_section_changed*: defines that the current instance of the task specification is replaced with the next instance.

An additional event that is introduced in this FSM is the *e_final_goal_area* event. It defines that the robot has covered all path sections, i.e. it has reached the final goal area and it is within the tube bounds. Other control states and their respective logic considered in this FSM, namely *START-TO-CRUISE* and *CRUISE_THROUGH_TUBE* states, are reused from the FSM that was developed for the first use case.

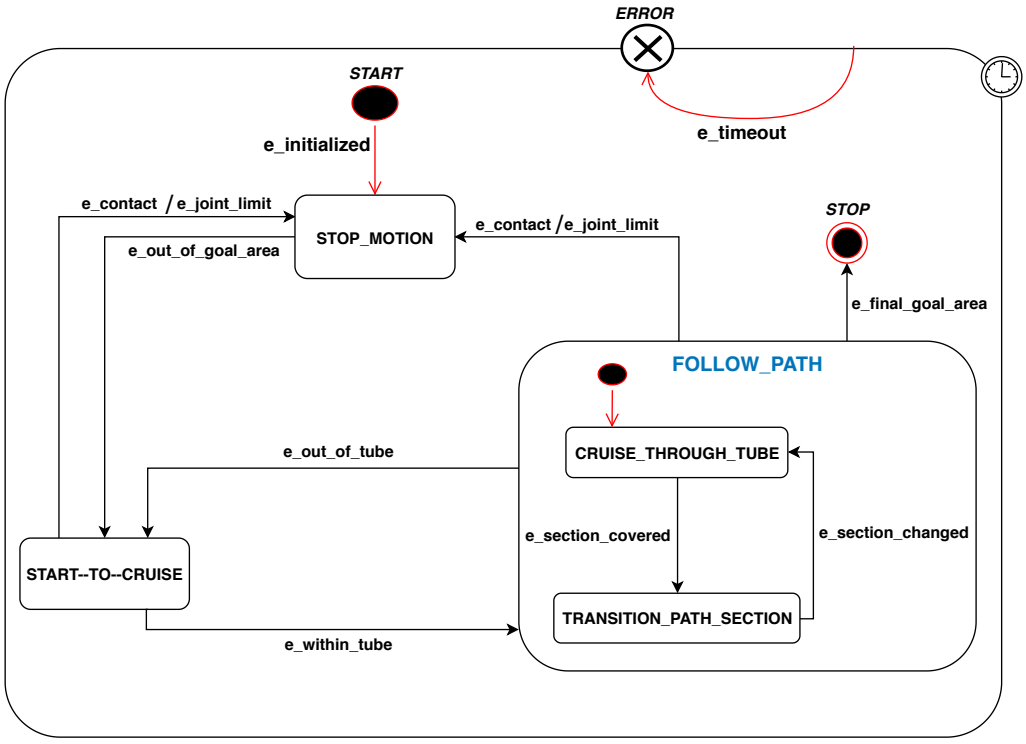

Figure 6.9: Finite state machine developed for the task of *following a pre-defined path*. Figure based on [88].

6.3 Cleaning a Surface

6.3.1 Strategy

For solving the control problem presented in the third use case (section 1.2), i.e. enabling explicit control of robot's Cartesian forces together with the control of its motion variables, current control strategies, such as, for example, [66], [67], are designed to maintain predefined force *setpoints*. However, in this way, the approaches are over-constraining robot control, which is not necessary for most applications.

The knowledge about the task defined under this use case allows for possible *relaxations* in solving this control problem. Here, the proposed *relaxation* defines that in directions in which the contact force is desired, the controller is not constrained to maintain a force setpoint, but rather control the force to remain within specified tolerances. Furthermore, the idea is to extend the strategy defined for the previous use cases with the above-mentioned force control relaxation. More specifically, for realizing the task of *cleaning a surface* and meeting the defined challenges in this use case, the idea is to exploit the already existing feature of the Popov-Vereshchagin algorithm, i.e. the ability for specifying task drivers via different interfaces. Here, the *artificial external force* interface can be exploited for specifying control commands in directions along which contact with the environment is maintained. On the other side, as in the previous use cases, the *Cartesian acceleration constraint* interface can be used for specifying control commands in directions along which the robot is required to perform a motion.

6.3.2 Application

Task specification

A task specification formalism that models the motions and forces necessary for the execution of *cleaning a surface* task, is developed by extending the work conducted for the previous two use cases. The following extension is motivated by the work presented in [81] and [50]. To model the contact situation in this uses case, an additional orthogonal *task frame* (T^F) is introduced [81] and three of its directions will be used for specifying instantaneous force requirements.

In this task specification model, motion-related terms are specified with respect to *motion task frame* T_k^M, while force-related terms are specified with respect to the *force task frame* T^F. More specifically, similarly to the previous two use cases, the velocity, position and orientation terms are specified with respect to linear X, linear Y and angular Z directions of the T_k^M frame, respectively. On the other side, the desired normal force F_z and its associated tolerance $\sigma_{F,z}$ are specified with respect to contact normal, i.e. linear Z direction, while the reaction moments F_{ax} and F_{ay}, and their associated tolerances $\sigma_{F,ax}$ and $\sigma_{F,ay}$ are specified with respect to angular X and angular Y directions of the T^F frame, respectively [81].

An example of the task specification created for the third use case is presented in the following:

Task specification for the third use case: Example 1

move compliantly {

- ${}^{T_k^M}X$: velocity v [m/s], velocity-tolerance σ_v [m/s]
- ${}^{T_k^M}Y$: 0 [m], position-tube σ_y [m]
- ${}^{T^F}Z$: force F_z [N], force-tolerance $\sigma_{F,z}$ [N]
- ${}^{T^F}aX$: 0 [Nm], moment-tolerance $\sigma_{F,ax}$ [Nm]
- ${}^{T^F}aY$: 0 [Nm], moment-tolerance $\sigma_{F,ay}$ [Nm]
- ${}^{T_k^M}aZ$: no specification

} until either:

- goal area reached, tolerance σ_x [m]
- or ${}^{T^F}F_X$ / ${}^{T^F}F_Y > f$ [N]
- or task time $> t$ [s]

Here, a *motion task frame* is fixed at a location that is defined by the task programmer, and its position and orientation do not change throughout the task execution. However, if the task also consists of a pre-defined Cartesian path (as in the previous use case), and not only of a single goal state, each waypoint k has associated its own *motion task frame* T_k^M. Nevertheless, in this use case, only one *force task frame* T^F is defined.

The necessity for including an additional *task frame* for the specification of force-related variables, comes from the fact that contact geometry may not be constant, throughout the task execution. In other words, for correctly executing this task, the robot is required to follow a surface of unknown shape, with its tool segment or manipulated object. For the aforementioned reasons, it is necessary to enable continuous adaptation of the frame's orientation and position during the motion, such that, the frame maintains the constant relative orientation and position w.r.t. the contact [81]. In this way, the task directions of the *force task frame* remain unchanged throughout the task execution. Thus, the specification of control values associated with these directions can be kept constant in run-time. This adaptation requirement can be (as suggested in [81]) fulfilled by attaching the origin of *force task frame* T^F in the bottom center of the manipulated object, which is in contact with the surface, and letting it remain fixed to that location during the object's motion. Additionally, to maintain proper alignment of the manipulated object with the surface, it is necessary to regulate reaction moments generated about X and Y axes of this frame and in turn, keep the Z-axis pointing in direction of the contact normal [81]. For that reason, in the above-presented task specification, moment-tolerances are defined with respect to zero values.

The above-described task specification is designed for the robot system that is equipped with a *planar* object for cleaning a smooth surface. Nevertheless, other shapes of the manipulated object can also be included in this strategy by only modifying the task specification part that considers reaction moments. For more details on these modifications, we refer the reader to [81].

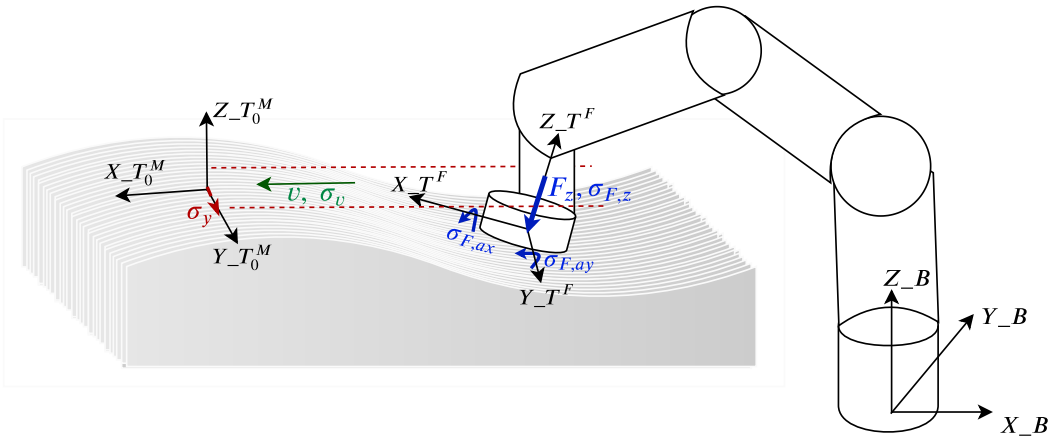

Figure 6.10: Illustration of the robot action performed in the third use case. In this example, only one goal area is defined. Nevertheless, if required, the task specification can be extended with a pre-defined Cartesian path, in the same manner as in the second use case.

A simple illustration of the robot action that realizes the task specified in this use case, is presented in figure 6.10. In this example, only one goal area is defined. Nevertheless, if required, the task specification can be extended with a pre-defined Cartesian path, in the same manner as in the second use case.

By having task constraints specified w.r.t. two different orthogonal frames, i.e. motion-related constraints specified in a frame that is fixed to the environment and force-related constraints specified in a moving frame, a situation in which these constraints conflict may occur during the task execution. The conflict between motion and force constraints or, in other words, the coupling between their associated task directions, is addressed by introducing explicit prioritization between these two types of constraints. More specifically, the Popov-Vereshchagin hybrid dynamics solver allows us to define a prioritization between the *external force* and *acceleration constraint* drivers [3], [5]. As described in section 6.3.1, the strategy developed for this use case defines that the solver's *acceleration constraint* interface will be used for specifying motion control commands, while the *external force* interface will be used for specifying force control commands. In the formulation of the Popov-Vereshchagin algorithm that is presented in chapter 4, the acceleration constraints are prioritized over the external force driver. Therefore, in this control strategy, the motion-related constraints will have higher priority over the force-related constraints. However, in [5], an approach for changing this prioritization, such that the higher priority is given to the *external force* driver, is presented. Nevertheless, this change of prioritization will not be considered in this work.

Another example of the task specification created for this use case is presented in the following:

Task specification for the third use case: Example 2

move compliantly {

- ${}^{T_k^M}X$: velocity v [m/s], velocity-tolerance σ_v [m/s]
- ${}^{T_k^M}Y$: 0 [m], position-tube σ_y [m]
- ${}^{T^F}Z$: force F_z [N], force-tolerance $\sigma_{F,z}$ [N]
- ${}^{T^F}aX$: 0 [Nm], force-tolerance $\sigma_{F,ax}$ [Nm]
- ${}^{T^F}aY$: 0 [Nm], force-tolerance $\sigma_{F,ay}$ [Nm]
- ${}^{T_k^M}aZ$: angle φ_{az} [rad], orientation tube σ_{az} [rad]

} until either:

- goal area reached, tolerance σ_x [m]
- or ${}^{T^F}F_X$ / ${}^{T^F}F_Y > f$ [N]
- or task time $> t$ [s]

Control Architecture

A control architecture that enables execution of the task defined under this use case is presented in figure 6.11. In addition to the robot states that constitute the control feedback used in the previous two use cases, here, the control feedback also consists of the measured reaction forces which act on the manipulated object. These wrench measurements are directly used for calculating contact force deviations from the task-defined bounds. In this control loop, the complete task error defines: *i*) a 3×1 vector E associated with robot pose deviations in linear X, linear Y and angular Z directions, *ii*) a scalar value $e_{x,v}$ associated with robot's tube speed deviation in linear X direction and *iii*) a 3×1 vector E_F associated with robot's contact force deviations in linear Z, angular X and angular Y directions. The error function graph presented in figure 6.12 depicts the calculations of the aforementioned error quantities. Every DOF defined in the task specification has its associated error element in the aforementioned task error. However, the computed robot position error along the linear X direction, will not be used in the following computations of the adaptive controller. Nevertheless, this error element will be used for monitoring purposes by the FSM component, which will be presented in the following.

The above-mentioned error quantities are passed as input to an adaptive controller that consists of six independent (decoupled) ABAG controllers. This component is presented in figure 6.13. Its output is represented by a 6×1 vector of Cartesian control commands $\vec{u}$, defined in percentages. However, as previously mentioned in section 6.1.2, more concrete meaning to these control commands is given outside the controller's computations. For that reason, each motion-related

element of the vector $\vec{u}$ is scaled with its corresponding element in a 3×1 vector of the maximum allowed *acceleration energy* setpoints. On the other side, each force-related element of the vector $\vec{u}$ is scaled with its corresponding element in a 3×1 vector of the maximum allowed *artificial external force* setpoints. Results of this operation are: *i*) the *acceleration energy* setpoints b_N expressed w.r.t. the *motion task frame* T_k^M and *ii*) the *artificial external force* setpoints $F^{ext-artificial}$ expressed w.r.t. the *force task frame* T^F. However, the next component in this control loop, namely the Popov-Vereshchagin solver, expects that the Cartesian space task drivers are expressed w.r.t. the robot *base frame*. For that reason, is it necessary to perform a coordinate transformation of these values. Here, the *acceleration constraint* motion driver is transformed in the same manner as in the previous two use cases (see section 6.1.2). However, the *artificial external force* driver is transformed using rotation matrix ${}^B R_{T^F}$, which represents relative orientation of the *force task frame* T^F w.r.t robot's *base frame* B.

Other components that constitute this control architecture, such as, the constrained hybrid dynamics solver and the prediction component, are reused from the control architectures defined in the previous two use cases, without modifications. However, the *finite state machine* (FSM) is modified for this use case, and more details about these modifications are presented in the following.

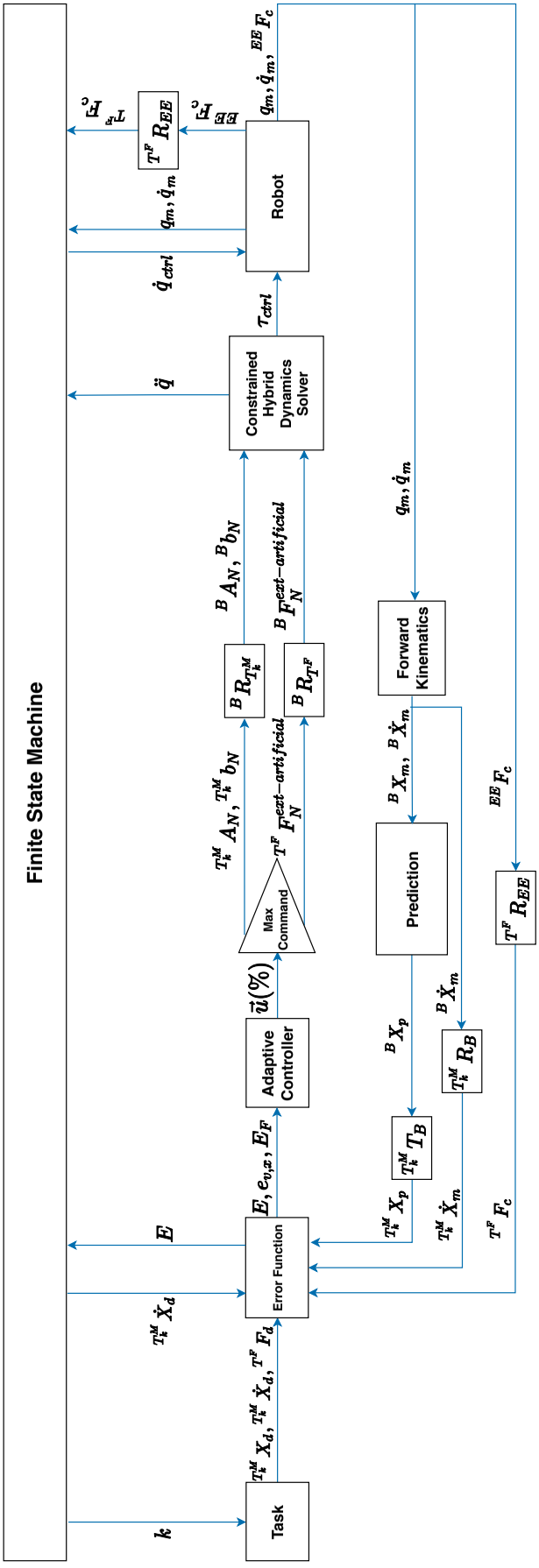

Figure 6.11: Main control loop for the third use case.

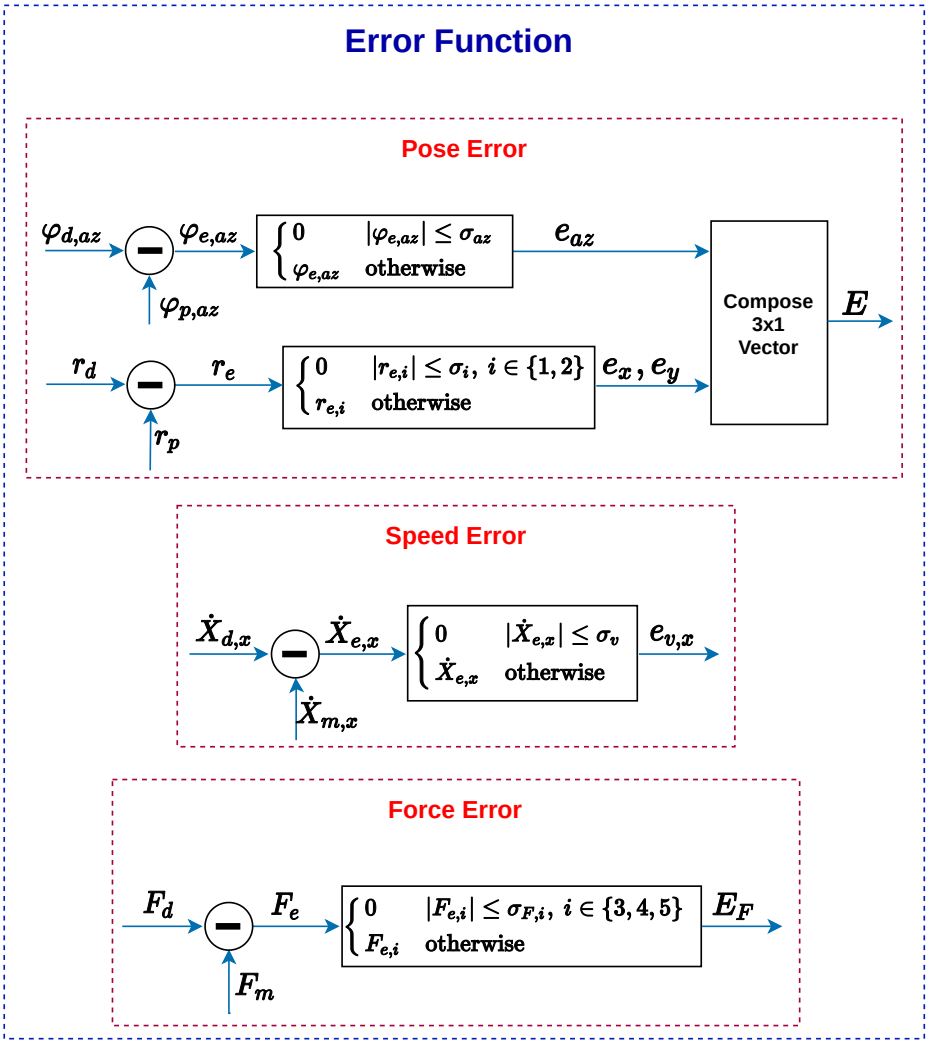

Figure 6.12: Error function for the third use case. The complete task error defines: *i*) a 3×1 vector E associated with robot pose deviations in linear X, linear Y and angular Z directions, *ii*) a scalar value $e_{x,v}$ associated with robot's tube speed deviations in linear X direction and *iii*) a 3×1 vector E_F associated with robot's contact force deviations in linear Z, angular X and angular Y directions.

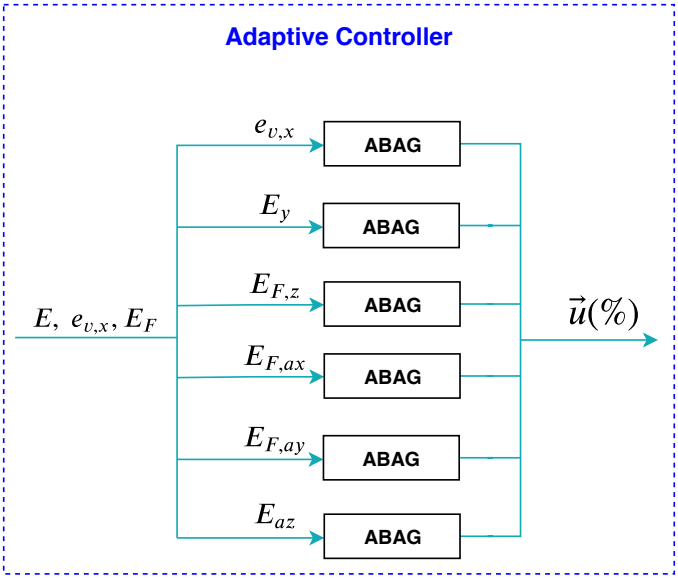

Figure 6.13: Adaptive controller for the third use case. In the linear X direction, tube-speed is controlled. In the linear Y direction robot deviation from the position-tube is controlled. In the linear Z, angular X and angular Y directions, contact forces are regulated. Finally, in the angular Z direction, robot deviation from the orientation-tube is controlled.

Finite-State Machine (FSM): Here, for the task of *cleaning a surface*, the FSM is created by reusing and extending control states of the FSMs that were developed for the first and second use case. More specifically, *STOP_MOTION* and *TRANSITION_PATH_SECTION* states are reused without modifications, while the *START-TO-CRUISE* and *CRUISE_THROUGH_TUBE* states are reused and extended. Additionally, two new control states, *APPROACH_SURFACE* and *ALIGN_WITH_SURFACE*, are introduced and they are motivated by work presented in [88]. A graphical representation of the FSM created for this uses, is illustrated in figure 6.14.

In this use case, for the *START-TO-CRUISE* control state, an extended version of the motion behaviour considered in the first use case (see section 6.1.2), is defined. More specifically, in this control state, while the robot is moving towards the task-specified tubular region and maintaining the zero tube-speed, the robot is also following a surface of unknown shape and applying the desired normal force, as defined in the above-presented task specification model. Furthermore, the motion behaviour that is previously considered in the *CRUISE_THROUGH_TUBE* state, is also extended in the same way. More specifically, in this control state, while the robot is *cruising* through the tubular area, i.e. moving towards the goal state and keeping itself within the task-specified tube bounds, the robot is also following a surface of unknown shape and applying the desired normal force, as defined in the above-presented task specification.

The newly introduced control states, i.e. the *APPROACH_SURFACE* and *ALIGN_WITH_SURFACE* states, define motion behaviours where the robot is approaching towards and aligning with the surface, respectively. More specifically, in the former state, the robot action is characterized by a *guarded motion* [81], [89], i.e. the robot moves towards the surface and it stops when the sensors detect a contact force. This type of robot action can be specified using the same task specification model that is introduced for the first use case (see section 6.1.2). On the other side, the latter state is defined by the behaviour where the robot exerts a certain force against the surface and at the same time, opposes the reaction moments generated on the manipulated object while keeping the zero tangential velocities. The task specification model for this robot action is proposed and presented in [81].

Many of the events defined in the FSMs from previous two use cases are reused here. However, several new events that trigger the connecting transitions between the states in this use case are introduced and they are the following [88]:

- *e_no_contact*: defines that the robot's sensors have not sensed a contact with the environment, in any of the linear X, Y, Z directions.
- *e_contact_x_y*: defines that the robot's sensors have sensed a contact with the environment, in the linear X or linear Y directions.
- *e_contact_z*: defines that the robot's sensors have sensed a contact with the environment, in the linear Z direction.
- *e_align_done*: defines that the robot has maintained the above-described alignment for a pre-defined time period [88].

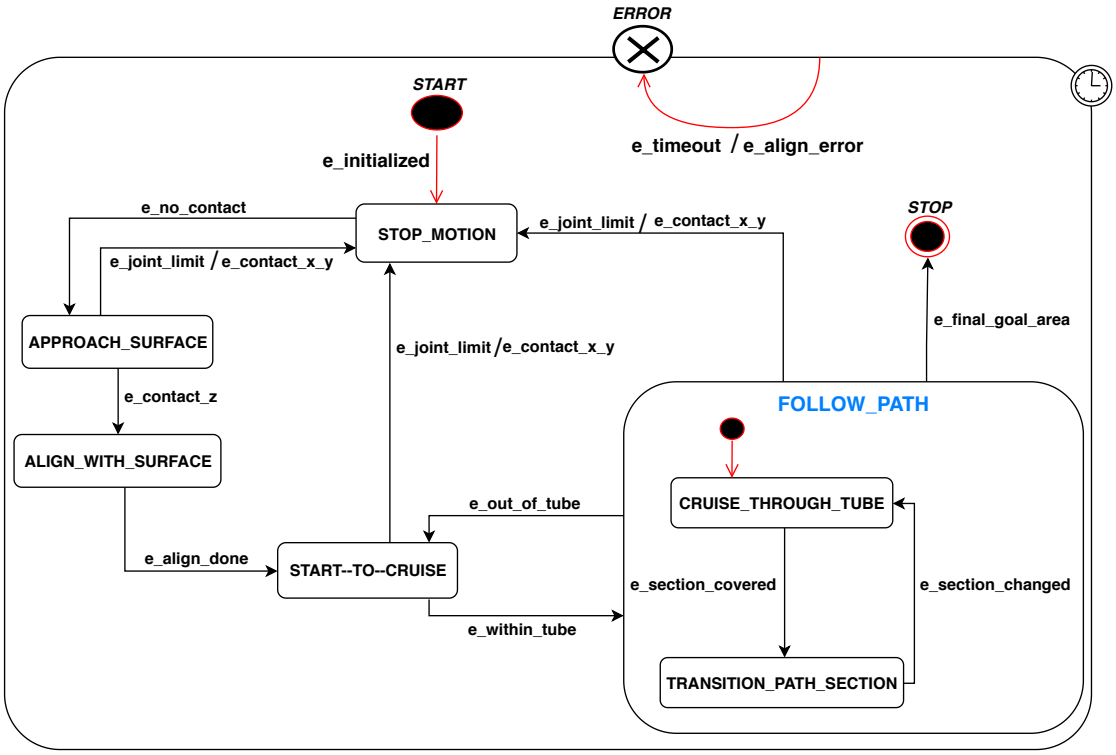

Figure 6.14: Finite state machine developed for the task of *cleaning a surface*. Figure based on [88].

- *e_ align_ error*: defines that the robot failed to regulate the task-defined contact linear force and moments [88].

6.4 Transporting an Unfamiliar Package

6.4.1 Strategy

To meet the challenge of this use case, the derived control approach should consist of a method for compensating the effects that a grasped (unfamiliar) object, i.e. an unknown load on the robot's end-effector, causes. More specifically, the control architecture should update the system's model parameters and control inputs accordingly, in order to enable a sufficiently accurate and stable motion execution. However, here, the idea is to extend the control strategy defined for the first use case (see section 6.1.1) and introduce the following control relaxation. To relax the control for this type of task, the objective here is to take advantage of the already existing computations performed in the control loop, to infer the necessary information about the grasped object. More specifically, for deriving a solution to this problem, features of the already considered ABAG controller and the Popov-Vereshchagin hybrid dynamics solver can be exploited.

As presented in section 5.1, the ABAG algorithm is using feedback information to approximate the unknown (non-modelled) dynamics parameters (inertia-mass, damping, elasticity) of the controlled system [6]. More specifically, the information about approximated dynamics is resembled by the controller's *bias* term, which is constantly adapting its value by following the trend in error. Furthermore, as the *bias* term directly contributes to the ABAG's final control command, this controller, in turn, compensates for the effects of the aforementioned disturbances. However, in this uses case, the idea is to exploit this already available (computed) information about the non-modelled dynamics, in order to compensate for the unknown load in a more systematic way. Namely, we can update and improve the system's model used in the computations of instantaneous robot dynamics and in this way, take the object's load into account explicitly. In other words, with this procedure, we can improve the *feedforward* part of the developed control architecture.

The Popov-Vereshchagin algorithm allows us to update the system's model in two ways, namely *i*) by directly alternating mass-inertia parameters of the kinematic chain or *ii*) by specifying a *physical* Cartesian force on the end-effector via solver's *external force* interface (see section 4.2).

6.4.2 Application

Task specification

For specifying the necessary motion in the task of transporting a package, the already developed task specification model for the first use case (see section 6.1.2) will be reused here.

Control Architecture

A control architecture that enables execution of the task defined under this use case is presented in figure 6.15. All components that constitute the control loop defined in the first use case, are reused in this architecture without modifications.

However, here, an additional component is added, namely, an estimator unit that serves for estimating an unknown model of the grasped object. The graphical representation of this component is presented in figure 6.16. Here, the first part monitors (over a particular *time window*) ABAG's *gain* and *bias* signals ($\vec{g}$, $\vec{b}$), in order to detect whether *i*) the *bias* signal has converged to a constant value (i.e. it approximated the non-modelled dynamics of the system) and *ii*) the *gain* signal has converged to the zero value. When such a situation occurs (i.e. the ABAG controllers have reached a steady-state), the monitor raises *e_signals_converged* event which triggers the second part of this component. Here, the current estimates of either mass-inertia parameters of the grasped object or, the force exerted on the end-effector by this object are updated. In figure 6.16, terms $\hat{m}_O$, $\hat{r}_O^{com}$ and $\hat{I}_O^{rot}$ represent the estimated object's mass, center of mass and rotational inertia, respectively, while term ${}^B\hat{F}_N^{ext-physical}$ represents the estimated spatial force on the end-effector. Here, term $\vec{b}_{reference}$ represents the vector of steady-state *bias* values that correspond to a situation in which the robot has performed, in advance, a specific type of motion *without* holding the object in its hand. The purpose of this term will be described with more details, in the following.

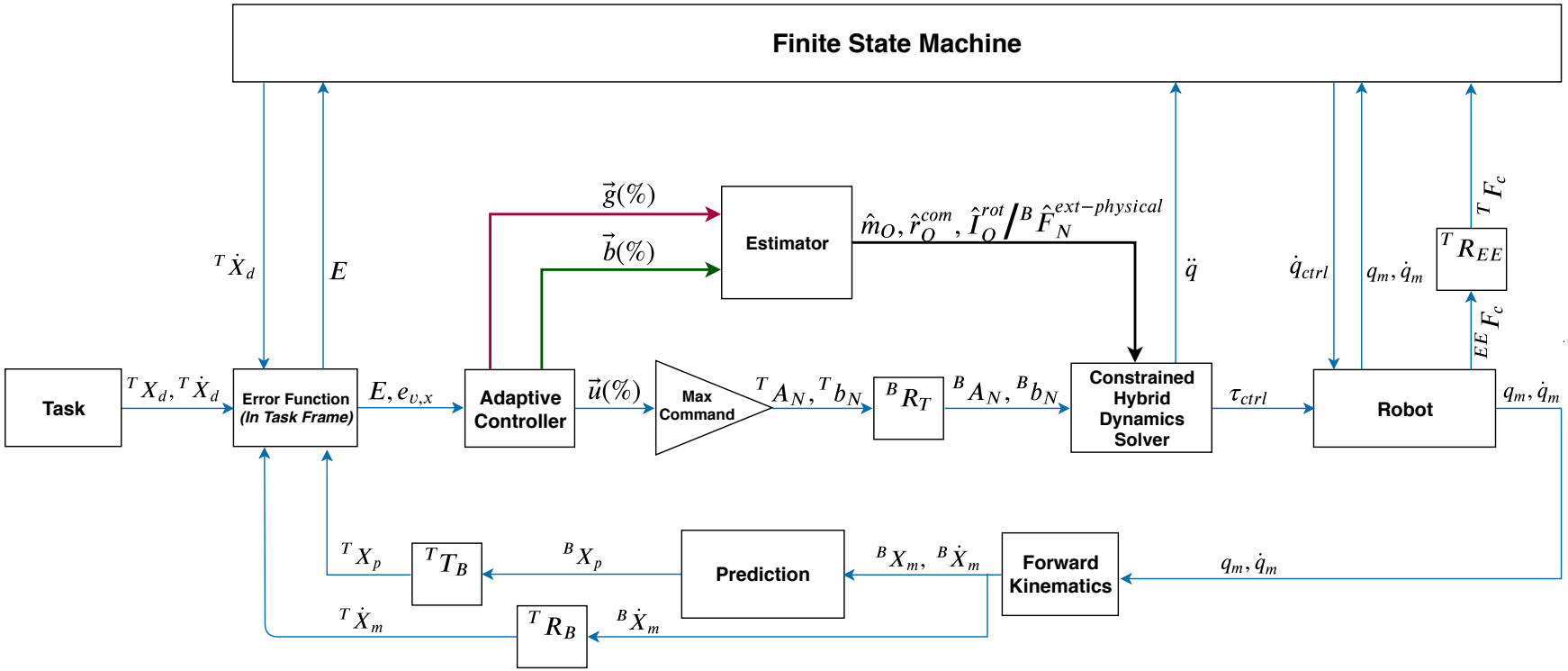

Figure 6.15: Main control loop for the fourth use case.

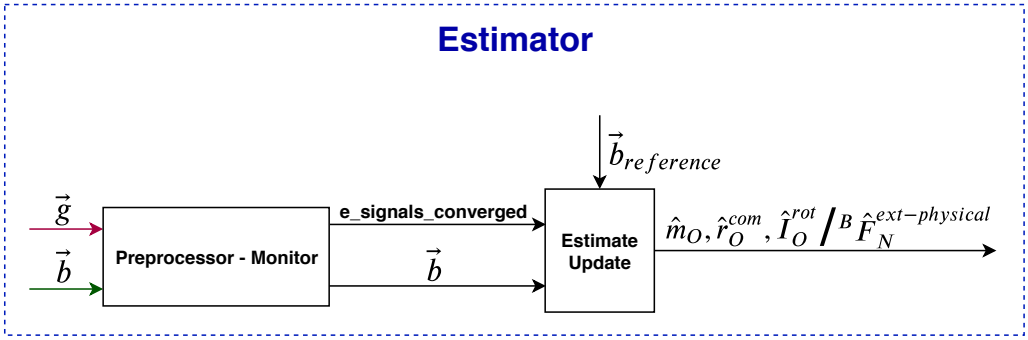

Figure 6.16: Estimator component for the task of *transporting an unfamiliar package*.

Excitation of Parameters

When a robot carries an object, the load impacts different aspects of the robot's dynamics [14], [19]. More specifically, the impact originating from the object's mass can be directly observed in *i*) the gravitational force that constantly acts on the robot system and *ii*) the inertial force that opposes the robot system from accelerating in a linear direction, i.e. resist a change in the object's linear motion. However, as the above-mentioned forces are considered to act on the object's *center of mass* (COM), the position of this point additionally influences the effects that these forces cause. On the other side, the impact originating from the object's rotational inertia can be observed in the inertial torque that opposes the robot system from accelerating in a angular direction, i.e. resist a change in the object's angular motion.

Mass and Center of Mass In order to estimate the unknown object's *mass* and *center of mass* parameters or, the effects that they cause, it is necessary to excite these parameters [14]. The excitation can be performed in two ways, passively and actively.

The passive excitation method exploits an already existing force in nature, i.e. the gravitational force that constantly acts on the grasped object. Namely, after lifting the object and before starting the transportation motion, the robot can be commanded to hold the object in a steady pose, i.e. maintain the zero end-effector/object's velocity. In this situation, due to not accurate model (unknown load), the computations of robot dynamics (i.e. the Popov-Vereshchagin solver) will not accurately consider the effects of the gravitational force that acts on the robot system. To compensate for these effects and maintain the zero end-effector/object's velocity, the ABAG controllers (see the graph in figure 6.4) will adapt magnitudes and signs of their respective *bias* terms, for all Cartesian DOFs in which this force creates an impact. Additionally, it is necessary to, beforehand, perform a *reference* trial with the robot, i.e. the same type of motion (maintaining the zero end-effector velocity, in the same joint configuration) but in this case, *without* the object in its hand. In this way, the ABAG controllers will adapt the *reference bias* signals that will be used in the following estimation procedure.

In the actual estimation procedure, once the ABAG controllers have reached a steady-state, for each DOF, the value of associated *bias* signal will be compared with the value of its counterpart $b_{reference}$, which was previously adapted. The *difference* between values of these two *bias* signal sets (i.e. those that correspond to *i*) the motion *with* the object and *ii*) motion *without* the object in the hand) will resemble the effects of unknown object's mass and center of mass. Thus, based on these observed differences, either the mass and COM ($\hat{m}_O$ and $\hat{r}_O^{com}$) parameters or, the specification of the spatial *physical* external force acting on the end-effector ($^B\hat{F}_N^{ext-physical}$), will be updated in the system's model accordingly. However, in order to complete the estimation, it is often necessary to perform this update procedure multiple times, until the current *bias* values ($\vec{b}$) converge to the reference *bias* values ($\vec{b}_{reference}$).

In the case where a user chooses the option of directly alternating *mass* and *center of mass* parameters in the model, instead of alternating the specification of $^B\hat{F}_N^{ext-physical}$ term, it necessary to distinguish (in the estimation procedure) between the effects of unknown *mass* and the effects of unknown *center of mass*. Here, the knowledge of how the gravitational force impacts objects with different center of mass positions will be exploited. Namely, if a current estimate of the COM position (which is used in the computations of robot dynamics) is not correct, the effect will be seen in the change of *bias* value for more then one DOF. In other words, a non-negligible difference between the current and *reference* bias will be observed in, not just the direction of gravitational force, but also in one or multiple other directions, e.g. about other two orthogonal axes.

In the above-described excitation method, the force that excites the *mass* and *center of mass* parameters or, the effects that they cause, already exist in nature. However, the excitation of those parameters or, the effects that they cause, can also be performed actively; by accelerating the object in a linear direction that is orthogonal to the direction of gravitational force. Here, the impact originating from the object's mass can be directly observed in the *inertial* force that opposes the system from accelerating in a linear direction, i.e. resist a change in the object's linear motion. Similarly to the previous method, the estimation procedure in this method starts after the grasped object is lifted. However, here, instead of commanding the robot to hold the object in a steady pose, the robot is commanded to move the object in a linear direction (that is orthogonal to the direction of gravitational force) with a *constant acceleration*. Due to inaccurate model, the computations of robot dynamics (i.e. the Popov-Vereshchagin solver) will not accurately consider the aforementioned *inertial* force that acts on the robot system. To compensate for these effects and maintain the constant linear acceleration, the ABAG controllers will adapt magnitudes and signs of their respective *bias* terms, for all DOFs in which the *inertial* force creates an impact. Here as well, it is necessary to (beforehand) perform a *reference* trial with the robot, i.e. the same type of motion but in this case, *without* the object in its hand; in order to produce the *reference bias* signals. The rest of the estimation process considered in this active excitation method is the same as in the previously described method.

Rotational Inertia Here, excitation of the object's *rotational inertia* will be performed in an active way, i.e. by actively accelerating the object in an angular direction. The impact originating from the object's rotational inertia can be observed in the *inertial* torque that resist a change in the object's angular motion. The estimation procedure in this method also starts after the grasped object is lifted. However, here, instead of commanding the robot to move the object in a linear direction, the robot is commanded to move the object in a angular direction, with a *constant acceleration*. Due to not accurate model, the computations of robot dynamics (i.e. the Popov-Vereshchagin solver) will not accurately consider the aforementioned *inertial* torque that acts on the robot system. To compensate for these effects and maintain the constant angular acceleration, the ABAG controllers will adapt magnitudes and signs of their respective *bias* terms, for all DOFs in which this *inertial* torque creates an impact. Here as well, it is necessary to (beforehand) perform a *reference* trial with the robot, i.e. the same type of angular motion but in this case, *without* the object in its hand; in order to produce the *reference bias* signals.

In the actual estimation procedure, the *difference* between values of these two *bias* signal sets (i.e. those that correspond to *i*) the motion *with* the object and *ii*) motion *without* the object in the hand) will resemble the effects of unknown object's *rotational inertia*. Thus, based on these observed differences, either the rotational inertia parameter ($\hat{I}_O^{rot}$) or, the specification of the spatial *physical* external force acting on the end-effector (${}^B\hat{F}_N^{ext-physical}$), will be updated in the model accordingly. However, in order to complete the estimation, it is often necessary to perform this update procedure multiple times, until the current *bias* values ($\vec{b}$) converge to the reference *bias* values ($\vec{b}_{reference}$).

7

Experimental Evaluation and Results

This chapter presents an experimental evaluation of the proposed strategies, i.e. evaluation of the task specification models and control architectures, for the first three uses cases considered in section 1.2. Here, the goal is to validate the derived control approach for each previously defined robot task, in a simulation environment and/or on a real robot platform.

For providing a simulation environment in which the following tests are conducted, an already existing software framework is used, i.e. the "OROCOS/ROS Components for Light Weight Robots" [90] framework, which combines the Gazebo, an open-source multi-robot simulator [91] and OROCOS Real-Time Toolkit [92]. In this framework, the simulated robot model is the 7-DOF KUKA Lightweight Robot 4 [93]. The reader should note that in this simulation framework, the friction in robot's joints and noise in robot's sensors are simulated based on the framework's internal (build-in) friction and noise models, which are not part of the robot's dynamics model. However, in the following simulation-based experiments, the robot dynamics model that is used in dynamics computations of the developed control architectures is the same as the robot dynamics model used by this simulation framework.

On the other side, the real robot platform that is used in this evaluation approach is the 5-DOF KUKA youBot [94]. However, for this platform, the robot model that is provided in the official youBot-Store documentation [95] is used in dynamics computations of the developed control architectures.

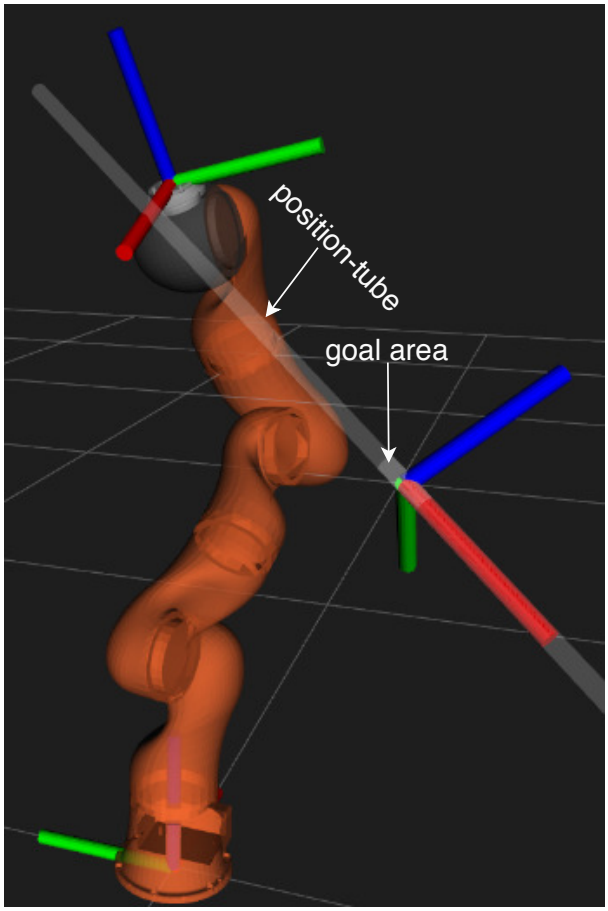

Figure 7.1: Visualization of the KUKA LWR 4 model and its initial configuration for the *pre-grasp motion* task execution, in the simulation environment. Here, the simulation software [90] has generated a virtual tube (based on the task-defined bonds), to enhance the visualization for this task execution. The virtual tube is depicted with gray color. For each frame presented in the figure, red color stands for X axis, green color for Y axis and blue color for Z axis.

7.1 Pre-Grasp Motion

7.1.1 Simulation Environment

Experimental Setup

In this experiment, *pre-grasp motion* task is performed in a simulation environment. Here, the robot is commanded to move towards a pre-defined goal area while keeping its end-effector within the task defined position-tube bounds. Additionally, the robot is constrained to maintain a predefined speed throughout the tube, until it reaches the goal area. However, the initial robot configuration is chosen in such a way that, at the start of the task execution, the end-effector's position is outside of the position-tube bounds. The experimental setup is presented in figure 7.1, while the task specification created for this test is presented in the following:

Parameter	**Value**
Initial joint configuration	2.967, 1.023, -0.131, 1.612, 0.221, 0.177, 0.015 [rad]
Task frame position	X: -0.200, Y: -0.308, Z: 0.632 [m]
Task frame orientation	$\mathrm{R} = \begin{bmatrix} -0.432 & 0.730 & 0.527 \\ -0.730 & 0.058 & -0.679 \\ -0.527 & -0.679 & 0.508 \end{bmatrix}$ [m]
S-Curve velocity profile	$0.05 + 0.12 \cdot \sin(r_x \cdot 5.0)$ [m/s]
Prediction horizon	2.5 s
Maximum commands	X: 60.0, Y: 60.0, Z: 60.0 $[\frac{Nm}{s^2}]$.
Control loop frequency	630 Hz
Gravity compensation	Disabled

Table 7.1: Main control parameters for the *pre-grasp motion* experiment in the simulation environment.

Task specification for the *pre-grasp motion* task: Simulation environment

move compliantly { // with task frame directions

- ${}^{T}X$: s-curve velocity profile [m/s], velocity-tolerance 0.005 [m/s]
- ${}^{T}Y$: 0 [m], position-tube 0.01 [m]
- ${}^{T}Z$: 0 [m], position-tube 0.01 [m]
- ${}^{T}aX$:, ${}^{T}aY$:, ${}^{T}aZ$: no specification

} until either:

- goal area reached, tolerance 0.03 [m]
- or ${}^{T}F_X$ / ${}^{T}F_Y$ / ${}^{T}F_Z > 0.5$ [N]
- or task time > 9 [s]

The main control parameters used in this experiment are summarized in table 7.1. Here, the position and orientation of the *task frame* are expressed with respect to the robot's base frame. However, the current robot's position in the X direction, represented by r_x term, is expressed with respect to the *task frame*. In this table, the *maximum commands* represent the maximum allowed acceleration energy setpoints for the robot's end-effector (for more details see section 6.1.2).

Results

The result of this experiment is depicted in figures 7.2, 7.3, 7.4, 7.5 and 7.6. In these control graphs, the reference and measured robot positions and velocity are all defined with respect to the *task frame* (T).

From the first four above-presented control graphs, we can observe three different control phases in this task execution. More specifically, since at the start of the task execution the end-effector's position is outside of the tube bounds, the finite state machine (FSM) that is part of the control architecture developed in section 6.1.2 starts the task execution in the *START-TO-CRUISE* state. Here, the robot is commanded to maintain the zero tube-velocity (in the linear X direction of the task frame), until it reaches the task-defined position-tube bounds. Once the robot is inside the tube, at a time around 4.5 s (see figures 7.3 and 7.4), the FSM makes the transition to the *CRUISE_ THROUGH_ TUBE* state. From this point on, the robot moves through the tube with the task defined velocity. However, once the robot reaches the goal area, at a time around 8.5 s (see figure 7.5), the FSM switches to the *STOP_ MOTION* state.

The graph presented in figure 7.2, reports the response of the main control loop for the linear X-direction, i.e. how well the control architecture developed in section 6.1.2 tracks the changing speed reference. Here, the first row depicts the measured and desired robot velocities, together with the upper and lower velocity-tolerances. The second row presents the computed velocity error, i.e. the difference between the measured and desired velocity values, at each time step (for more details on this error calculations, see figure 6.3). Last two rows in this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals, for this DOF.

The presented control graph shows that the developed control architecture has achieved sufficiently accurate and stable tracking of the changing velocity reference, with minor overshoots over the predefined velocity-tolerances. Additionally, the results show a very good response of the ABAG controller, even in the case of sudden changes in the reference values.

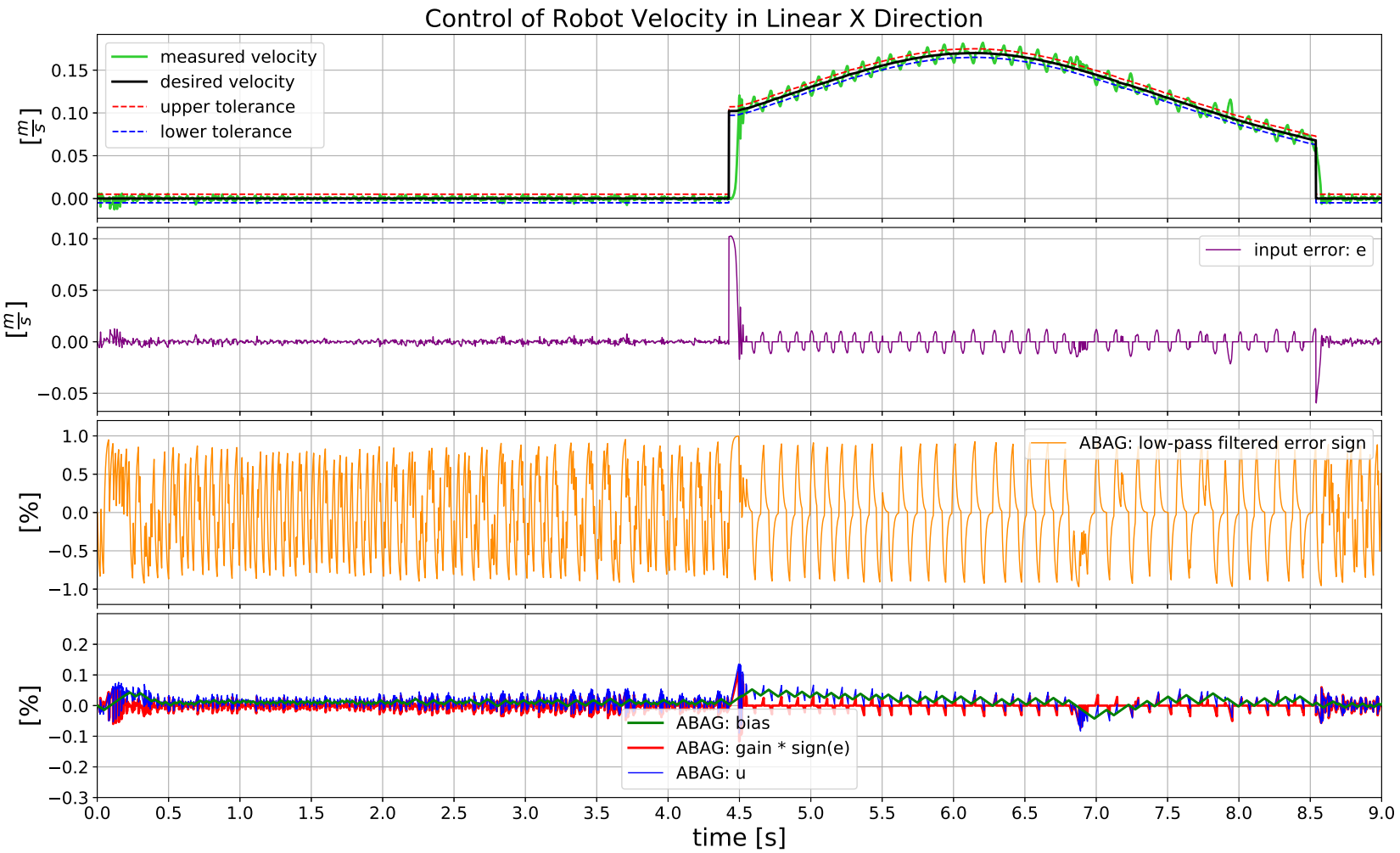

Figure 7.2: Response of the main control loop for the linear X-direction, during the *pre-grasp motion* task execution in the simulation environment. Here, the measured and reference velocity values are expressed with respect to the task frame (T). The notable time-points are at 4.5 s and 8.5 s.

The control graphs presented in figures 7.3 and 7.4, report the response of the main control loop for the linear Y and linear Z directions, respectively. More specifically, these control graphs show how well the developed control architecture keeps the robot's end-effector within the defined position-tube bounds. The first row, in both graphs, depicts the measured robot position together with the upper and lower position-tube tolerances. The second row, in both graphs, presents the computed position error which is, in this case, defined by the difference between the *predicted* and desired position values, at each time step (for more details on this error calculations, see section 6.1.2). The last two rows of these control graphs depict the behaviour of the ABAG controllers, i.e. their adapted signals.

The presented control graphs show that the developed control architecture has met the predefined thresholds and achieved stable tracking of the task-defined position tube, for both linear Y and linear Z directions, without any overshoots from these tight bounds.

In figure 7.6, the computed joint torque commands are depicted, together with the upper and lower saturation limits, for each robot's joint. These saturation limits are enforced in the dynamics computations of this control architecture, i.e. in line 26 of **Algorithm** 2. However, from this figure, we can observe that the robot has executed the task by using a very small percentage of the maximum joint torques. Furthermore, actuation in some of the joints, such as actuation in first, fifth and seventh, was not required at all, during the large part of the task execution. This behaviour occurs due to the fact that both, the ABAG controller (while computing the *non-instantaneous* control) and the Popov-Vereshchagin solver (while resolving the *instantaneous* dynamics) are exploiting the already existing forces in nature, such as, in this case, modelled gravitational and non-modelled joint friction forces, to achieve the desired robot states.

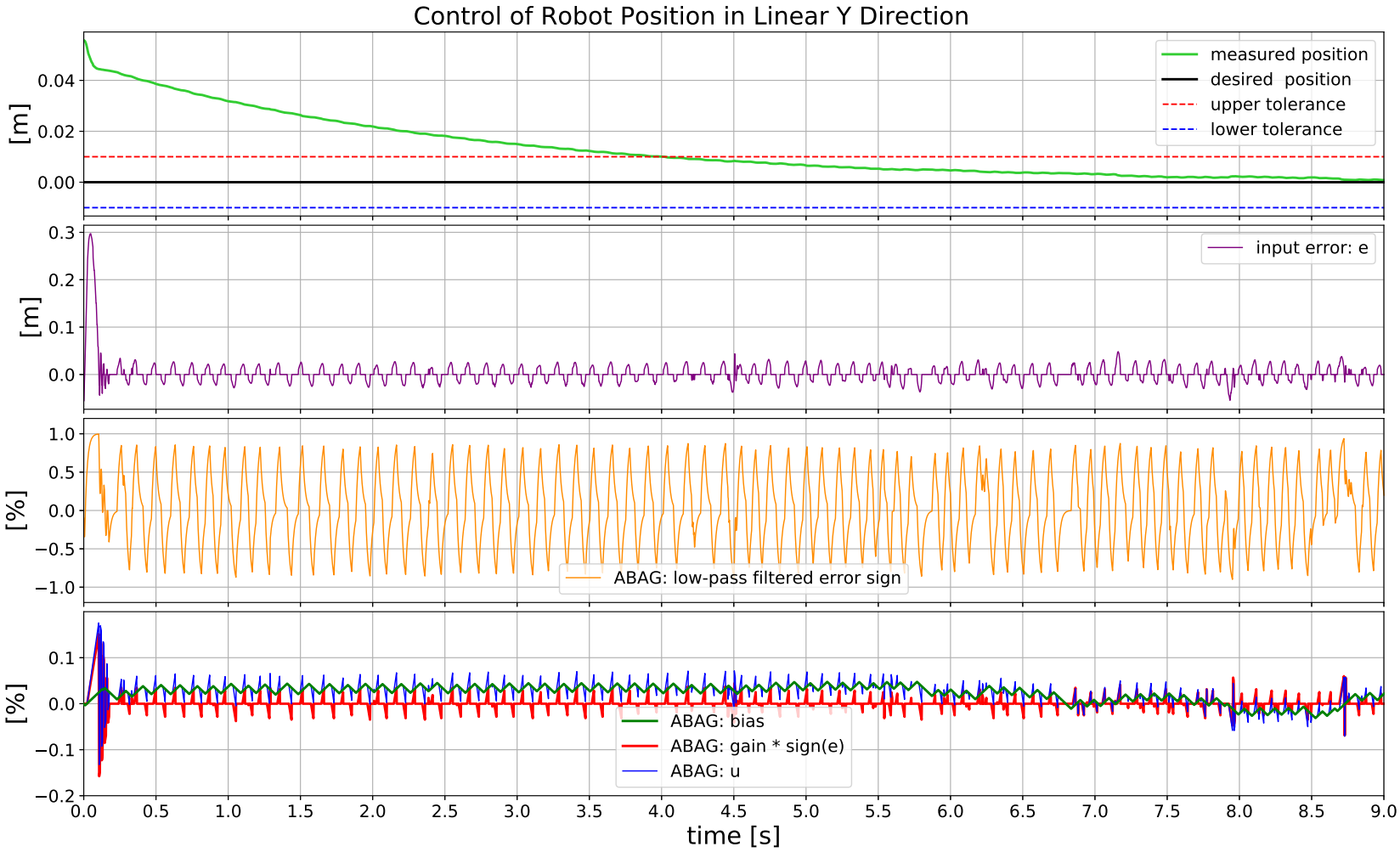

Figure 7.3: Response of the main control loop for the linear Y-direction, during the *pre-grasp motion* task execution in the simulation environment. Here, the measured and reference position values are expressed with respect to the task frame (T).

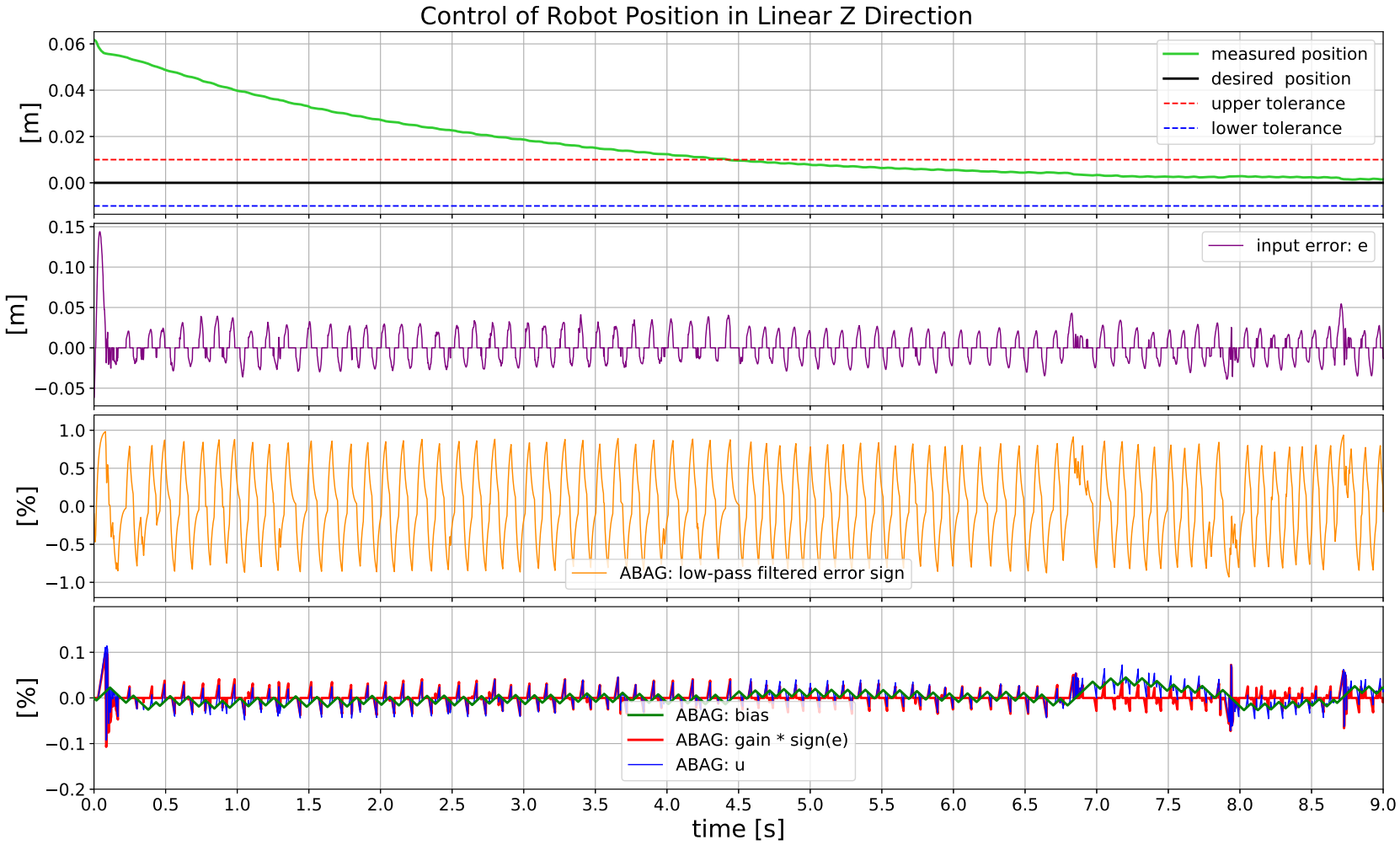

Figure 7.4: Response of the main control loop for the linear Z-direction, during the *pre-grasp motion* task execution in the simulation environment. Here, the measured and reference position values are expressed with respect to the task frame (T).

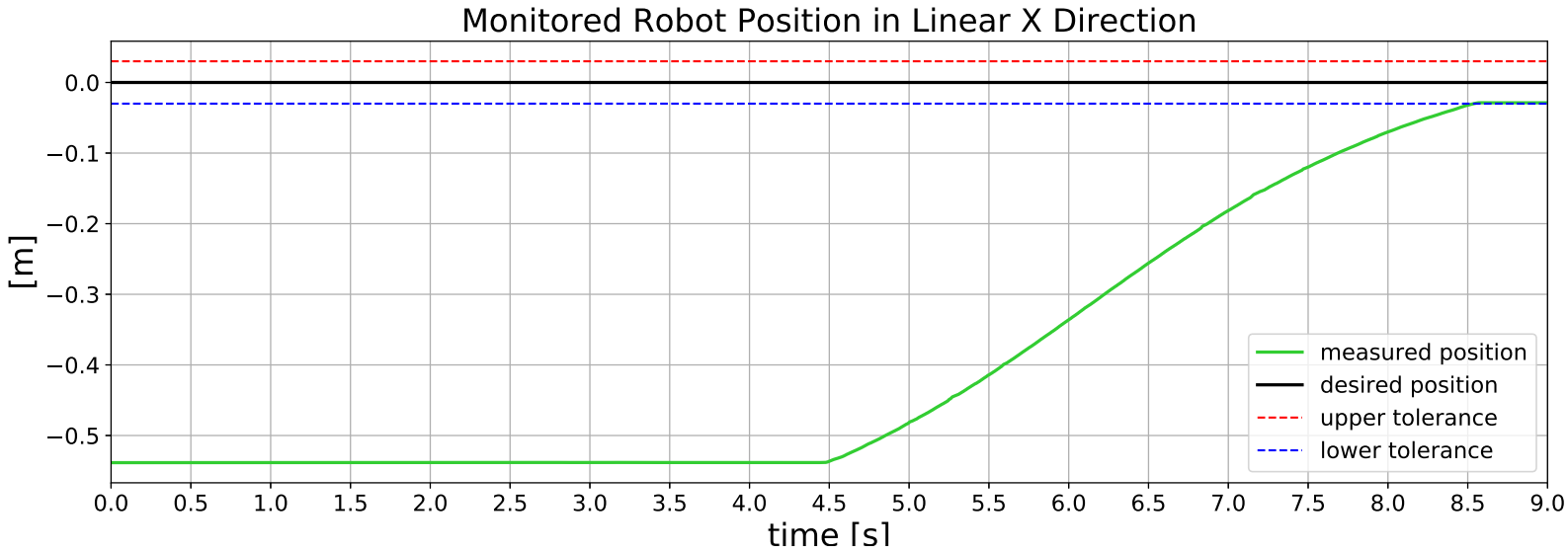

Figure 7.5: Resulting robot's position in the linear X direction, during the *pre-grasp motion* task execution in the simulation environment. Here, the measured position and goal-area values are expressed with respect to the task frame (T). The notable time-points are at 4.5 s and 8.5 s.

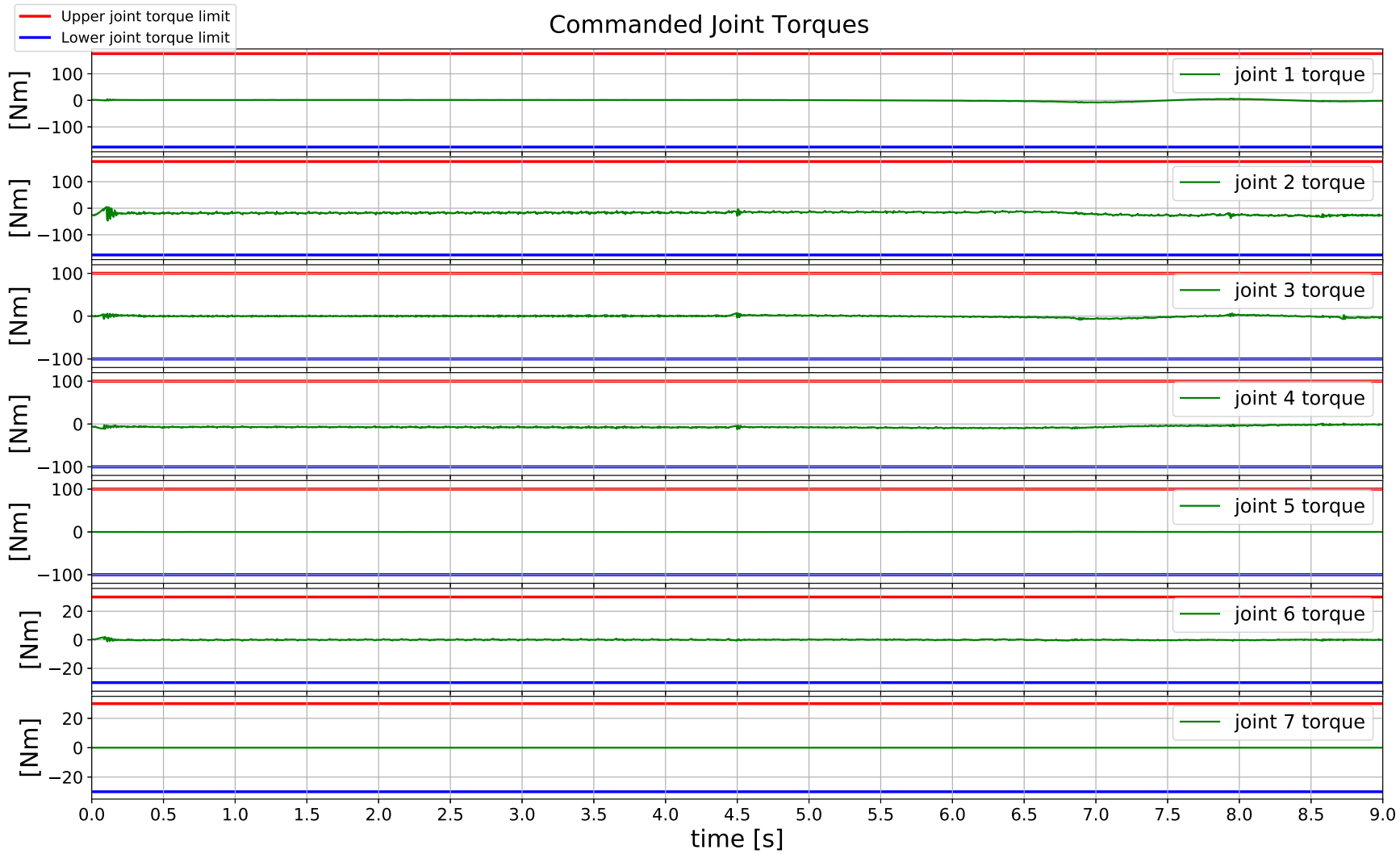

Figure 7.6: Computed joint torque commands, during the *pre-grasp motion* task execution in the simulation environment.

7.1.2 Real Robot Platform

Experimental Setup

In this experiment, *pre-grasp motion* task is performed using a real robot platform. Similarly to the previous experiment, the robot is commanded to move towards a pre-defined goal area while keeping its end-effector within the task defined position-tube bounds. Additionally, the robot is constrained to maintain a predefined speed throughout the tube, until it reaches the goal area. The initial robot configuration is chosen in such a way that, at the start of the task execution, the end-effector's position is outside of the position-tube bounds. However, in contrast to the previous experiment, here, the position and orientation of the *task frame* are chosen in such a way that, the robot is required to move in an upward direction, in order to reach the goal area. The reason for making such a setup is a necessity to evaluate performance of the control architecture developed in section 6.1.2 for the case when the gravitational force interfere (contribute negatively to) the task execution. The experimental setup is presented in figure 7.7, while the task specification created for this test is presented in the following:

Task specification for the *pre-grasp motion* task:
Real robot platform

move compliantly { // with task frame directions

- ${}^{T}X$: s-curve velocity profile [m/s], velocity-tolerance 0.003 [m/s]
- ${}^{T}Y$: 0 [m], position-tube 0.015 [m]
- ${}^{T}Z$: 0 [m], position-tube 0.015 [m]
- ${}^{T}aX$:, ${}^{T}aY$:, ${}^{T}aZ$: no specification

} until either:

- goal area reached, tolerance 0.02 [m]
- or task time > 9.5 [s]

The main control parameters used in this experiment are summarized in table 7.2. Here, the position and orientation of the *task frame* are expressed with respect to the robot's base frame. However, the current robot's position in the X direction, represented by r_x term, is expressed with respect to the *task frame*. In this table, the *maximum commands* represent the maximum allowed acceleration energy setpoints for the robot's end-effector (for more details see section 6.1.2).

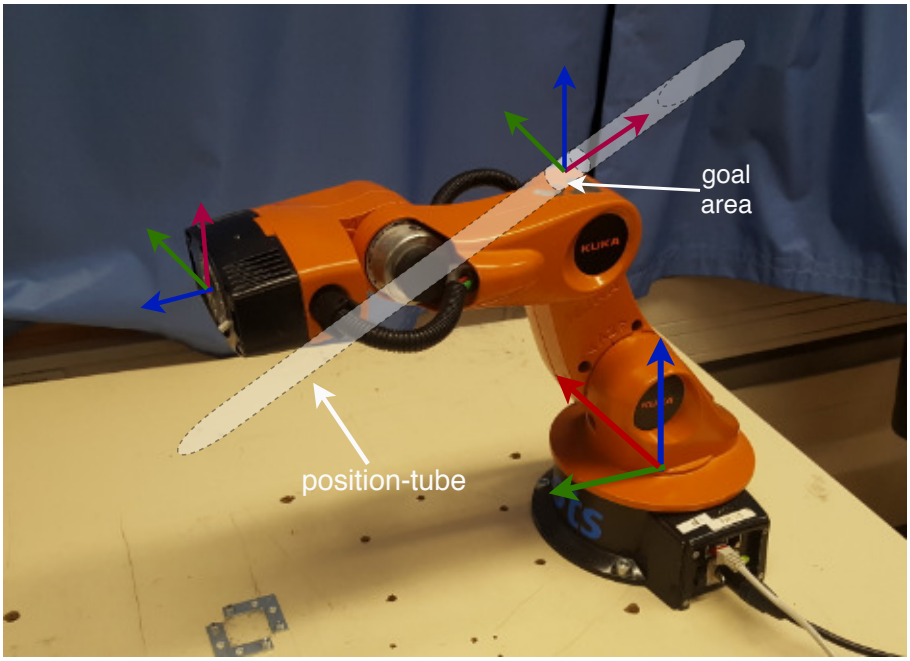

Figure 7.7: Visualization of the real 5-DOF KUKA youBot and its initial configuration for the *pre-grasp motion* task execution. Here, a virtual tube is manually added in the image, in order to enhance the visualization for this task execution. The virtual tube is depicted with gray color. For each frame presented in the figure, red color stands for X axis, green color for Y axis and blue color for Z axis.

Parameter	**Value**
Initial joint configuration	1.384, 1.597, -1.495, 1.925, 2.957 [rad]
Task frame position	X: 0.019, Y: 0.185, Z: 0.240 [m]
Task frame orientation	$\mathrm{R} = \begin{bmatrix} 0.0 & 0.914 & -0.404 \\ -0.914 & 0.163 & 0.369 \\ 0.404 & 0.369 & 0.836 \end{bmatrix}$ [m]
S-Curve velocity profile	$0.05 \cdot \sin(r_x \cdot 15.0)$ [m/s]
Prediction horizon	2.5 s
Maximum commands	X: 10.0, Y: 10.0, Z: 10.0 $[\frac{Nm}{s^2}]$.
Control loop frequency	660 Hz
Gravity compensation	Disabled

Table 7.2: Main control parameters for the *pre-grasp motion* experiment on the real robot platform.

Results

The result of this experiment is depicted in figures 7.8, 7.9, 7.10, 7.11 and 7.12. In these control graphs, the reference and measured robot positions and velocity are all defined with respect to the *task frame* (T).

From the first four above-presented control graphs, we can observe three different control phases in this task execution. More specifically, since at the start of the task execution, the end-effector's position is outside of the tube bounds, the finite state machine (FSM) that is part of the control architecture developed in section 6.1.2 starts the task execution in the *START-TO-CRUISE* state. Here, the robot is commanded to maintain the zero tube-velocity (in the linear X direction of the task frame), until it reaches the task-defined position-tube bounds. Once the robot is inside the tube, at a time around 4.7 s (see figures 7.9 and 7.10), the FSM makes the transition to the *CRUISE_ THROUGH_ TUBE* state. From this point on, the robot moves through the tube with the task defined velocity. However, once the robot reaches the goal area, at a time around 8.5 s (see figure 7.11), the FSM switches to the *STOP_ MOTION* state.

The graph presented in figure 7.8, reports the response of the main control loop for the linear X-direction, i.e. how well the developed control architecture tracks the changing speed reference. Here, the first row depicts the measured and desired robot velocities, together with the upper and lower velocity-tolerances. The second row presents the computed velocity error, i.e. the difference between the measured and desired velocity values, at each time step (for more details on this error calculations, see figure 6.3). Last two rows in this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals, for this DOF.

The presented control graph shows that the developed control architecture has achieved sufficiently accurate and stable tracking of the changing velocity reference, with minor overshoots over the predefined velocity-tolerances. Additionally, the results show a good response of the ABAG controller, even in the case of sudden changes in the reference values.

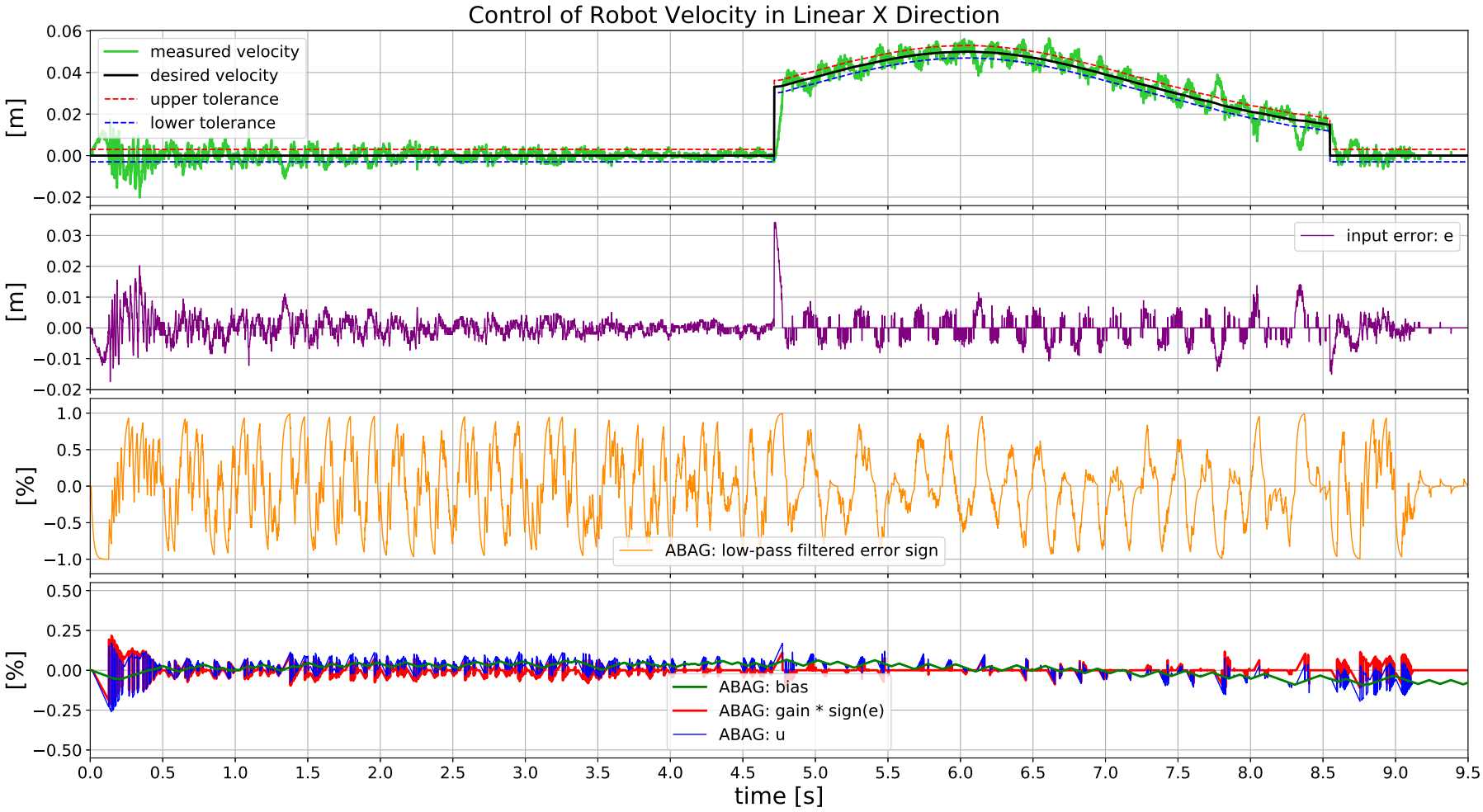

Figure 7.8: Response of the main control loop for the linear X-direction, during the *pre-grasp motion* task execution on the real robot platform. Here, the measured and reference velocity values are expressed with respect to the task frame (T). The notable time-points are at 4.7 s and 8.5 s.

The control graphs presented in figures 7.9 and 7.10, report the response of the main control loop for the linear Y and linear Z directions, respectively. More specifically, these control graphs show how well the developed control architecture keeps the robot's end-effector within the defined position-tube bounds. The first row, in both graphs, depicts the measured robot position together with the upper and lower position-tube tolerances. The second row, in both graphs, presents the computed position error which is, in this case, defined by the difference between the *predicted* and desired position values, at each time step (for more details on this error calculations, see section 6.1.2). The last two rows of these control graphs depict the behaviour of the ABAG controllers, i.e. their adapted signals.

The presented control graphs show that the developed control architecture has met the predefined thresholds and achieved stable tracking of the task-defined position tube, for both linear Y and linear Z directions, without any overshoots from these tight bounds.

In figure 7.12, the computed joint torque commands are depicted, together with the upper and lower saturation limits, for each robot's joint. From this graph, we can observe that the robot, during the task execution, used a very small percentage of the maximum joint commands for only the first and last joint. Stronger actuation of these two joints is not necessary since the specified task requires the robot to move in a vertical plane, that is defined by the rotational axes of the first and last joint. However, we can observe that the robot has used a higher percentage of the maximum joint commands for the second, third and fourth joint, during the task execution. This behaviour occurs due to the fact that, in this case, the gravitational force interfere (contribute negatively to) the task execution and for that reason, both the ABAG controller (while computing the *non-instantaneous* control) and the Popov-Vereshchagin solver (while resolving the *instantaneous* dynamics) are *not* exploiting this already existing force in nature to achieve the desired robot states. Additionally, as seen in the aforementioned graph, the computed command-signals for these three joints are less smooth compared to the joint command-signals produced in the previously conducted simulation-based experiment. The factors that influence this behaviour are: *i*) noisy estimates of the robot state, produced by the youBot's internal sensors and *ii*) not completely accurate robot model used in the dynamics computations. The former factor interferes the computations of the ABAG controllers. However, both the former and the latter factors interfere the computation of *instantaneous* robot dynamics performed by the Popov-Vereshchagin solver.

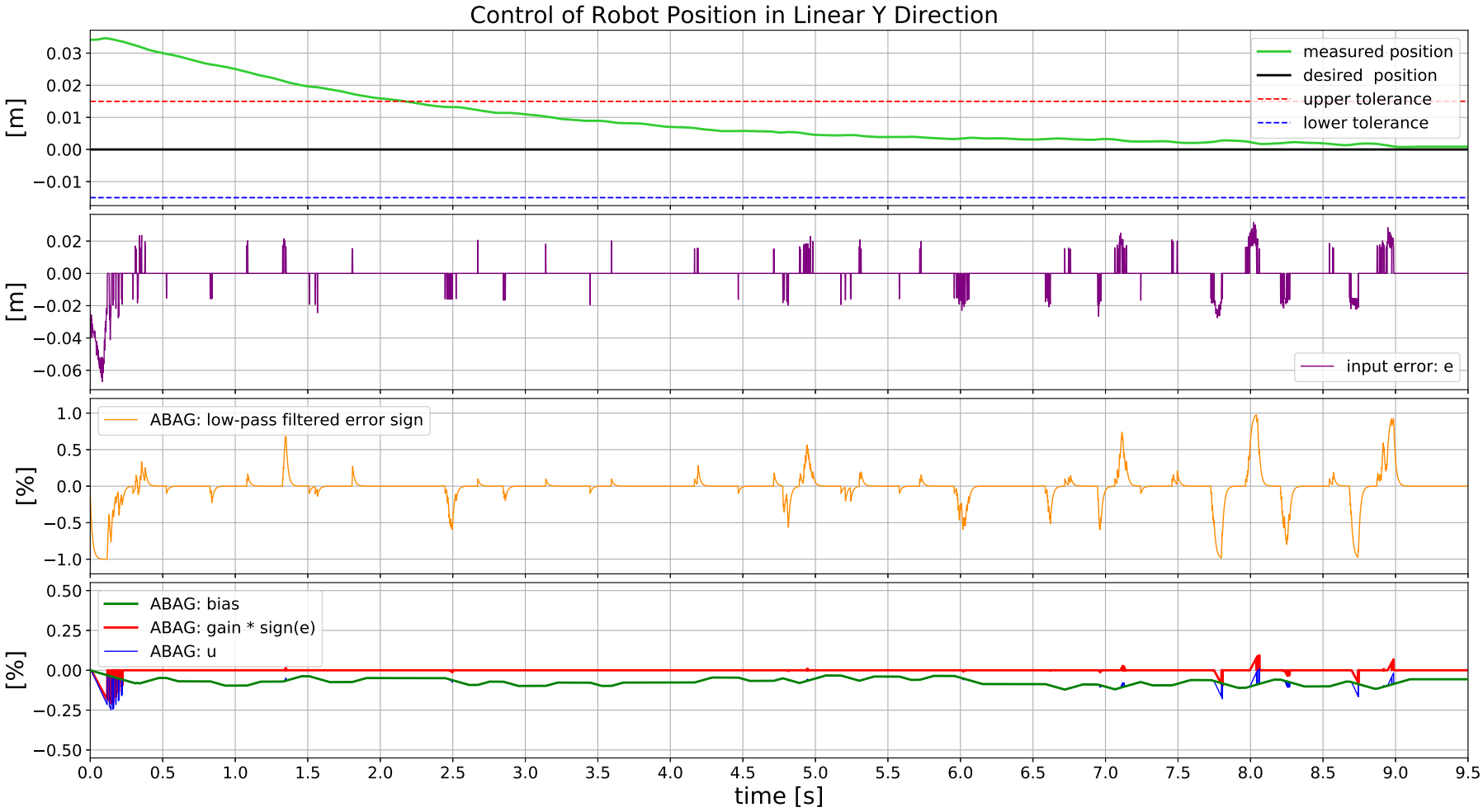

Figure 7.9: Response of the main control loop for the linear Y-direction, during the *pre-grasp motion* task execution on the real robot platform. Here, the measured and reference position values are expressed with respect to the task frame (T).

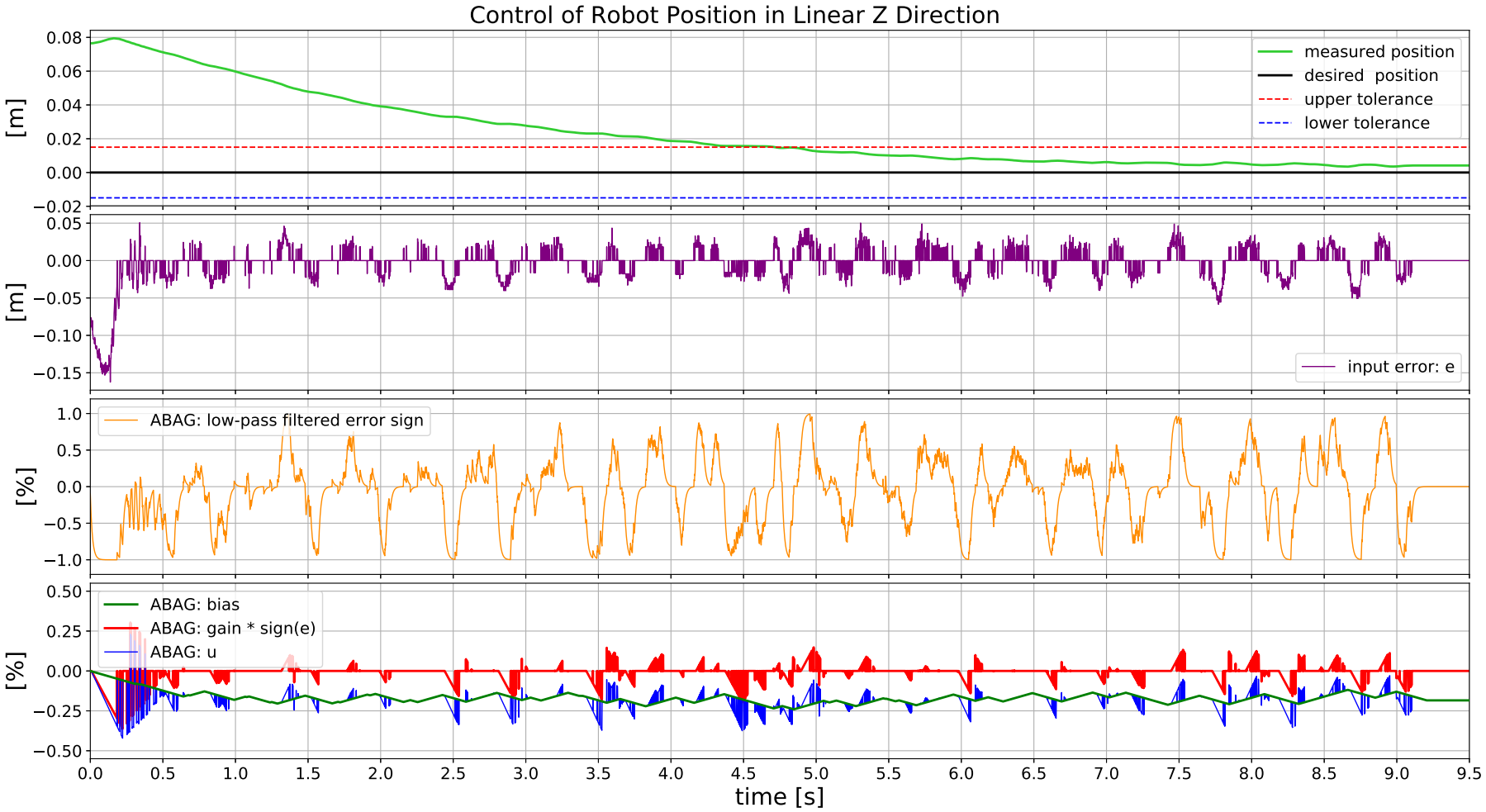

Figure 7.10: Response of the main control loop for the linear Z-direction, during the *pre-grasp motion* task execution on the real robot platform. Here, the measured and reference position values are expressed with respect to the task frame (T).

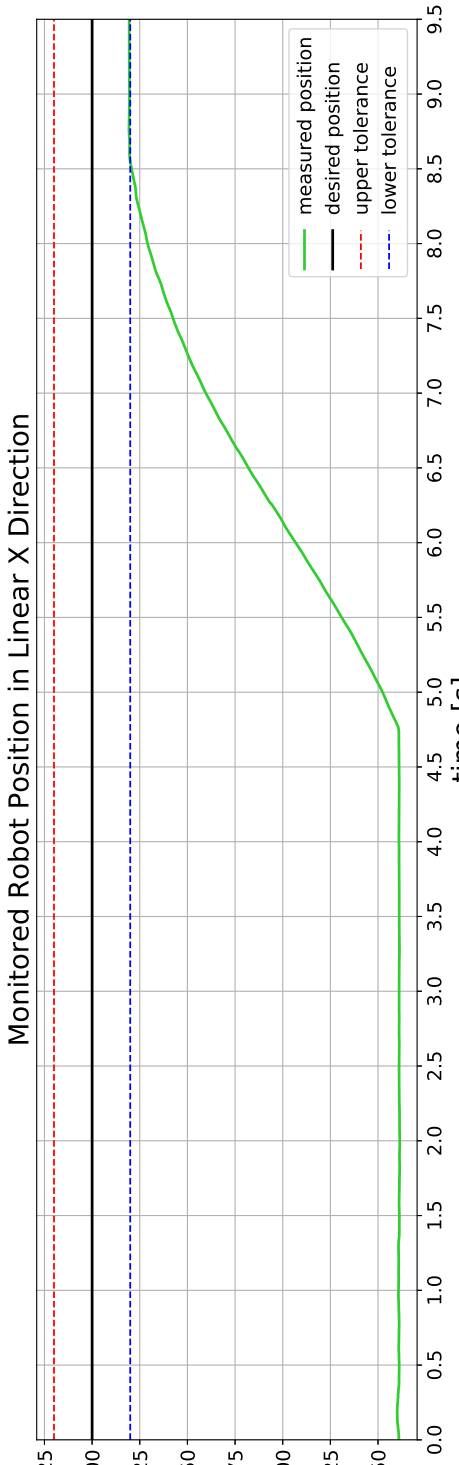

Figure 7.11: Resulting robot's position in the linear X direction, during the *pre-grasp motion* task execution on the real robot platform. Here, the measured position and goal-area values are expressed with respect to the task frame (T). The notable time-points are at 4.7 s and 8.5 s.

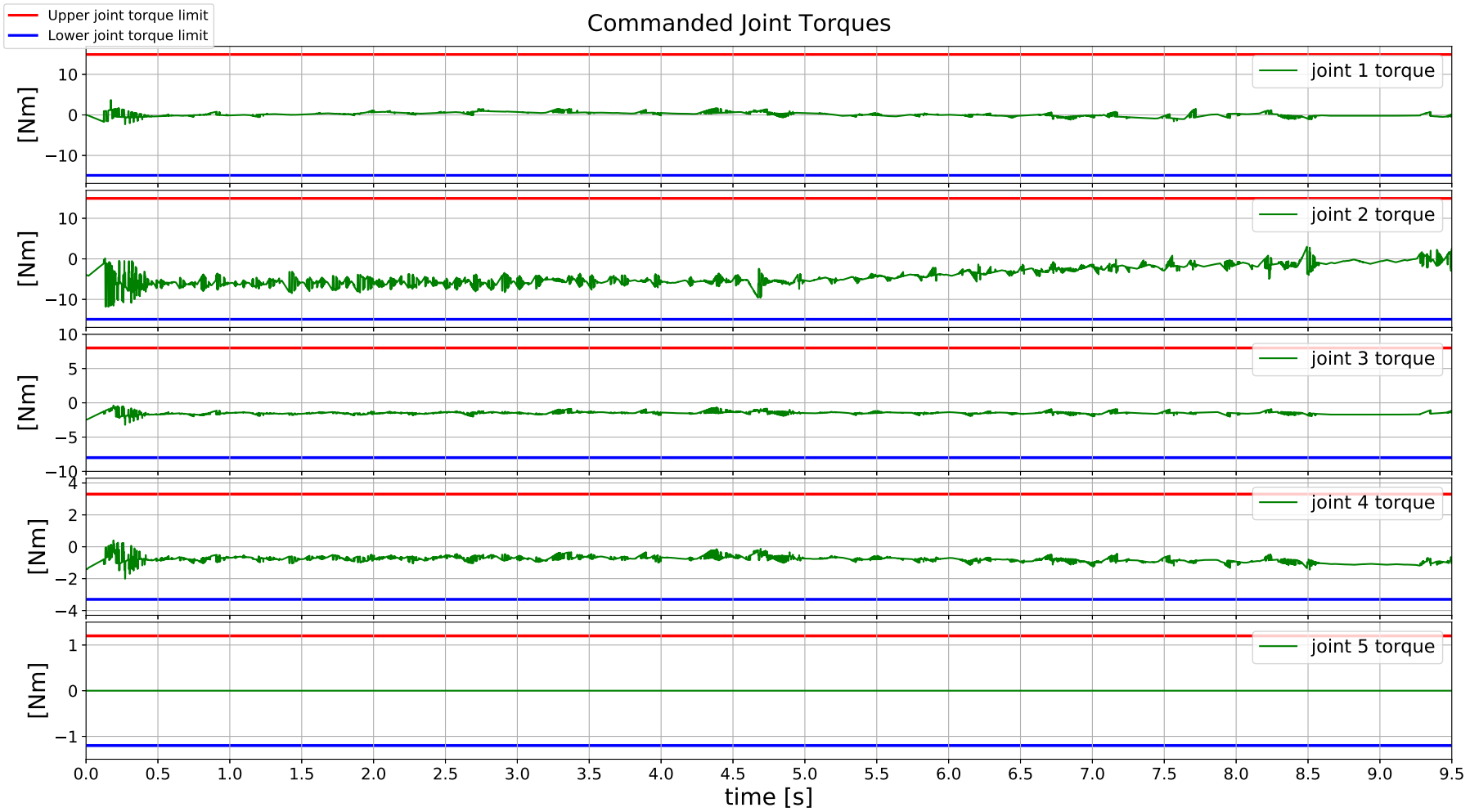

Figure 7.12: Computed joint torque commands, during the *pre-grasp motion* task execution on the real robot platform. The notable time-points are at 4.7 s and 8.5 s.

7.2 Following a Pre-defined Path

7.2.1 Simulation Environment

Experimental Setup

In this experiment, *following a pre-defined path* task is performed in a simulation environment. Here, the task is defined with multiple goal states/areas instead of just one, i.e. the robot is commanded to follow predefined position waypoints along the Cartesian path, while keeping its end-effector within the task defined position-tube bounds. Additionally, the robot is constrained to maintain a predefined speed while cruising through tubular areas, until it completes the path traversal. As previously described in section 6.2.2, each waypoint in the task-defined Cartesian path has associated its own *task frame*. This means that it is necessary to define one instance of the task specification for each waypoint in this path. However, in this experiment, the created task specification instances share the same task parameters (such as position-tube bounds, velocity references and tolerances, goal area tolerances, contact force and total task-time thresholds), while being associated to different *task frames*. The experimental setup is presented in figure 7.13, while an instance of the task specification created for this test is presented in the following:

Task specification for the *following a pre-defined path* task: Simulation environment

move compliantly { // with task frame directions

- ${}^{T_k}X$: 0.06 [m/s], velocity-tolerance 0.005 [m/s]
- ${}^{T_k}Y$: 0 [m], position-tube 0.01 [m]
- ${}^{T_k}Z$: 0 [m], position-tube 0.01 [m]
- ${}^{T_k}aX$:
- ${}^{T_k}aY$:
- ${}^{T_k}aZ$:

} no specification

} until either:

- goal area reached, tolerance 0.001 [m]
- or ${}^{T_k}F_X$ / ${}^{T_k}F_Y$/ ${}^{T_k}F_Z > 0.5$ [N]
- or total task-time > 12 [s]

Here, T_k stands for the *task frame* associated with *k-th* waypoint in the defined path. In this setup, the initial robot configuration is chosen in such a way that, at the start of the task execution, the end-effector's position is outside of the tube bounds, which are defined in the task specification instance that corresponds to the first waypoint.

The main control parameters used in this experiment are summarized in table 7.3. Here, the positions of the start and end path-points are expressed with respect to the robot's base frame. The path's step parameters are also expressed with respect to the robot's base frame. In this table, the *maximum*

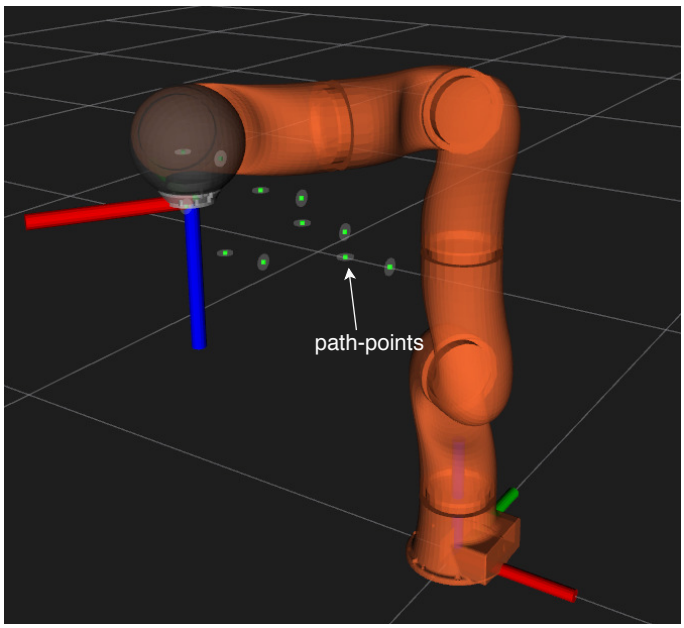

Figure 7.13: Visualization of the KUKA LWR 4 model and its initial configuration for the *following a pre-defined path* task, in the simulation environment. Here, the simulation software [90] has generated virtual path-points and their associated goal areas (based on the task-defined path and bounds), to enhance the visualization for this task execution. These waypoints are represented with green color, while the virtual goal-area for each waypoint is depicted with gray color. Here, a visualization of the virtual tubular areas is omitted.

commands represent the maximum allowed acceleration energy setpoints for the robot's end-effector (for more details on this quantity see section 6.2.2).

Results

The result of this experiment is depicted in figures 7.14, 7.15, 7.16, 7.17, 7.18 and 7.19. The first three control graphs report the response of the main control loop for the linear X, linear Y and linear Z directions, respectively, i.e. how well the control architecture developed in section 6.2.2 tracks the task defined path and bounds. In these three aforementioned control graphs, the measured and reference robot positions and velocity correspond to the whole path. Additionally, these measured and reference quantities are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*.

From the first five above-presented control graphs, we can observe four different control phases in this task execution. More specifically, since at the start of the task execution the end-effector's position is outside of the tube bounds, the finite state machine (FSM) that is part of the control architecture developed in section 6.2.2 starts the task execution in the *START-TO-CRUISE* state. Here, the robot is commanded to maintain the zero tube-velocity (in the linear X direction of the first task frame (T_1)), until it reaches the task-defined position-tube bounds (in the linear

Parameter	Value
Initial joint configuration	1.00, 0.00, 0.00, -1.57, 0.00, 1.57, 0.00 [rad]
Task path	**Step Sequence** Largest step in Z direction: 12.0 cm Largest step in X direction: 3.0 cm Number of path points: 12
Start path position	X: -0.200, Y: -0.348, Z: 0.632 [m]
End path position	X: 0.099, Y: -0.348, Z: 0.612 [m]
Prediction horizon	1.5 s
Maximum commands	X: 60.0, Y: 60.0, Z: 60.0 $[\frac{Nm}{s^2}]$.
Control loop frequency	630 Hz
Gravity compensation	Disabled

Table 7.3: Main control parameters for the *following a pre-defined path* experiment in the simulation environment.

Y and Z directions of the first task frame (T_1)). Once the robot is inside the tube, at a time around 0.1 s (see figures 7.15 and 7.16), the FSM makes the transition to the *CRUISE_THROUGH_TUBE* state. From this point on, the robot moves through the tube with the task defined velocity. Once the robot reaches the current sub-goal area in the path, the FSM makes a switch to the *TRANSITION_PATH_SECTION* state. Here, the current instance of the task specification, that is used in the control loop computations, is replaced with an instance that corresponds to the next waypoint in the path. For more details on the behaviour of these two (*CRUISE_THROUGH_TUBE* and *TRANSITION_PATH_SECTION*) states throughout the whole path traversal, see section 6.2.2. However, these changes of the task frame cause certain effects on the control computations, which can be observed in the figure 7.14. More specifically, at time instances around 0.4 s, 0.6 s, 1.9 s, 2.9 s, etc., we can observe large changes in the velocity measurements, i.e. the value of measured velocity drops to a value that is close to 0 $\frac{m}{s}$. These effects are caused by large changes in the relative orientation between two consecutive task frames (see figure 7.13). However, in the figure 7.14, we can also observe that the value of reference velocity did not change in these situations since the two consecutive task specification instances share the same velocity parameters. Nevertheless, once the robot reaches the final goal area, i.e. the one that corresponds to the last path-section, at a time around 11.8 s (see figure 7.14), the FSM switches to the *STOP_MOTION* state and stops the robot.

The control graph presented in figure 7.14, reports the response of the main control loop for the linear X-direction, i.e. how well the control architecture developed in section 6.2.2 tracks the speed reference. Here, the first row depicts the measured and desired robot velocities, together with the upper and lower velocity-tolerances. The second row presents the computed velocity error, i.e. the difference between the measured and desired velocity values, at each time step (for more details on this error calculations, see figure 6.3). Last two rows in this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals,

for this DOF.

Here, we can observe effects on (i.e. large changes in) the robot's velocity, caused by two unforeseen disturbances that act on robot system. The first disturbance occurred at the 0.0 s time, is produced by an imperfect disengage of the robot's *simulated* brakes. The imperfect disengaging of these brakes is performed internally by the simulation framework once it received the first set of joint commands. On the other side, the second disturbance occurred at a time around 2.1 s is produced by the robot's *null-space* motion, which is not explicitly controlled in this experiment. Nevertheless, the presented control graph shows that the developed control architecture has achieved sufficiently accurate and stable tracking of the velocity reference, with minor overshoots over the predefined velocity-tolerances. Additionally, the results show a very good response time of the ABAG controller; even in the case of *i*) sudden changes in the reference task frame, *ii*) disturbance produced by an imperfect disengage of robot's simulated brakes and *iii*) disturbance produced by the robot's *null-space* motion.

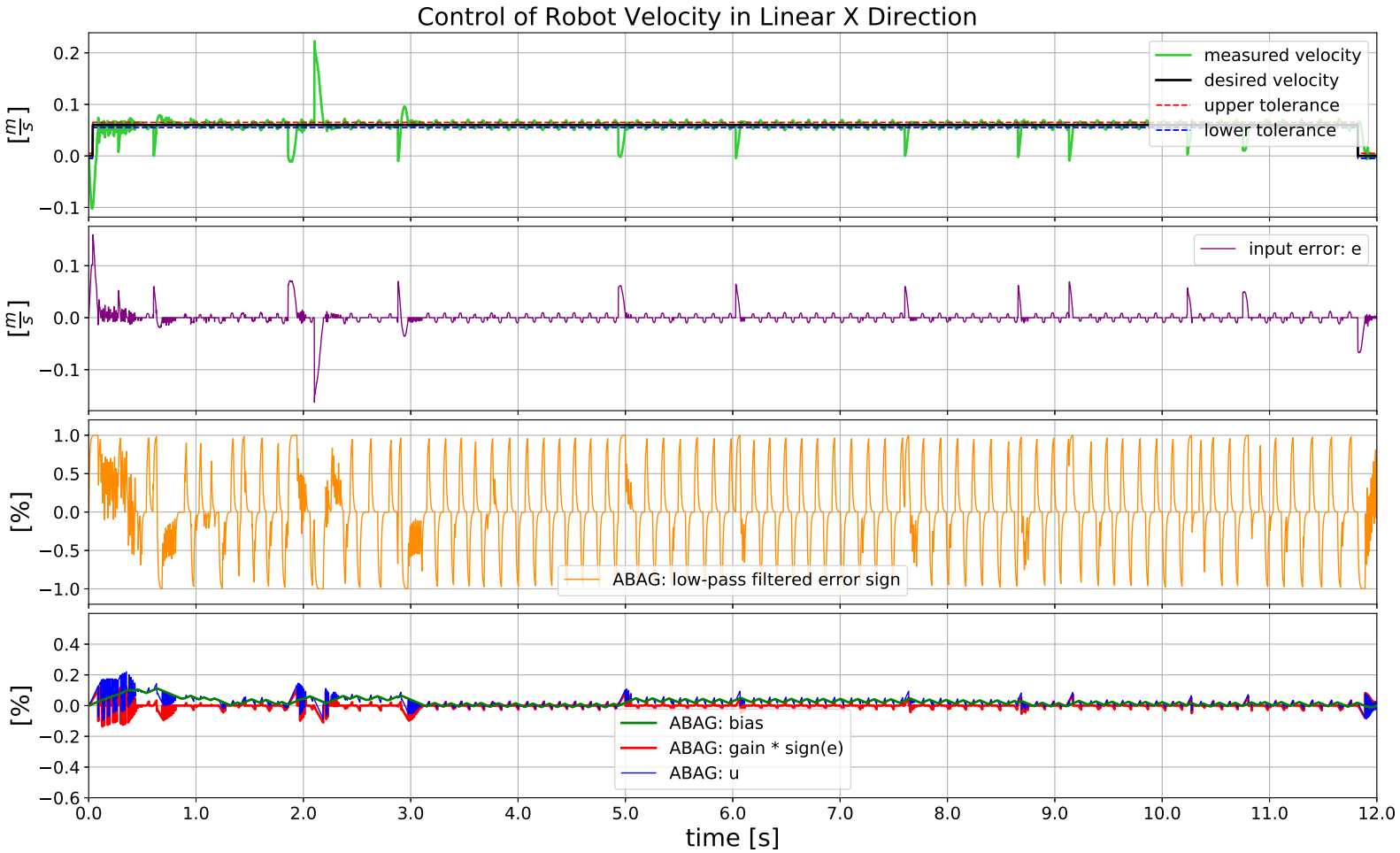

Figure 7.14: Response of the main control loop for the linear X-direction, during the *following a pre-defined path* task execution in the simulation environment. Here, the measured and reference velocity values are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*. The notable time-points are at 0.0 s, 0.1 s, 0.4 s, 0.6 s, 1.9 s, 2.1 s, 2.9 s, 4.9 s, 6.0 s, 7.6 s, 8.7 s, 9.2 s, 10.3 s, 10.8 s and 11.8 s

The control graphs presented in figures 7.15 and 7.16, report the response of the main control loop for the linear Y and linear Z directions, respectively. More specifically, these control graphs show how well the developed control architecture keeps the robot's end-effector within the defined position-tube bounds. The first row, in both graphs, depicts the measured robot position together with the upper and lower position-tube tolerances. The second row, in both graphs, presents the computed position error which is, in this case, defined by the difference between the *predicted* and desired position values, at each time step (for more details on this error calculations, see section 6.1.2). The last two rows of these control graphs depict the behaviour of the ABAG controllers, i.e. their adapted signals. The same position trend that is depicted in figure 7.15 is also visible in the figure 7.18, i.e. these two figures present the same aspect of the task execution but from the different frame of reference. On the other hand, the same position trend that is depicted in figure 7.16 is also visible in the figure 7.17, i.e. these two figures present the same aspect of the task execution but from the different frame of reference.

The aforementioned control graphs together with figures 7.17 and 7.18 show that the developed control architecture has met the predefined thresholds and achieved stable tracking of the task-defined tubular areas, for both linear Y and linear Z directions, without any overshoots from these tight bounds.

In figure 7.19, the computed joint torque commands are depicted, together with the upper and lower saturation limits, for each robot's joint. These saturation limits are enforced in the dynamics computations of this control architecture, i.e. in line 26 of **Algorithm** 2. However, from this figure, we can observe that the robot has executed the task by using a very small percentage of the maximum joint torques. Furthermore, actuation in some of the joints, such as actuation in first, fifth and seventh, was not required at all, during the large part of the task execution. This behaviour occurs due to the fact that both, the ABAG controller (while computing the *non-instantaneous* control) and the Popov-Vereshchagin solver (while resolving the *instantaneous* dynamics) are (whenever possible) exploiting the already existing forces in nature, such as, in this case, modelled gravitational and non-modelled joint friction forces, to achieve the desired robot states.

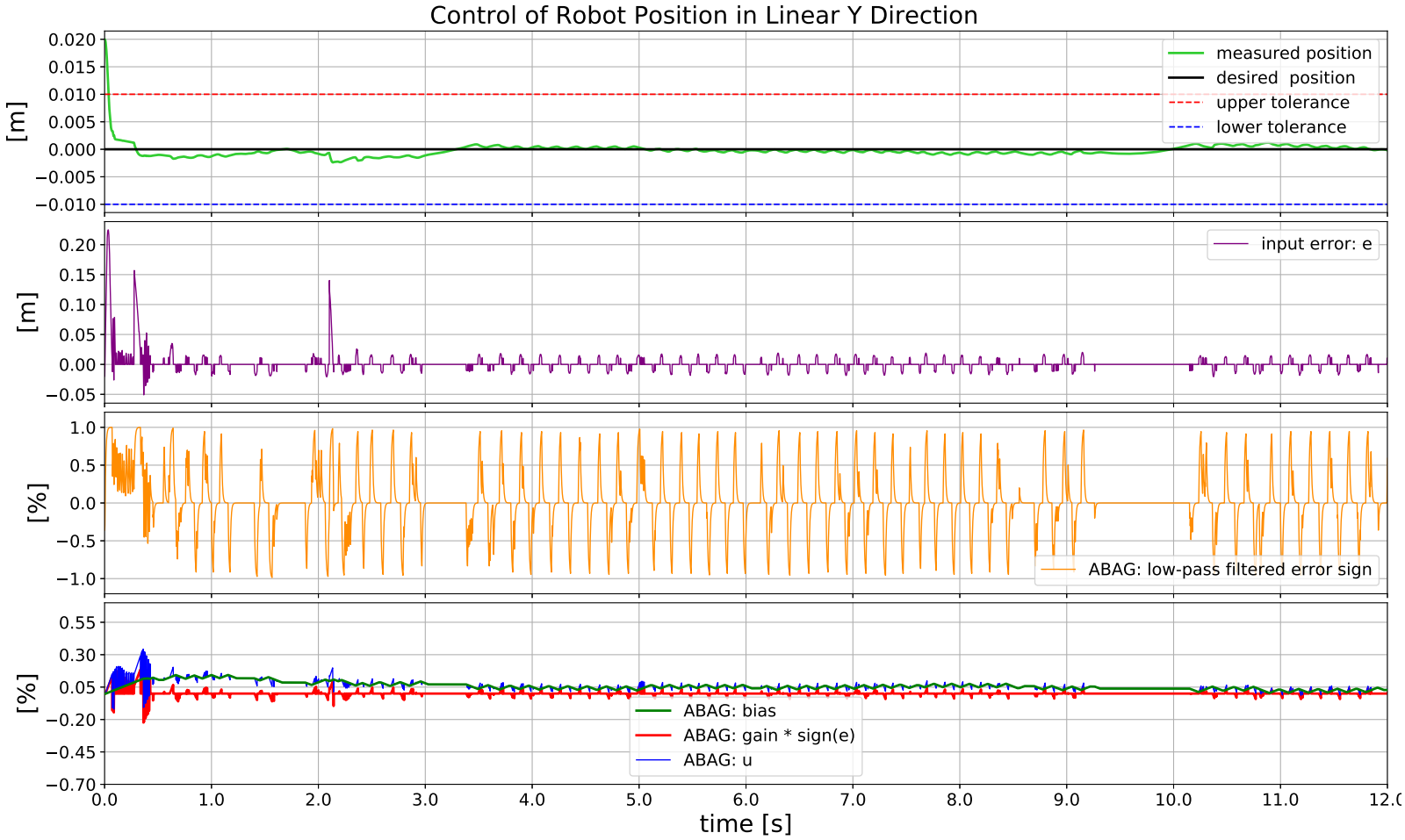

Figure 7.15: Response of the main control loop for the linear Y-direction, during the *following a pre-defined path* task execution in the simulation environment. Here, the measured and reference position values are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*.

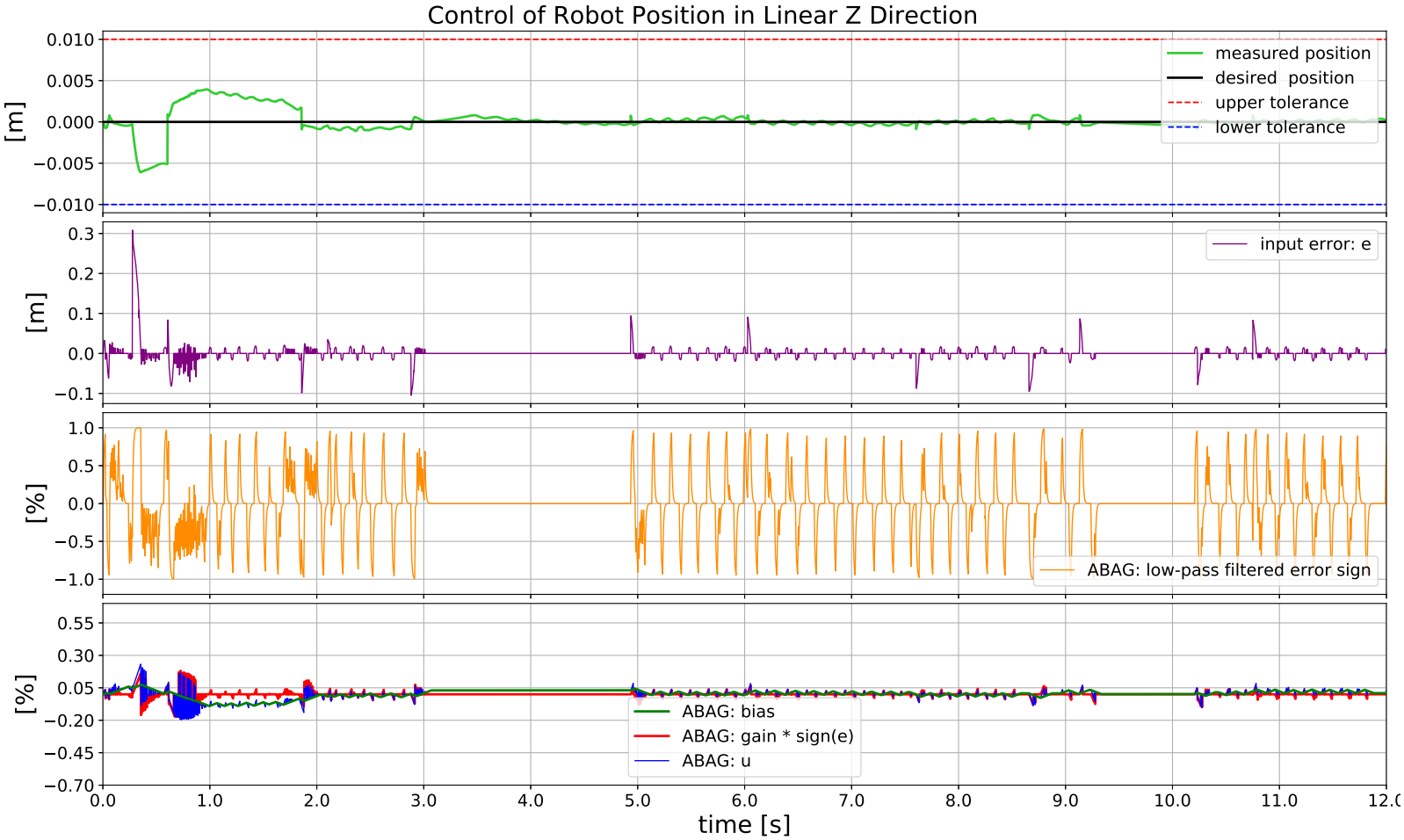

Figure 7.16: Response of the main control loop for the linear Z-direction, during the *following a pre-defined path* task execution in the simulation environment. Here, the measured and reference position values are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*.

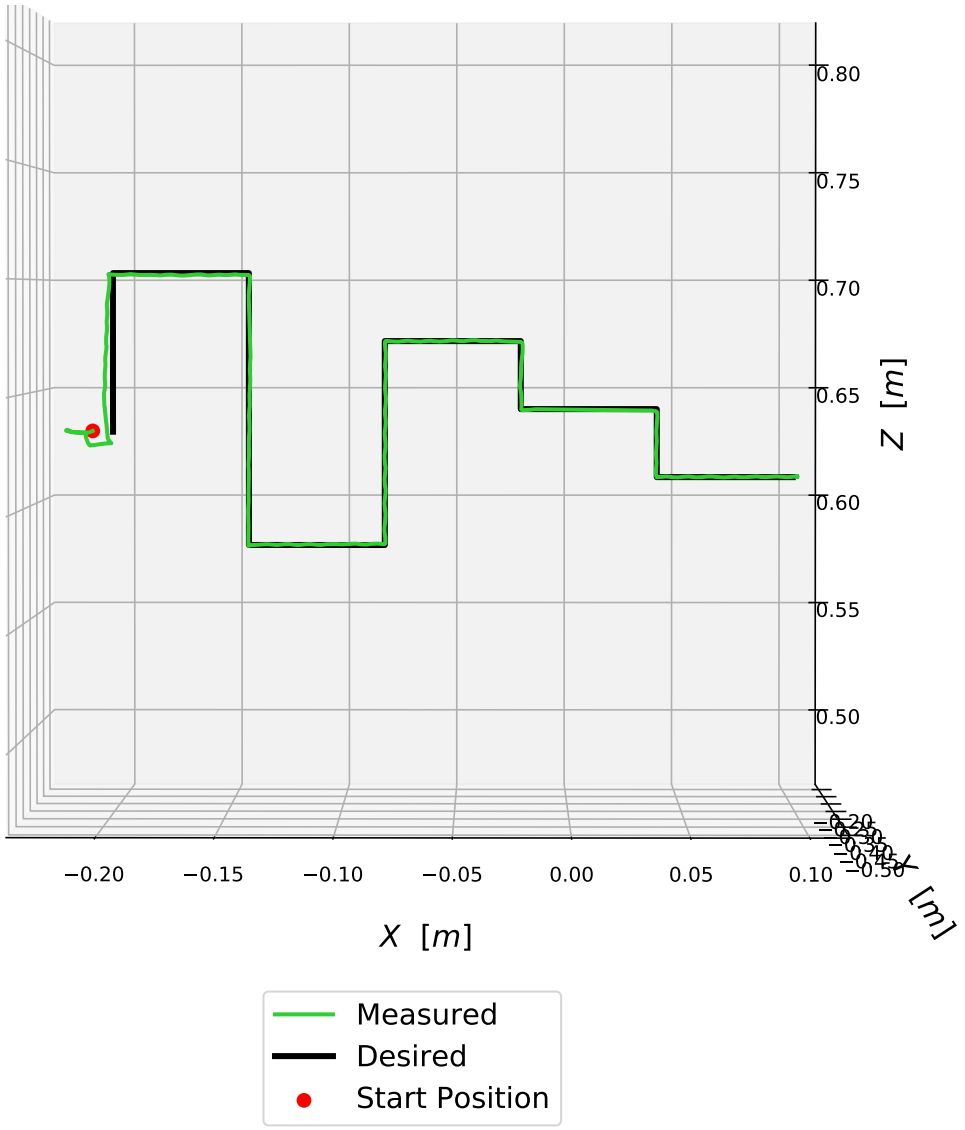

Figure 7.17: Resulting robot's position in the X and Z directions, during the *following a pre-defined path* task execution in the simulation environment. Here, the measured and reference position values are expressed with respect to the robot's base frame.

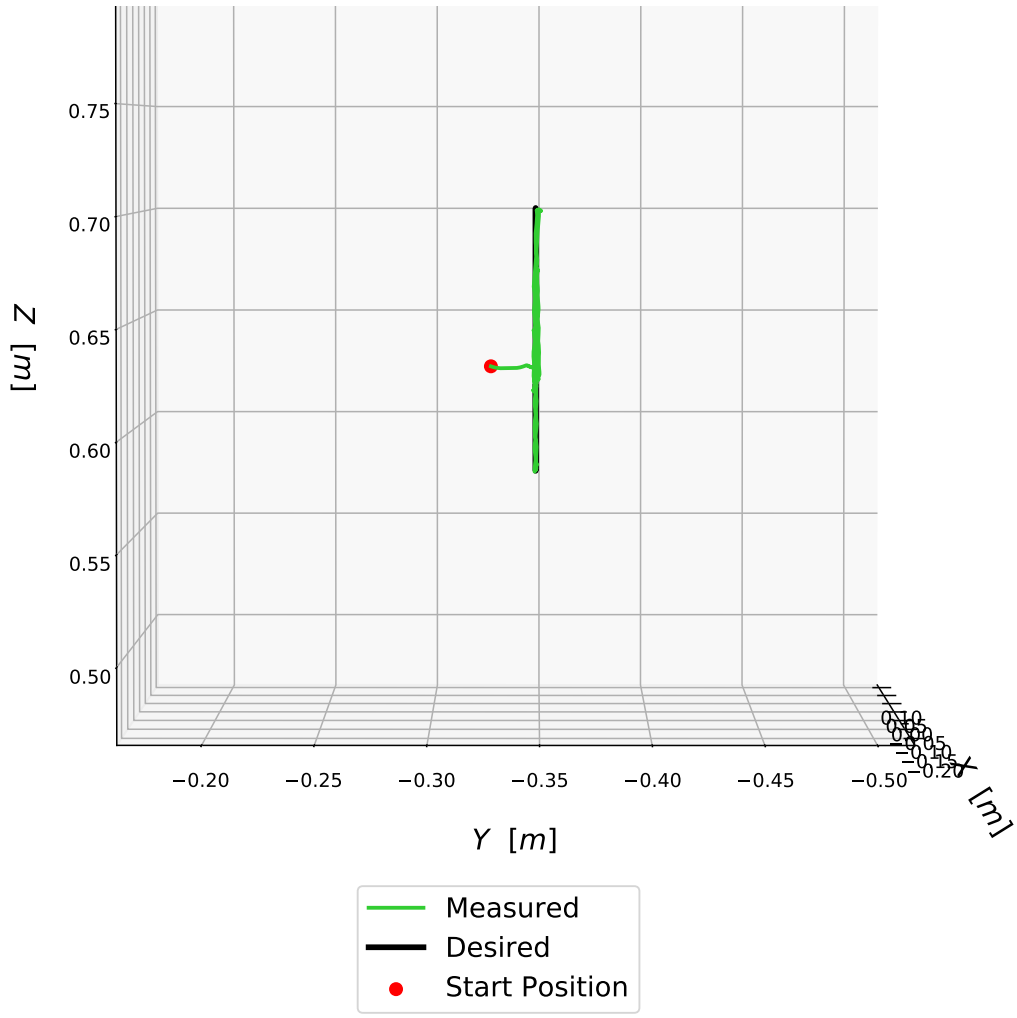

Figure 7.18: Resulting robot's position in the Y and Z directions, during the *following a pre-defined path* task execution in the simulation environment. Here, the measured and reference position values are expressed with respect to the robot's base frame.

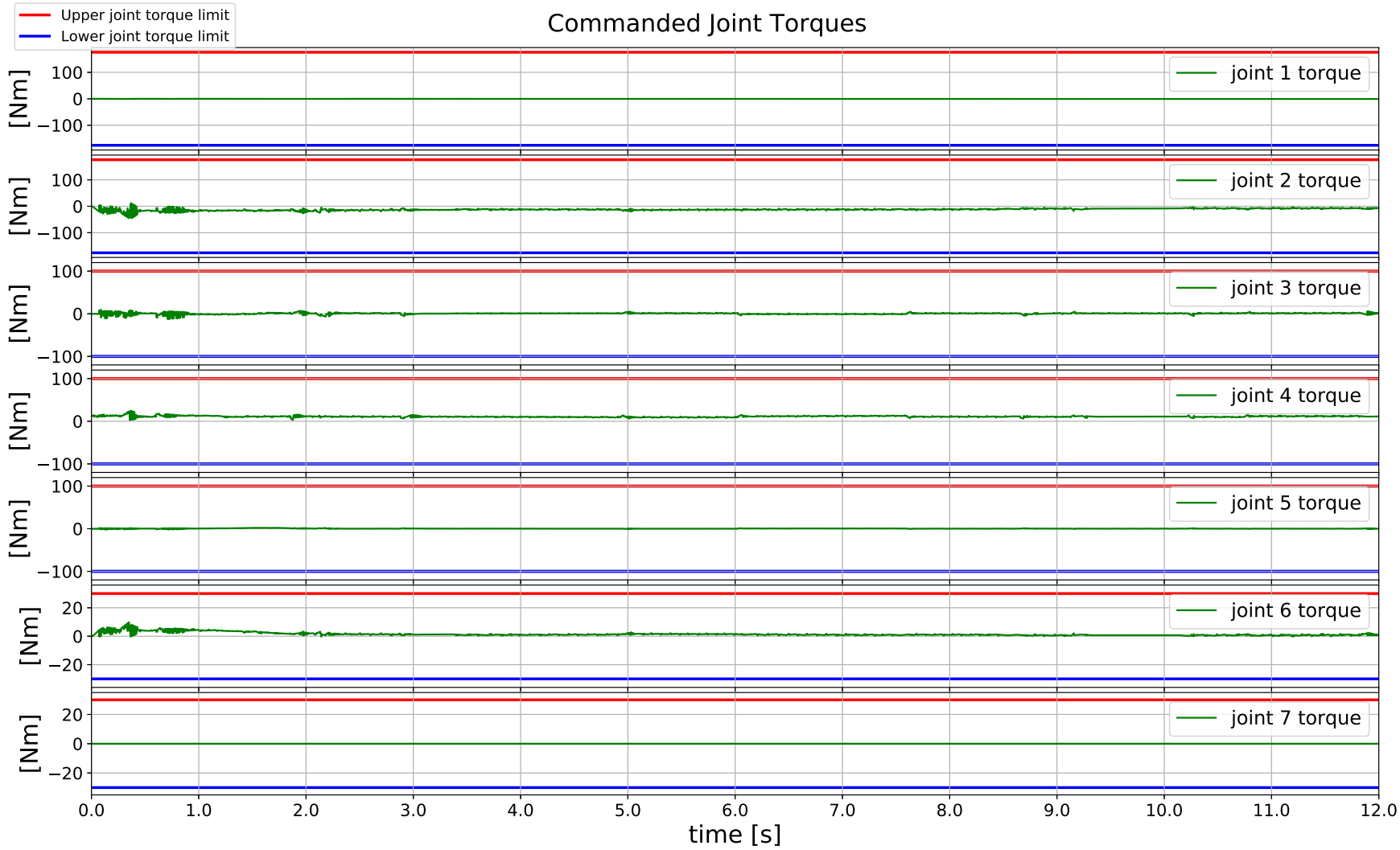

Figure 7.19: Computed joint torque commands, during the *following a pre-defined path* task execution in the simulation environment.

7.2.2 Real Robot Platform

Experimental Setup

In this experiment, *following a pre-defined path* task is performed on a real robot platform, i.e. on the 5-DOF KUKA youBot. Here, similarly to the previous experiment, the task is defined with multiple goal states/areas instead of just one, i.e. the robot is commanded to follow predefined position waypoints along the Cartesian path, while keeping its end-effector within the task defined position-tube bounds. Additionally, the robot is constrained to maintain a predefined speed while cruising through tubular areas, until it completes the path traversal. As previously described in section 6.2.2, each waypoint in the task-defined Cartesian path has associated its own *task frame*. This means that it is necessary to define one instance of the task specification for each waypoint in this path. However, in this experiment, the created task specification instances share the same task parameters (such as position-tube bounds, velocity references and tolerances, goal area tolerances, contact force and total task-time thresholds), while being associated to different *task frames*. The task specification created for this test is presented in the following:

Task specification for the *following a pre-defined path* task: Real robot platform

move compliantly { // with task frame directions

- ${}^{T_k}X$: 0.05 [m/s], velocity-tolerance 0.003 [m/s]
- ${}^{T_k}Y$: 0 [m], position-tube 0.015 [m]
- ${}^{T_k}Z$: 0 [m], position-tube 0.015 [m]
- ${}^{T_k}aX$:
- ${}^{T_k}aY$:
- ${}^{T_k}aZ$:

  } no specification

} until either:

- goal area reached, tolerance 0.01 [m]
- or total task-time $>$ 18 [s]

Here, T_k stands for the *task frame* associated with *k-th* waypoint in the defined path. In this setup, the initial robot configuration is chosen in such a way that, at the start of the task execution, the end-effector's position is already inside the tube bounds, which are defined in the task specification instance that corresponds to the first waypoint.

The main control parameters used in this experiment are summarized in table 7.4. Here, the positions of the start and end path-points are expressed with respect to the robot's base frame. In this table, the *maximum commands* represent the maximum allowed acceleration energy setpoints for the robot's end-effector (for more details on this quantity see section 6.2.2).

Parameter	Value
Initial joint configuration	1.000, 1.353, -0.549, 0.732, 2.962 [rad]
Task path	**Linear chirp function** Start frequency: 0.5 Hz End frequency: 4.5 Hz Amplitude: 0.05 [m] Number of path points: 90
Start path position	X: -0.085, Y: 0.272, Z: 0.238 [m]
End path position	X: 0.222, Y: 0.285, Z: 0.238 [m]
Prediction horizon	2.5 s
Maximum commands	X: 10.0, Y: 10.0, Z: 10.0 $[\frac{Nm}{s^2}]$.
Control loop frequency	660 Hz
Gravity compensation	Disabled

Table 7.4: Main control parameters for the *following a pre-defined path* experiment on the real robot platform.

Results

The result of this experiment is depicted in figures 7.20, 7.21, 7.22, 7.23, 7.24 and 7.25. The first three control graphs report the response of the main control loop for the linear X, linear Y and linear Z directions, respectively, i.e. how well the control architecture developed in section 6.2.2 tracks the task defined path and bounds, on a real robot platform. In these three aforementioned control graphs, the measured and reference robot's positions and velocity correspond to the whole path. Additionally, these measured and reference quantities are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*.

From the first five above-presented control graphs, we can observe three different control phases in this task execution. More specifically, since at the start of the task execution, the end-effector's position is already inside the tube bounds, the finite state machine (FSM), that is part of the control architecture developed in section 6.2.2, starts the task execution in the *START-TO-CRUISE* state and immediately at the 0.0s time (in the first iteration of the control loop), makes the transition to the *CRUISE_THROUGH_TUBE* state (see figures 7.21 and 7.22). From this point on, the robot moves through the tube with the task defined velocity. Once the robot reaches the current sub-goal area in the path, the FSM makes a switch to the *TRANSITION_PATH_SECTION* state. Here, the current instance of the task specification, that is used in the control loop computations, is replaced with an instance that corresponds to the next waypoint in the path. For more details on the behaviour of these two (*CRUISE_THROUGH_TUBE* and *TRANSITION_PATH_SECTION*) states throughout the whole path traversal, see section 6.2.2. However, these changes of the task frame cause certain effects on the control computations, which can be observed in the figure 7.20. More specifically, at time instances around 3.6 s, 5.6 s, 7.6 s, 9.4, etc., we can observe large changes in the velocity measurements, i.e. the value of measured velocity drops to a value

that is close to or bellow 0 $\frac{m}{s}$. These effects are caused by large changes (in most cases even larger compared to those in the simulation-based experiment) in the relative orientation between two consecutive task frames. As the frequency of the task-defined chirp function increases over time, the effect becomes stronger. However, in the figure 7.20, we can also observe that the value of reference velocity did not change in these situations since the two consecutive task specification instances share the same velocity parameters. Nevertheless, once the robot reaches the final goal area, i.e. the one that corresponds to the last path-section, at a time around 17.1 s (see figure 7.20), the FSM switches to the *STOP_MOTION* state and stops the robot.

The control graph presented in figure 7.20, reports the response of the main control loop for the linear X-direction, i.e. how well the control architecture developed in section 6.2.2 tracks the speed reference. Here, the first row depicts the measured and desired robot velocities, together with the upper and lower velocity-tolerances. The second row presents the computed velocity error, i.e. the difference between the measured and desired velocity values, at each time step (for more details on this error calculations, see figure 6.3). Last two rows in this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals, for this DOF.

Here, no unforeseen disturbances have occurred during the task execution. The presented control graph shows that the developed control architecture has achieved sufficiently accurate and stable tracking of the velocity reference, with minor overshoots over the tight velocity-tolerances. Additionally, the results show a very good response time of the ABAG controller; even in the case of sudden changes in the reference task frame.

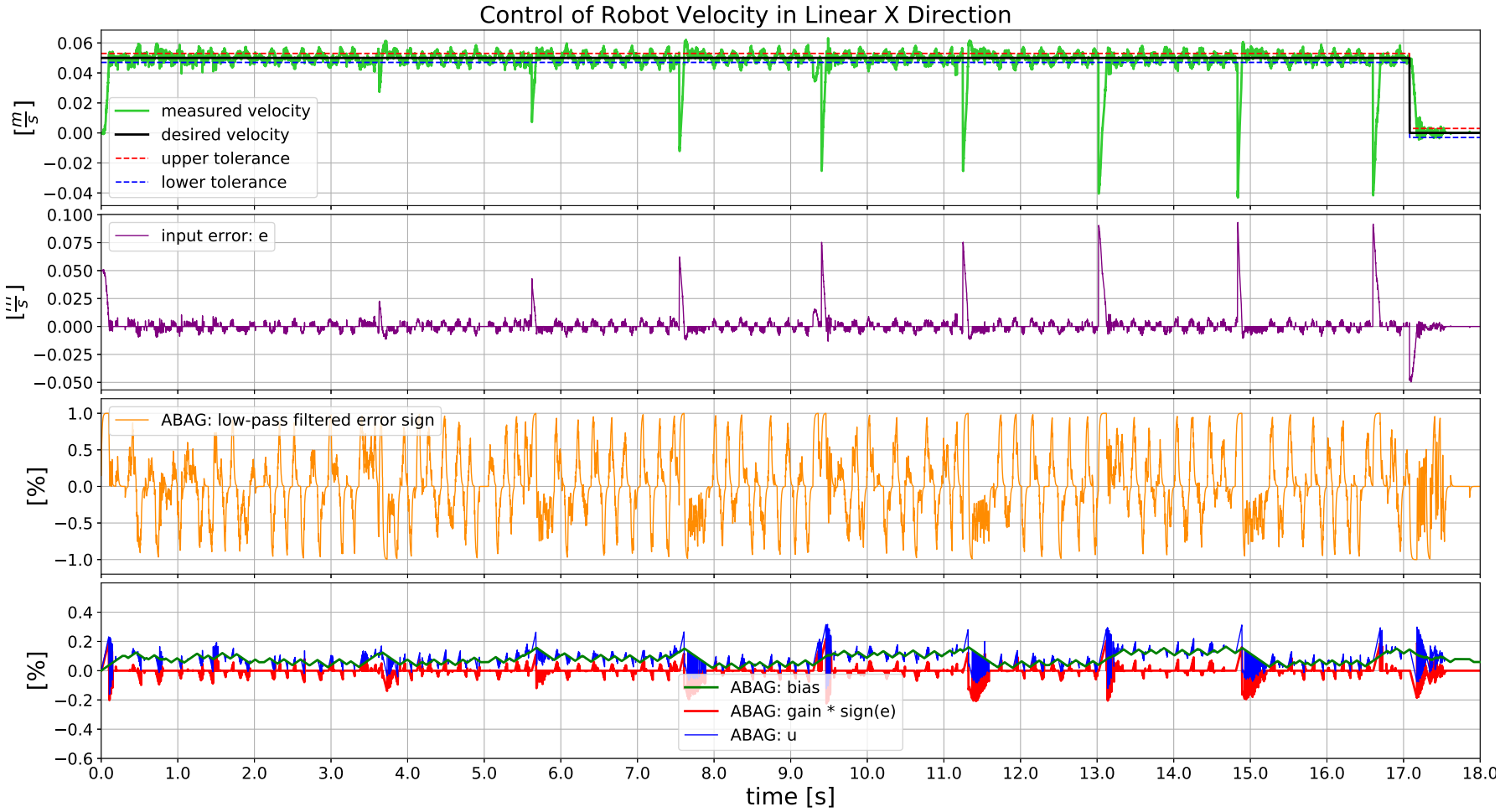

Figure 7.20: Response of the main control loop for the linear X-direction, during the *following a pre-defined path* task execution on the real robot platform. Here, the measured and reference velocity values are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*. The notable time-points are at 3.6 s, 5.6 s, 7.6 s, 9.4 s, 11.2 s, 13.0 s, 14.9 s, 16.6 s and 17.1 s

The control graphs presented in figures 7.21 and 7.22, report the response of the main control loop for the linear Y and linear Z directions, respectively. More specifically, these control graphs show how well the developed control architecture keeps the robot's end-effector within the defined position-tube bounds. Here, the same position trend that is presented in figure 7.21 is also visible in the figure 7.23, i.e. these two figures present the same aspect of the task execution but from the different frame of reference. On the other hand, the same position trend that is presented in figure 7.22 is also visible in the figure 7.24, i.e. these two figures present the same aspect of the task execution but from the different frame of reference.

The aforementioned control graphs together with figures 7.23 and 7.24 show that the developed control architecture has met the predefined thresholds and achieved stable tracking of the task-defined tubular areas, for both linear Y and linear Z directions, without any overshoots from these tight bounds. However, the position oscillations that are clearly visible in figures 7.21 and 7.22, occur due to limitations in robot's physical capabilities. More specifically, the control graph presented in figure 7.21 shows us that the robot has limited capabilities for performing curved motions. On the other side, the control graph presented in figure 7.22 shows us that the system's performance degrades as the manipulator stretches further in the space.

In figure 7.25, the computed joint torque commands are depicted, together with the upper and lower saturation limits, for each robot's joint. From this graph, we can observe that the robot has used a high percentage of the maximum joint commands for the second, third and fourth joint, during the task execution. This behaviour occurs due to: *i*) the aforementioned limitations in robot's physical capabilities and *ii*) the fact that gravitational force does not assist (contribute) the task execution and thus, it is *not* exploited for achieving the desired states. Additionally, in this graph, we can see that the computed command-signals for first four joints are less smooth compared to the joint command-signals produced in the previously conducted simulation-based experiment. The factors that influence this behaviour are: *i*) noisy estimates of the robot state, produced by the youBot's internal sensors and *ii*) not completely accurate robot model used in the dynamics computations. The former factor interferes the computations of the ABAG controllers. However, both the former and latter factors interfere the computation of *instantaneous* robot dynamics performed by the Popov-Vereshchagin solver.

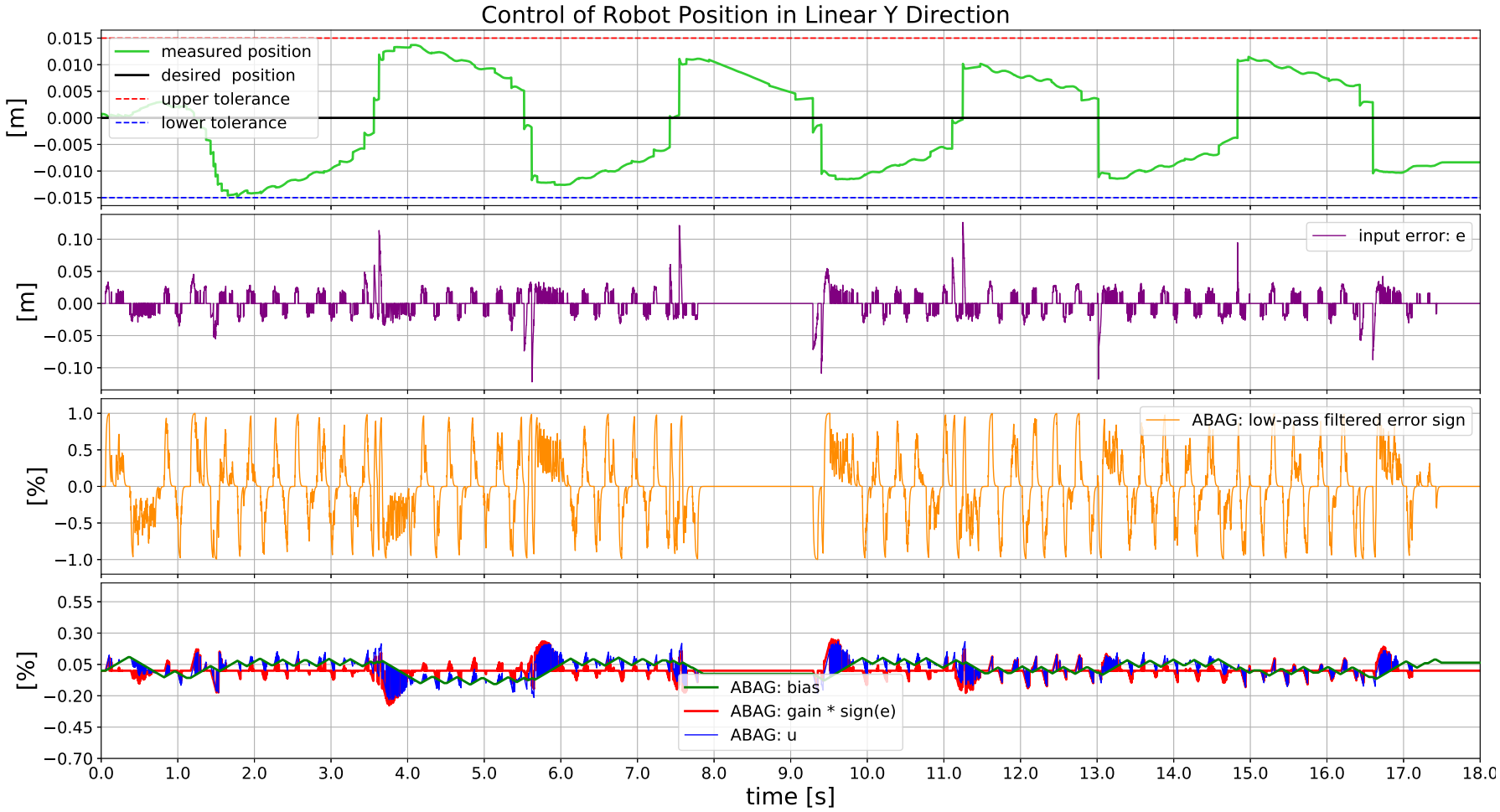

Figure 7.21: Response of the main control loop for the linear Y-direction, during the *following a pre-defined path* task execution on the real robot platform. Here, the measured and reference position values are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*.

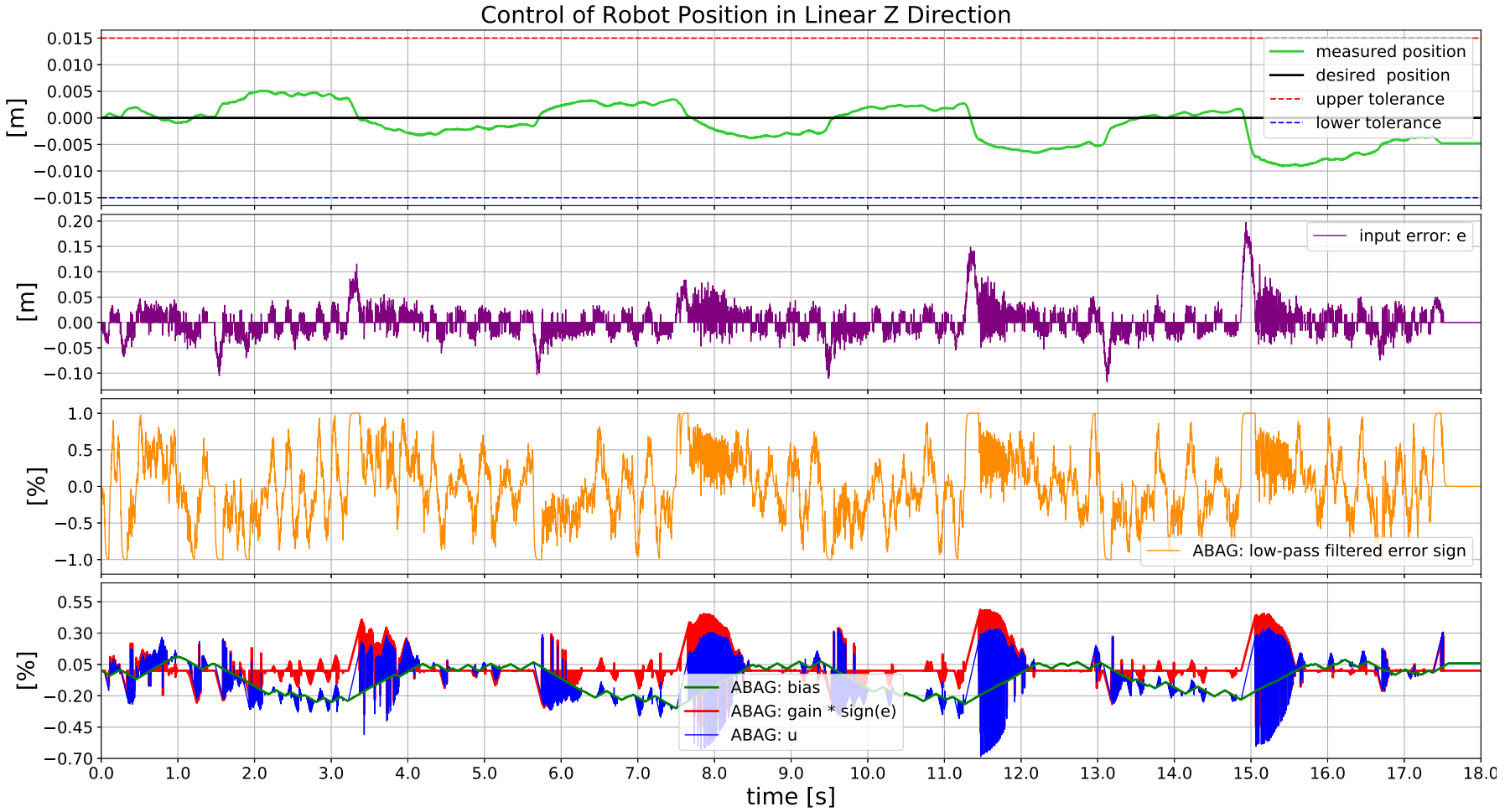

Figure 7.22: Response of the main control loop for the linear Z-direction, during the *following a pre-defined path* task execution on the real robot platform. Here, the measured and reference position values are, at each time instance, expressed with respect to a *task frame* (T_k) that corresponds to the path section through which the robot is currently *cruising*.

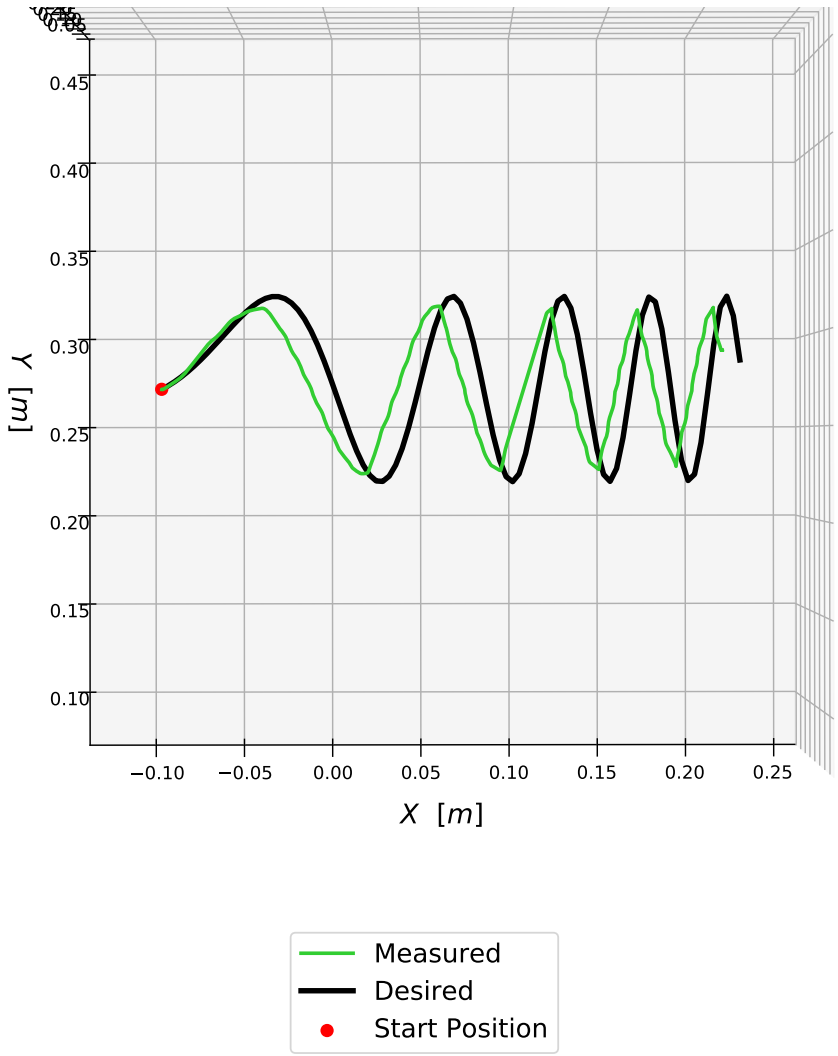

Figure 7.23: Resulting robot's position in the X and Y directions, during the *following a pre-defined path* task execution on the real robot platform. Here, the measured and reference position values are expressed with respect to the robot's base frame.

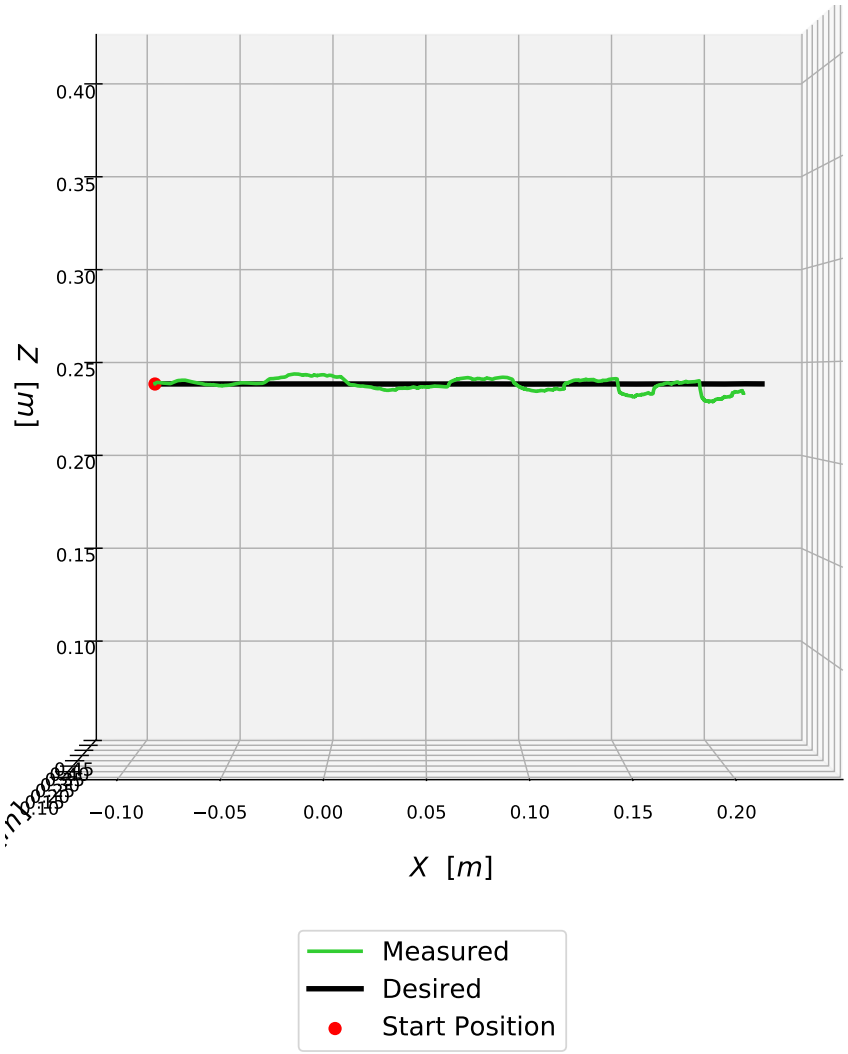

Figure 7.24: Resulting robot's position in the X and Z directions, during the *following a pre-defined path* task execution on the real robot platform. Here, the measured and reference position values are expressed with respect to the robot's base frame.

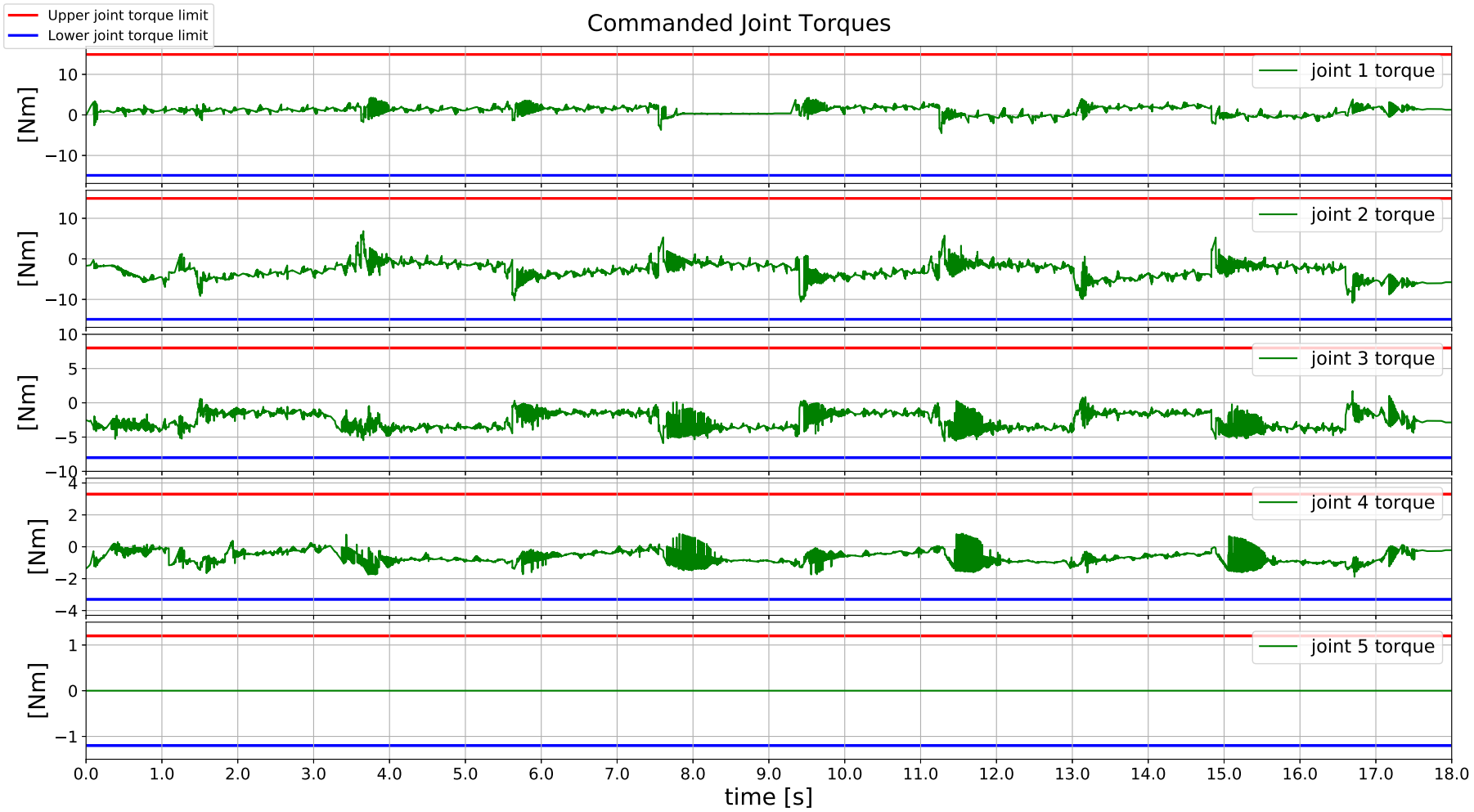

Figure 7.25: Computed joint torque commands, during the *following a pre-defined path* task execution on the real robot platform. The notable time-points are at 3.6 s, 5.6 s, 7.6 s, 9.4 s, 11.2 s, 13.0 s, 14.9 s, 16.6 s and 17.1 s

7.3 Cleaning a Surface

Experimental Setup

In this experiment, *cleaning a surface* task is performed in a simulation environment, only. The reason for not performing this type of evaluation on a real robot platform is the unavailability of a real robot system that is equipped with sensors for estimating external (contact) forces.

Here, the robot is commanded to move towards a pre-defined goal area by *i*) keeping its end-effector within the defined position-tube bounds and *ii*) maintaining a predefined speed throughout this tube, until it reaches the goal area. Additionally, the robot is required to - at the same time - follow a surface of unknown shape with its manipulated object, i.e. maintain proper alignment between the manipulated object and this surface. In this test, the initial robot configuration is chosen in such a way that the robot's end-effector is, at the start of the task execution, located above the test surface. Due to such configuration, before it starts the aforementioned (i.e. the main) part of the *cleaning a surface* task, the robot must, first approach the surface and then, align with it. Here, for approaching the test surface (a table in this case), the robot action is characterized by a *guarded motion* [81], [89], i.e. the robot moves towards the surface and stops when its built-in force/torque sensor detects a contact force. This type of robot action is specified using the same task specification model that is introduced for the first use case (see section 6.1.2) and previously evaluated in section 7.1. On the other side, the initial alignment is defined by the behaviour where the robot exerts a certain force against the surface and at the same time, opposes the reaction moments generated on the manipulated object while keeping the zero tangential velocities. The task specification model for that particular robot action is proposed and presented in [81].

Nevetheless, the task specification created for the main part of the *cleaning a surface* test is presented in the following:

Task specification for the *cleaning a surface* task: Simulation environment

move compliantly {

- ${}^{T^M}X$: 0.03 [m/s], velocity-tolerance 0.003 [m/s]
- ${}^{T^M}Y$: 0 [m], position-tube 0.07 [m]
- ${}^{T^F}Z$: 0.03 [N], force-tolerance 0.003 [N]
- ${}^{T^F}aX$: 0 [Nm], moment-tolerance 0.005 [Nm]
- ${}^{T^F}aY$: 0 [Nm], moment-tolerance 0.005 [Nm]
- ${}^{T^M}aZ$: no specification

} until either:

- goal area reached, tolerance 0.003 [m]
- or ${}^{T^F}F_X$ / ${}^{T^F}F_Y > 1$ [N]
- or task time > 8 [s]

The experimental setup for this test is presented in figure 7.26. Here, the robot system is equipped with a planar rigid object for cleaning the test surface and the task consist of a single goal area. Here, the *motion task frame* (T^M) is fixed on the table, and its position and orientation do not change throughout the task execution. The main control parameters used in this experiment are summarized in table 7.5. Here, the position and orientation of the *motion task frame* (T^M) are expressed with respect to the robot's base frame. In this table, the *maximum commands* represent the maximum allowed *acceleration energy* and *force* setpoints for the robot's end-effector (for more details see section 6.3.2).

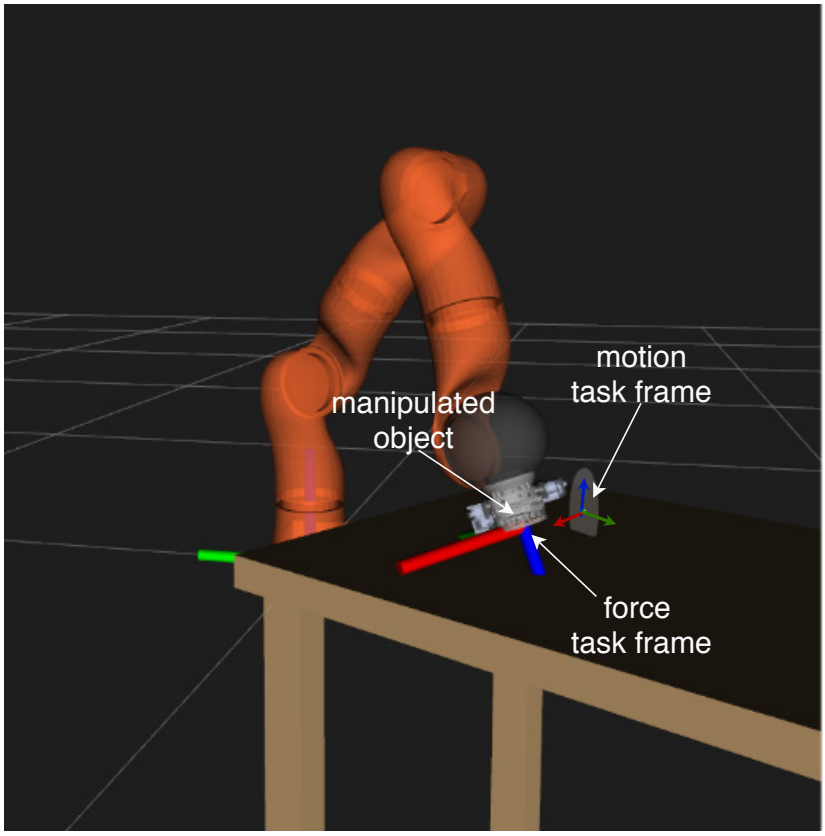

Figure 7.26: Visualization of the KUKA LWR 4 model, its initial configuration and a test surface (table) for the *cleaning a surface* task execution, in the simulation environment. Here, the simulation software [90] has generated a virtual goal-area (based on the task-defined bounds), to enhance the visualization for this task execution. The virtual goal-area is depicted with gray color. For each frame presented in the figure, red color stands for X axis, green color for Y axis and blue color for Z axis.

Parameter	**Value**
Initial joint configuration	1.12, 0.66, -0.13, -2.19, 1.64, 0.12, 0.01 [rad]
Force task frame location	Bottom center of the manipulated object
Motion task frame position	X: -0.009, Y: -0.454, Z: 0.120 [m]
Motion task frame orientation	$\mathrm{R} = \begin{bmatrix} 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.707107 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ [m]
Prediction horizon	2.5 s
Maximum commands	X: 80.0 $[\frac{Nm}{s^2}]$, Y: 70.0 $[\frac{Nm}{s^2}]$, Z: 300.0 $[N]$ aX: 30.0 $[Nm]$, aY: 30.0 $[Nm]$
Control loop frequency	770 Hz
Gravity compensation	Enabled

Table 7.5: Main control parameters for the *cleaning a surface* experiment in the simulation environment.

Results

Results from this experiment are depicted in figures 7.27, 7.28, 7.29, 7.30, 7.31, 7.32 and 7.33. Here, the reference and measured velocity and positions are defined with respect to the *motion task frame* (T^M), while the reference and measured force and moments are defined with respect to the *force task frame* (T^F).

The aforementioned control graphs describe robot behaviour that corresponds to the main part of the *cleaning a surface* test. More specifically, the presented robot data corresponds to the time period that started after the finite state machine (FSM), which is part of the control architecture developed in section 6.3.2, completed *APPROACH_SURFACE* and *ALIGN_WITH_SURFACE* states and from them, transitioned to the *START-TO-CRUISE* state. Nevertheless, from the first two above-presented control graphs, we can observe that the end-effector's position is initially outside of the tube bounds. Here, in the *START-TO-CRUISE* state, the robot is commanded to, at the same time, *i*) move towards the task-specified tubular region (in the linear Y direction of the motion task frame), *ii*) maintain the zero tube-velocity (in the linear X direction of the motion task frame) and *iii*) maintain alignment with the surface while applying the task-defined normal force (in the linear Z direction of the force task frame). Once the robot has reached the tube, at a time around 1.5 s (see figures 7.27 and 7.28), the FSM makes the transition to the *CRUISE_THROUGH_TUBE* state. From this point on, the robot is cruising through the tube with the task defined velocity and continues to maintain alignment with the surface while applying the task-defined normal force. However, once the robot reaches the goal area, at the 8.0 s time (see figure 7.32), the FSM switches to the *STOP_MOTION* state and the task is completed. Note that, in this case, the FSM does not consider *TRANSITION_PATH_SECTION* state throughout task execution since that task consists of only one goal area.

The graph presented in figure 7.27 reports the response of the main control loop for the linear X-direction, i.e. how well the control architecture developed in section 6.3.2 tracks the changing speed reference. Here, the first row depicts the measured and desired robot velocities, together with the upper and lower velocity-tolerances (defined in the task specification). The second row presents the computed velocity error, i.e. the difference between the measured and desired velocity values, at each time step (for more details on this error calculations, see figure 6.3). Last two rows in this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals. The presented control graph shows that the developed control architecture has not achieved sufficiently accurate and smooth tracking of the changing velocity reference. More specifically, the task-defined velocity tolerance of 0.003 [m/s] has not been satisfied throughout the execution. The factor that influences this behaviour is the continuously changing friction force that exists due to the contact between the manipulated object and the table. This disturbance opposes the robot from cruising throughout the tubular area and as it changes frequently, the ABAG controller is not able to adapt its inner signals on time to achieve a more stable response. This behaviour can be observed in the values of the ABAG's gain signal. We can see that this signal and thus, the command output u, oscillate, i.e. they are not able to converge to a steady value

in most parts of the task. Nevertheless, the origin of such frequently changing friction force is actually the frequently changing normal force that is exercised on the surface by the manipulated object. Moreover, at a time-period between 2.7s and 3.0s, we can clearly observe the effect of this force on the robot's tube-velocity. More specifically, for this period, figure 7.29 shows that the robot's normal force (in the linear Z direction of the force task frame) had a low and steady value. From figure 7.27 we see that the robot's tube-speed, at the same time, has converged to a steady and accurate value. In conclusion, these observations give us an insight that the control of the robot's tube-speed is, in turn, affected by the control of the robot's contact forces.

The control graph presented in figure 7.28 reports the response of the main control loop for the linear Y direction. More specifically, it shows how well the developed control architecture keeps the robot's end-effector within the defined position-tube bounds. The first row depicts the measured robot position together with the upper and lower position-tube tolerances. The second row presents the computed position error which is, in this case, defined by the difference between the *predicted* and desired position values, at each time step (for more details on this error calculations, see section 6.3.2). The last two rows of this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals. Even though the control of robot's motion in the linear X direction was not smooth and very accurate, the presented control graph in figure 7.28 shows that the developed control architecture has met the predefined thresholds and achieved stable tracking of the task-defined position tube for the linear Y direction, without any overshoots from these bounds.

The control graph presented in figure 7.29 reports the response of the main control loop for the linear Z direction, i.e. how well the control architecture developed in section 6.3.2 tracks the contact normal force reference. Here, the first row depicts the measured and desired robot normal force, together with the upper and lower force-tolerances (defined in the task specification). The second row presents the computed force error, i.e. the difference between the measured and desired force values, at each time step (for more details on this error calculations, see figure 6.12). Last two rows in this control graph depict the behaviour of the ABAG controller, i.e. its adapted signals. From this control graph, we can observe that the developed control architecture has not achieved sufficiently accurate and smooth tracking of the normal force reference. More specifically, the task-defined force tolerance of 0.003 [N] has not been satisfied throughout the execution. The factor that influences this behaviour is the low quality of the *force* feedback, provided by the simulation framework that is used for this experiment. More specifically, it has been observed that the simulated force/torque sensor produces very noisy force estimates. For that reason, it was necessary to select a very high value (i.e. 0.95) for the low-pass filtering factor α (presented in section 5.2), which is used by the ABAG controller in this DOF (see the third row of 7.29 graph).

The control graphs presented in figures 7.30 and 7.31, report the response of the main control loop for the angular X and angular Y directions, respectively. More specifically, these control graphs show how well the developed control architecture

regulates the reaction moments that act on the manipulated object (about X and Y axes of the force task frame). The first row, in both graphs, depicts the measured reaction moments together with the upper and lower task-defined tolerances. The second row (in both graphs), presents the regulation error which is, for each of these two DOFs, computed by *tracking* the directions of the linear forces exerted on the surface [81]. More specifically, for the angular X direction the computed error is $\arctan(-{}^{T^F}F_y, {}^{T^F}F_z)$, while for the angular Y direction the computed error is $\arctan(-{}^{T^F}F_x, {}^{T^F}F_z)$. Here, the ${}^{T^F}F_x, {}^{T^F}F_y, {}^{T^F}F_z$ force measurements are produced by the force/torque sensor that is mounted on the robot's end-effector. For more details on this type of regulation error calculation see [81]. Nevertheless, the reason for employing this type of error calculation, instead of directly computing the reaction-moment error presented in figure 6.12 (section 6.3.2) is the even lower quality of the *moment* measurements produced by the aforementioned force/torque sensor, compared to the *linear force* measurements. The last two rows of these control graphs depict the behaviour of the ABAG controllers, i.e. their adapted signals. The presented control graphs show that the developed control architecture has achieved sufficiently accurate and stable regulation of the reaction moments, for both angular X and angular Y directions, with minor overshoots over the task-defined tolerances.

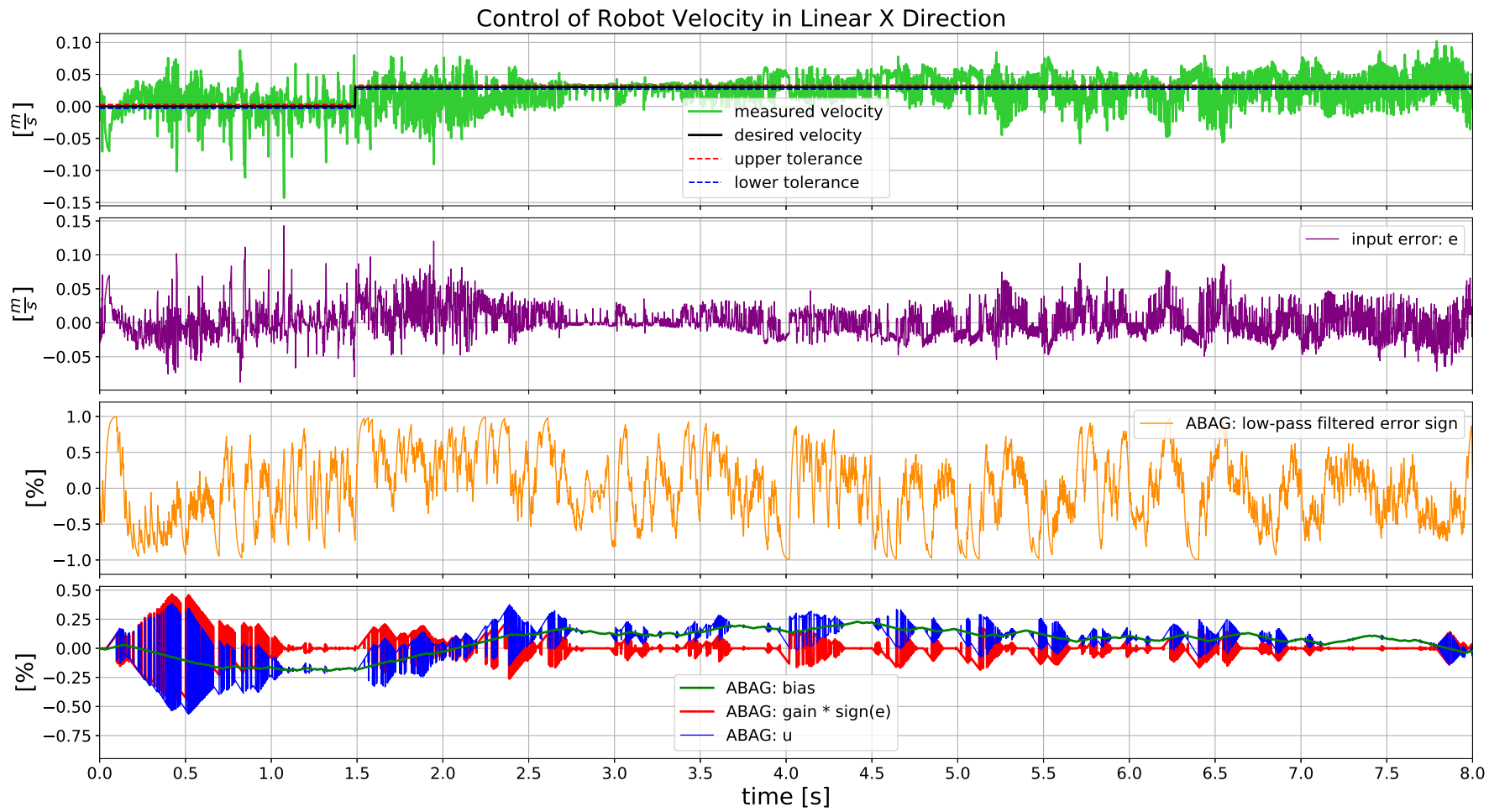

Figure 7.27: Response of the main control loop for the linear X-direction, during the *cleaning a surface* task execution in the simulation environment. Here, the measured and reference velocity values are expressed with respect to the motion task frame T^M. The notable time-points are at 1.5 s, 2.7 s and 3.0 s.

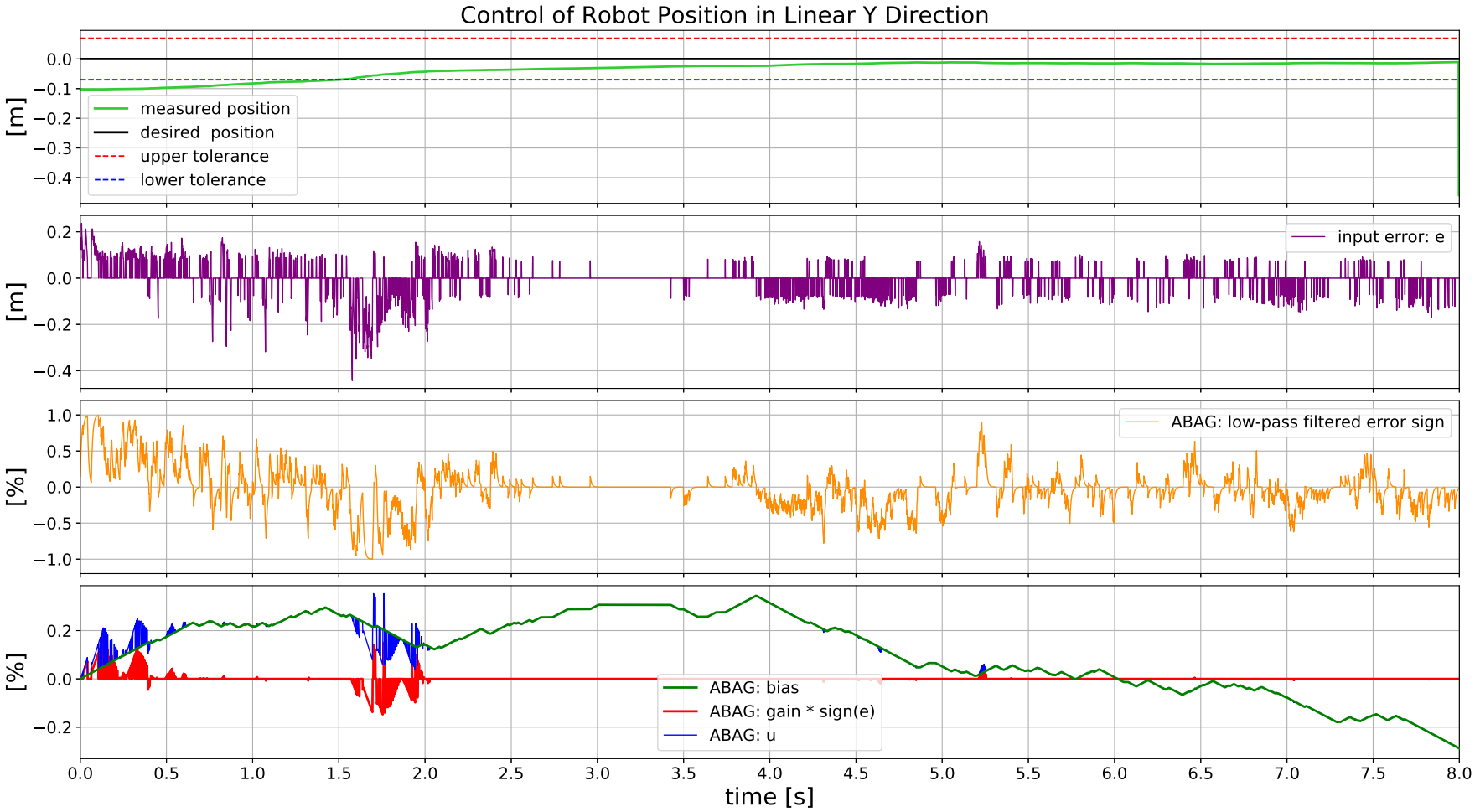

Figure 7.28: Response of the main control loop for the linear Y-direction, during the *cleaning a surface* task execution in the simulation environment. Here, the measured and reference position values are expressed with respect to the motion task frame T^M.

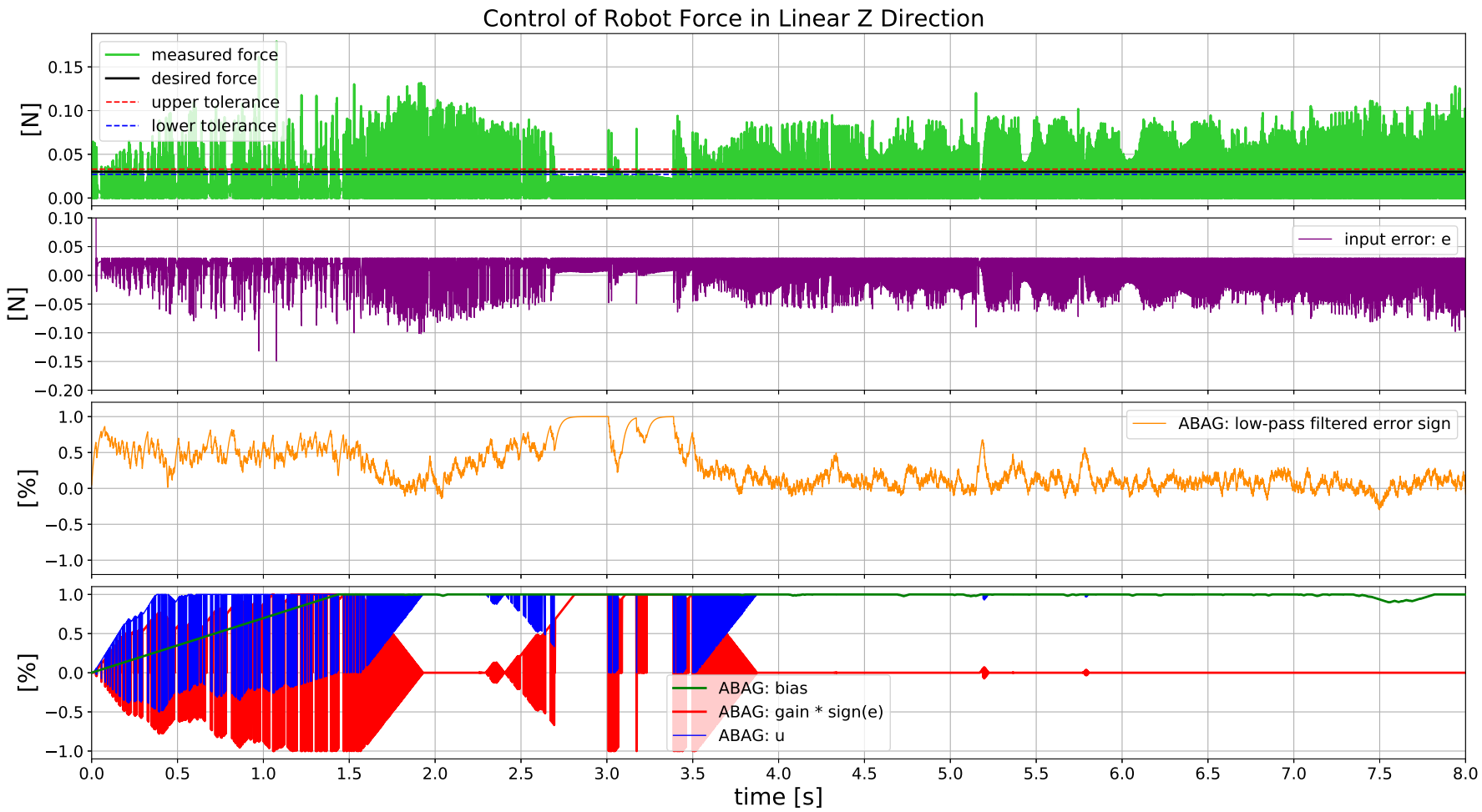

Figure 7.29: Response of the main control loop for the linear Z-direction, during the *cleaning a surface* task execution in the simulation environment. Here, the measured and reference force values are expressed with respect to the force task frame T^F. The notable time-points are at 1.5 s, 2.7 s and 3.0 s.

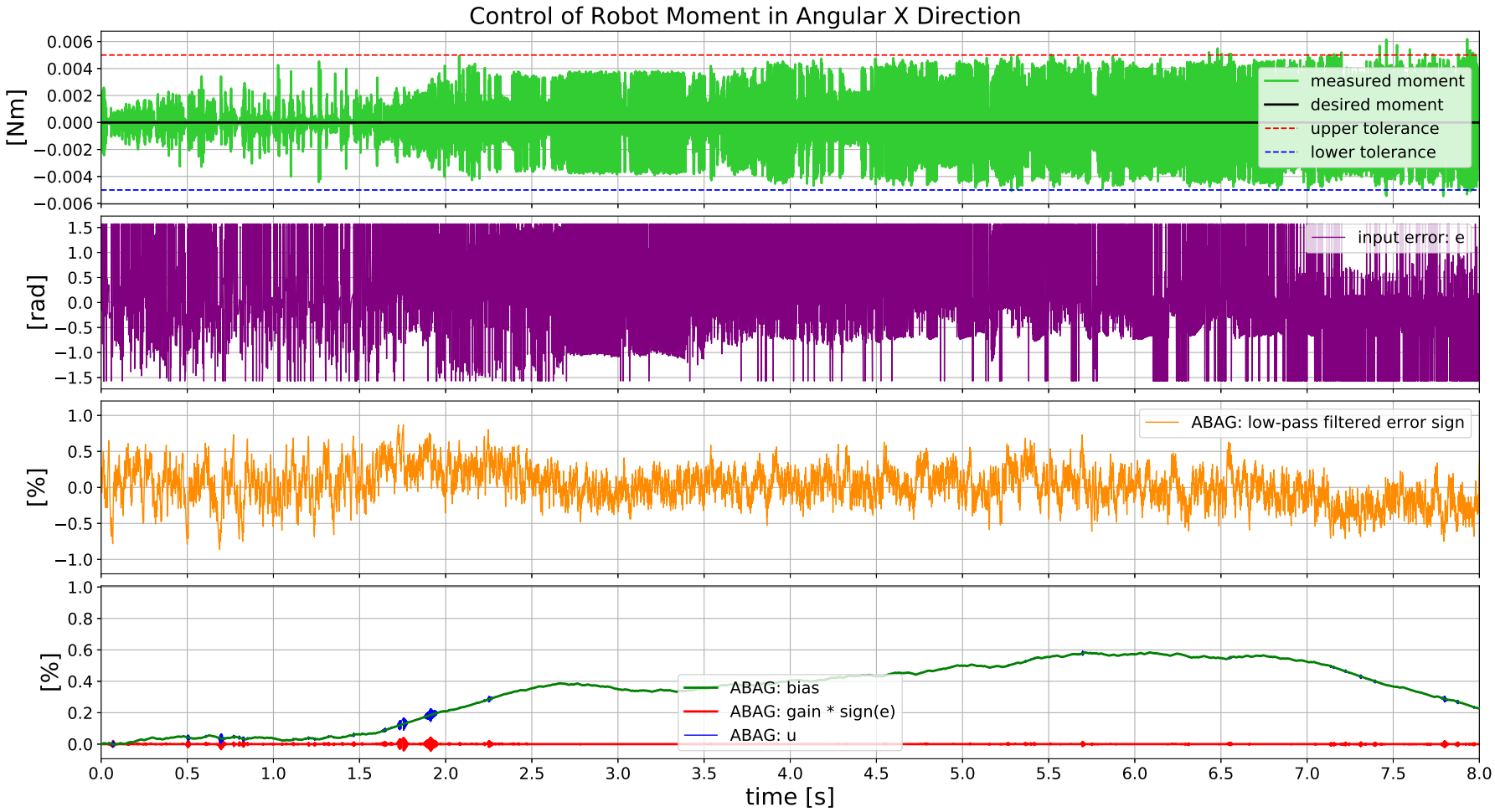

Figure 7.30: Response of the main control loop for the angular X-direction, during the *cleaning a surface* task execution in the simulation environment. Here, the measured and reference moment values are expressed with respect to the force task frame T^F.

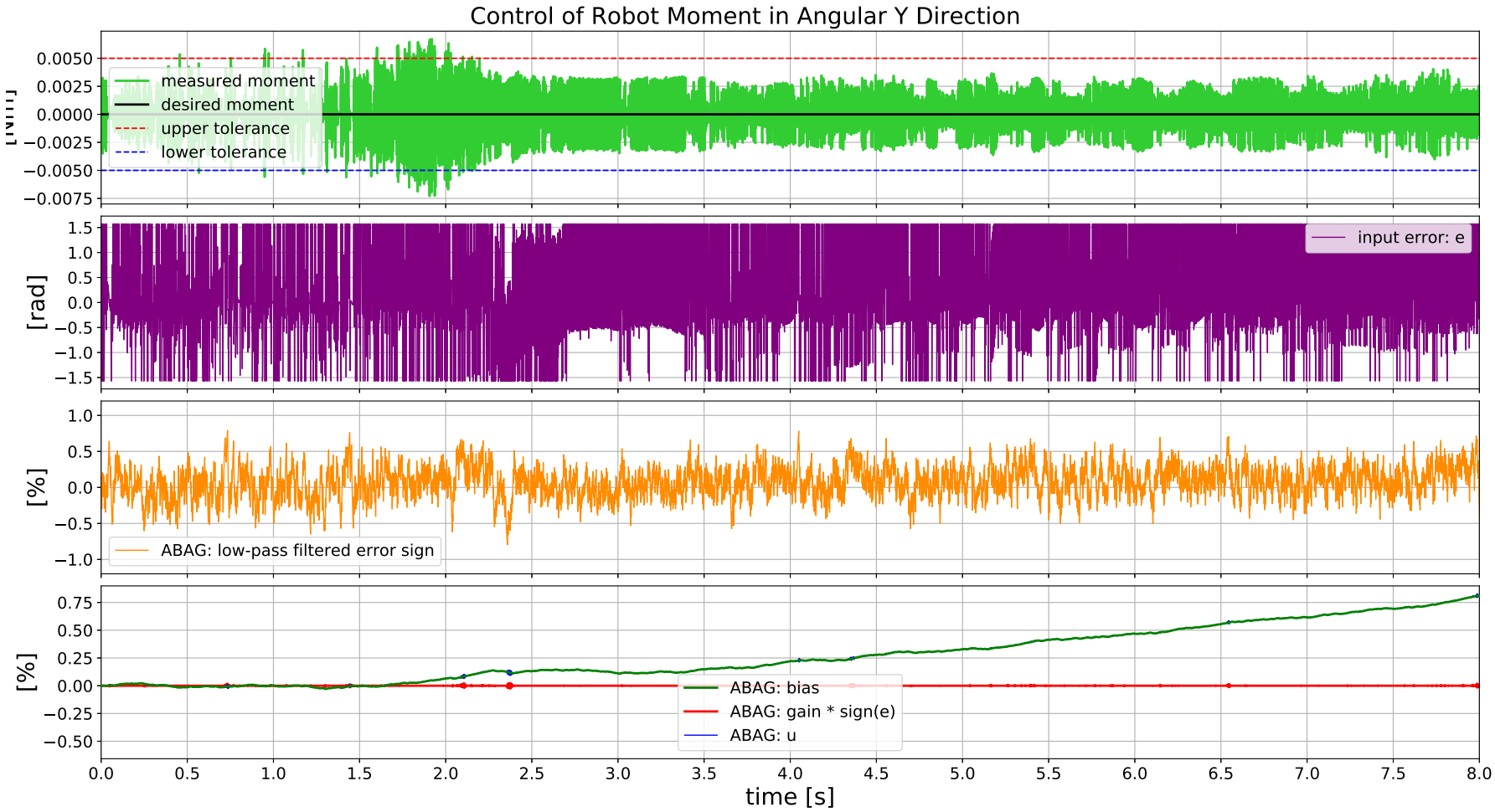

Figure 7.31: Response of the main control loop for the angular Y-direction, during the *cleaning a surface* task execution in the simulation environment. Here, the measured and reference moment values are expressed with respect to the force task frame T^F.

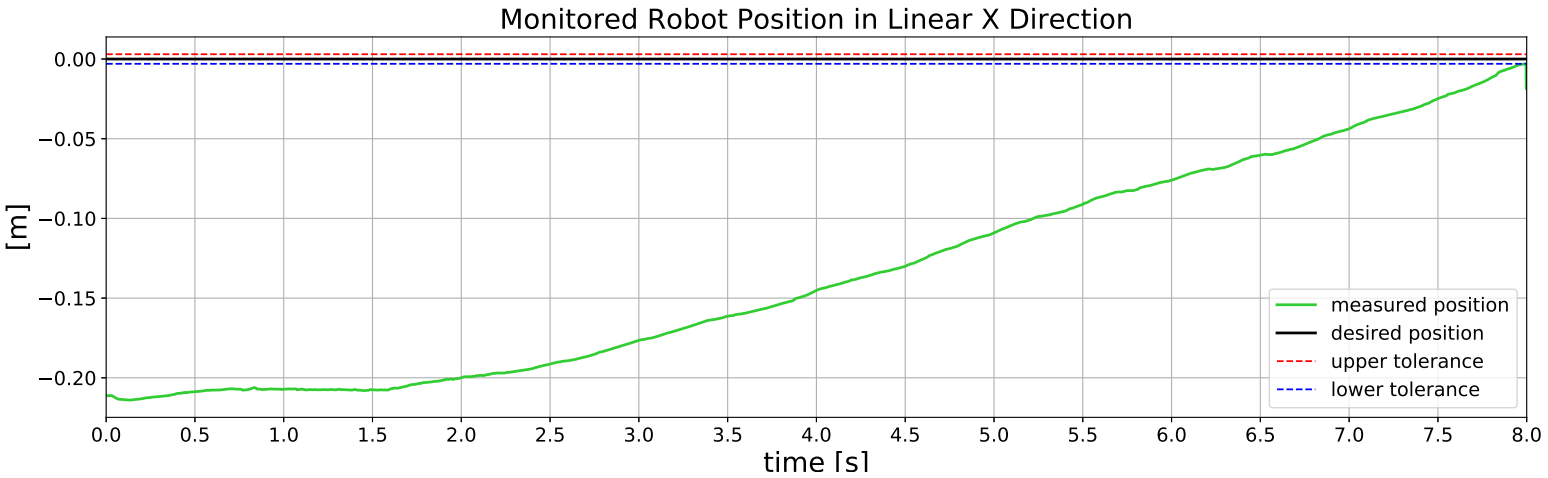

Figure 7.32: Resulting robot's position in the linear X direction, during the *cleaning a surface* task execution in the simulation environment. Here, the measured position and goal-area values are expressed with respect to the motion task frame T^M.

In figure 7.33, the computed joint torque commands are depicted, together with the upper and lower saturation limits, for each robot's joint. These saturation limits are enforced in the dynamics computations of this control architecture, i.e. in line 26 of **Algorithm** 2. From this figure, we can observe that the robot, once it has reached the tubular area and a sufficiently high tube-velocity (at a time around 2.0s), has executed the remaining part of the task by using a very small percentage of the maximum joint torques. This behaviour occurs due to the fact that the Popov-Vereshchagin solver (while resolving the *instantaneous* dynamics) is exploiting the already existing forces in nature, such as, in this case, gravitational force to achieve the desired robot state. Furthermore, actuation in some of the joints, such as actuation in fifth and seventh, was not required at all, during the large part of the task execution.

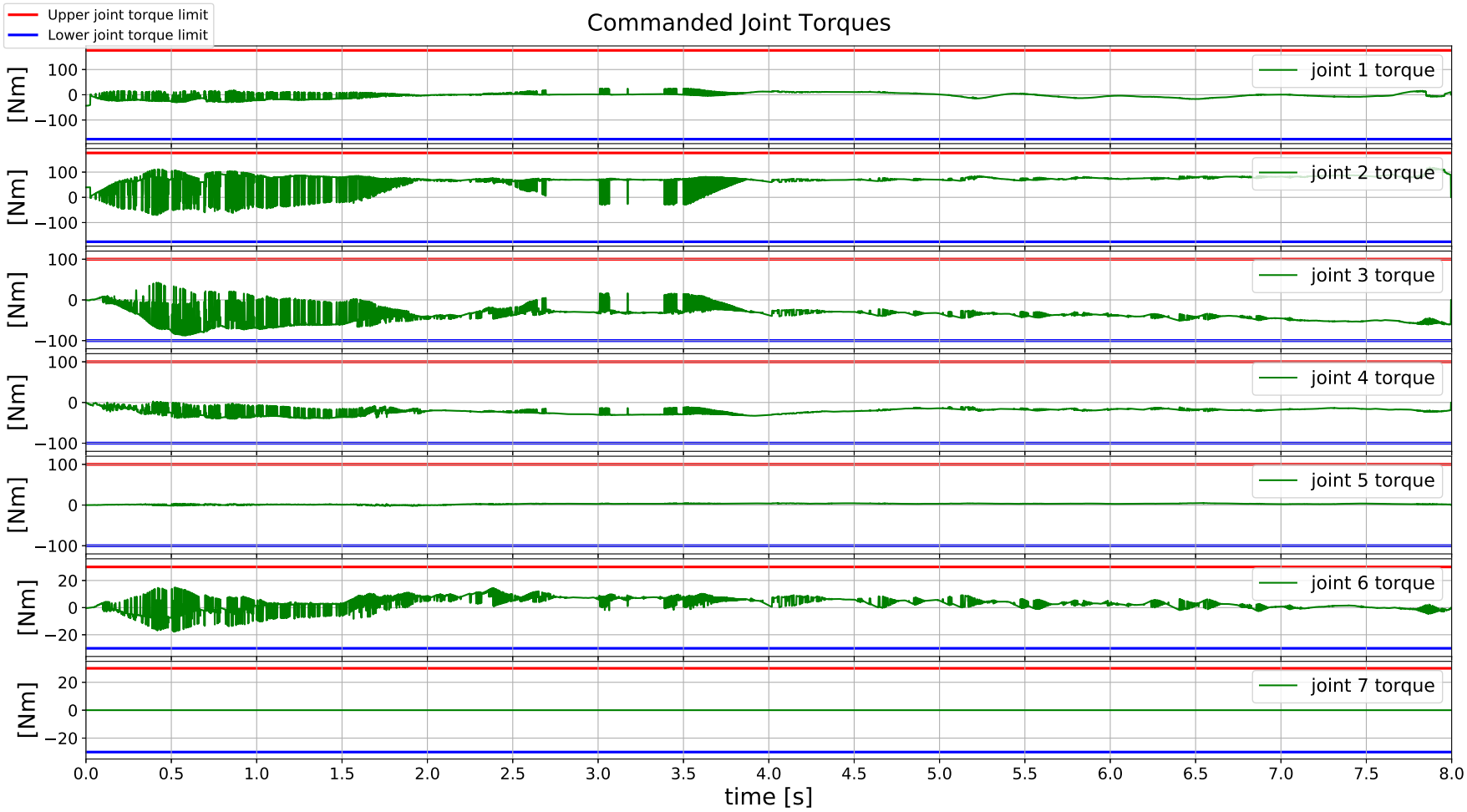

Figure 7.33: Computed joint torque commands, during the *cleaning a surface* task execution in the simulation environment. The notable time-points are at 2.7 s and 3.0 s.

8

Conclusion and Future Work

In this work, we created a lazy control strategy that relaxes several aspects of robot control and enables the correct execution of different robot tasks. Task specification models and control architectures that address the challenges presented in section 1.2 are developed.

This work provides a detailed review on how the previously developed acceleration constrained hybrid dynamics (ACHD) solver [2] and Adaptive Bias Adaptive Gain (ABAG) [6] controller can be used, and their features be exploited, for deriving different robot control architectures. Moreover, in this work, the solver's interface is extended with the *artificial* Cartesian force and feed-forward joint torque task drivers. Furthermore, to support these two building blocks in enabling the correct execution of the tasks presented in section 1.2, additional components have been developed. More specifically, components such as *i*) error functions for computing control deviations in different task directions, *ii*) prediction units that provide estimates of future robot states, *iii*) finite state machines that monitor execution of the specified tasks and alternate different control modes, and finally *iv*) an estimator unit that serves for estimating an unknown load. Finally, to model the desired motions and forces necessary for the execution of the aforementioned tasks, a modified version of *Task Frame Formalism* (TFF) [81], [82] was developed.

To demonstrate the feasibility of the proposed lazy control approach, an experimental evaluation was performed in a simulation environment and on a real robot platform. For the first two use cases presented in section 1.2, the experimental results have shown that the developed control architecture is able to provide sufficiently accurate and stable tracking of the task-specified constraints and tolerances, for both simulated and real robot. Evaluation of the proposed control approach for the task presented in the third use case was performed in a simulation environment. The results have shown that the developed control architecture, in some of the task directions, was not able to achieve completely accurate and smooth tracking of the task-defined constraints and tolerances. The reason for such degraded performance (in those task directions) was the low-quality force feedback, i.e. measurement provided by the simulation framework that was used for this experiment.

The rest of this section outlines the future work goals, i.e. proposes several improvements to the developed control strategy.

Extending future state estimations:

- In this work, for completing the control feedback, the prediction component is performing computations of future robot Cartesian poses by integrating the measured robot's Cartesian velocities. However, this component and thus, the control feedback can be extended and improved with predictions of the future robot's Cartesian velocities. This additional information can be simply derived from an output of the ACHD solver namely, the resultant spatial accelerations $\ddot{X}$ (computed for each segment in the kinematic chain). Nevertheless, this type of future robot state estimation can be improved even more by fusing the aforementioned model-based information with IMU data, if available.

- Currently, in the FSM, the information about robot's safety is derived based on *i*) the measured joint positions q and joint velocities $\dot{q}$, and *ii*) the resultant joint accelerations $\ddot{q}$ that are computed by the Popov-Vereshchagin solver. More specifically, this safety component is exploiting the already available *forward dynamics* calculations to estimate the future robot joint states. However, safety monitoring can be improved by also taking into account future robot's Cartesian states; to infer about the environmental impact on the robot's safety as well. For providing this type of state estimation, the above-mentioned (extended) prediction component can be re-used and combined with additional sensor information, such as perception data to enable broader and smarter safety monitoring.

Extending task specification model

- Currently, for each motion goal area defined in the task, the existing task specification model considers only one *motion frame*, with respect to which both position-tube and orientation-tube constraints are specified. However, this task specification model can be extended such that, it considers two *motion task frames*, one with respect to which the position-tube constraints will be specified and one with respect to which the orientation-tube constraints will be specified. In this way, the task specification model is even less constraining, i.e. it gives a user more freedom in designing the robot tasks.

- The developed task specification models, for the second and third use case, allow a user to specify task constraints for each waypoint independently. However, these task specification models and overall control architectures can be extended to also take into account the task constraints that explicitly define how the robot should behave while transitioning from one path section to another.

Improving estimation of the contact forces and moments: Currently, the developed control architecture for the *cleaning a surface* task, takes raw contact force and moment measurements into account as part of the control feedback. However, the estimate of these states and thus, the overall force control can be improved by employing some of the existing *state of the art* filtering methods [96] in the control computations.

References

[1] E. P. Popov, A. F. Vereshchagin, and S. L. Zenkevich, "Manipulyatsionnye roboty: Dinamika i algoritmy", *Nauka, Moscow*, 1978.

[2] A. F. Vereshchagin, "Modelling and control of motion of manipulation robots", *Soviet Journal of Computer and Systems Sciences*, vol. 27, pp. 29–38, 1989.

[3] A. Shakhimardanov, "Composable robot motion stack: Implementing constrained hybrid dynamics using semantic models of kinematic chains", PhD thesis, KU Leuven, 2015.

[4] D. Vukcevic, S. Schneider, and P. G. Plöger, "Extending a constrained hybrid dynamics solver for energy-optimal robot motions in the presence of static friction", Hochschule Bonn-Rhein-Sieg, Department of Computer Science, Tech. Rep. 03-2018, 2018, p. 66.

[5] S. Schneider and H. Bruyninckx, "Exploiting linearity in dynamics solvers for the design of composable robotic manipulation architectures", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[6] A. Franchi and A. Mallet, "Adaptive closed-loop speed control of BLDC motors with applications to multi-rotor aerial vehicles", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[7] E. Todorov, "Optimality principles in sensorimotor control", *Nature neuroscience*, vol. 7, p. 907, 2004.

[8] E. Todorov and M. I. Jordan, "Optimal feedback control as a theory of motor coordination", *Nature neuroscience*, vol. 5, p. 1226, 2002.

[9] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics", *arXiv preprint arXiv:1611.08268*, 2016.

[10] J. Baek, M. Jin, and S. Han, "A new adaptive sliding-mode control scheme for application to robot manipulators", *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 3628–3637, 2016.

[11] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[12] B. Bona and M. Indri, "Friction compensation in robotics: An overview", in *Decision and Control and European Control Conference IEEE, Spain*, 2005.

[13] J. Lee, H. Dallali, M. Jin, D. G. Caldwell, and N. G. Tsagarakis, "Robust and adaptive dynamic controller for fully-actuated robots in operational space under uncertainties", *Autonomous Robots*, vol. 43, pp. 1023–1040, 2018.

[14] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[15] H. Bruyninckx, E. Scioni, N. Hübel, F. Reniers, R. van de Molengraft, C. Schlegel, D. Stampfer, and A. Lotz, *Composable, explainable and verifiable robotics and cyber-physical systems-of-systems: meta models and best practices for resilient holonic architectures*. KU Leuven, 2019.

[16] L. I. G. Olascoaga, "Model predictive motion planning for robots", Master's thesis, Eindhoven University of Technology, 2015.

[17] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach", *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017.

[18] R. J. Griffin, G. Wiedebach, S. Bertrand, A. Leonessa, and J. Pratt, "Straight-leg walking through underconstrained whole-body control", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[19] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2008.

[20] R. Smits, E. Aertbelien, H. Bruyninckx, and A. Shakhimardanov, *Kinematics and Dynamics Library (KDL)*, Accessed: 2018-05-30, [Online] Available: http://www.orocos.org/kdl.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System", in *ICRA workshop on open source software*, 2009.

[22] M. L. Felis, "RBDL: an efficient rigid-body dynamics library using recursive algorithms", *Autonomous Robots*, vol. 41, pp. 495–511, 2017.

[23] M. Frigerio, J. Buchli, D. G. Caldwell, and C. Semini, "RobCoGen: A code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages", *Journal of Software Engineering for Robotics (JOSER)*, vol. 7, no. 1, pp. 36–54, 2016.

[24] N. Mansard and F. Valenza, *Pinocchio library*, Accessed: 2017-07-25, [Online] Available: https://stack-of-tasks.github.io/pinocchio/.

[25] Joint Japanese French Robotics Lab, *JRL-RBDyn library*, Accessed: 2017-07-25, [Online] Available: https://jrl-umi3218.github.io/RBDyn/doxygen/HEAD/index.html.

[26] I. Cervantes and J. Alvarez-Ramirez, "On the PID tracking control of robot manipulators", *Systems and Control Letters*, vol. 42, pp. 37–46, 2001.

[27] M. Copejans, "Adaptive control with a new constrained dynamics algorithm on the KUKA youBot robot", Master's thesis, KU Leuven, 2014.

[28] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the HRP-2 humanoid", in *IEEE International Conference on Intelligent Robots and Systems*, 2015.

[29] F. Ficuciello, A. Romano, L. Villani, and B. Siciliano, "Cartesian impedance control of redundant manipulators for human-robot co-manipulation", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[30] A. Albu-Schäffer and G. Hirzinger, "Cartesian impedance control techniques for torque controlled light-weight robots", in *IEEE International Conference on Robotics and Automation*, 2002.

[31] N. S. Nise, *Control Systems Engineering*. John Wiley & Sons, 2007.

[32] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[33] J.-J. E. Slotine and W. Li, *Applied nonlinear control*. 1991.

[34] S. Spurgeon, "Sliding mode control: A tutorial", in *European Control Conference (ECC)*, 2014.

[35] C. Edwards and S. Spurgeon, *Sliding mode control: theory and applications*. Crc Press, 1998.

[36] A. M. Dietrich, "Whole-body impedance control of wheeled humanoid robots", PhD thesis, Technische Universität München, 2015.

[37] C. Schindlbeck and S. Haddadin, "Unified passivity-based Cartesian force / impedance control for rigid and flexible joint robots via task-energy tanks", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[38] H. Sadeghian, L. Villani, M. Keshmiri, and B. Siciliano, "Task-space control of robot manipulators with null-space compliance", *IEEE Transactions on Robotics*, vol. 30, pp. 493–506, 2014.

[39] D. Q. Mayne, "Model predictive control: Recent developments and future promise", *Automatica*, vol. 50, pp. 2967 –2986, 2014.

[40] M. Giftthaler, M. Neunert, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli, "Automatic differentiation of rigid body dynamics for optimal control and estimation", *Advanced Robotics*, vol. 31, pp. 1225–1237, 2017.

[41] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach", *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017.

[42] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots", in *Humanoid Robots (Humanoids), IEEE-RAS International Conference on*, 2013.

[43] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms", in *Robotics: Science and Systems (RSS)*, 2018.

[44] T. Kunz and M. Stilman, "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[45] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", in *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[46] 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, pp. 328–373, 2013.

[47] O. Khatib, L. Sentis, J. Park, and J. Warren, "Whole-body dynamic behavior and control of human-like robots", *International Journal of Humanoid Robotics*, vol. 1, 2004.

[48] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation", *IEEE Journal on Robotics and Automation*, vol. 3, pp. 43–53, 1987.

[49] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks", in *International Conference on Advanced Robotics*, 2009.

[50] R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter, "iTaSC: A Tool for Multi-Sensor Integration in Robot Manipulation", in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. 2008.

[51] K.-S. Chang and O. Khatib, "Operational space dynamics: Efficient algorithms for modeling and control of branching mechanisms", in *IEEE International Conference on Robotics and Automation.*, 2000.

[52] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments", in *Robotics and Automation, IEEE International Conference on*, 2006.

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

[54] O. Khatib, L. Sentis, and J.-H. Park, "A unified framework for whole-body humanoid robot control with multiple constraints and contacts", in *European Robotics Symposium*.

[55] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks", *IEEE Transactions on Robotics*, vol. 25, pp. 670–685, 2009.

[56] L. Saab, N. Mansard, F. Keith, J. Y. Fourquet, and P. Soueres, "Generation of dynamic motion for anthropomorphic systems under prioritized equality and inequality constraints", in *IEEE International Conference on Robotics and Automation*, 2011.

[57] L. Saab, O. Ramos, N. Mansard, P. SouÃĺres, and J. Y. Fourquet, "Generic dynamic motion generation with multiple unilateral constraints", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[58] N. Mansard, "A dedicated solver for fast operational-space inverse dynamics", in *IEEE International Conference on Robotics and Automation*, 2012.

[59] L. Saab, O. E. Ramos, F. Keith, N. Mansard, P. Soueres, and J.-Y. Fourquet, "Dynamic whole-body motion generation under rigid contacts and other unilateral constraints", *IEEE Transactions on Robotics*, vol. 29, 2013.

[60] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation", *The International Journal of Robotics Research*, vol. 33, 2014.

[61] Lamiraux, Florent and Stasse, Olivier and Mansard, Nicolas, *Stack of Tasks*, Accessed: 2017-05-30, [Online] Available: http://stack-of-tasks.github.io/.

[62] R. Fletcher, "A general quadratic programming algorithm", *IMA Journal of Applied Mathematics*, vol. 7, pp. 76–91, 1971.

[63] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty", *The International Journal of Robotics Research*, vol. 26, pp. 433–455, 2007.

[64] W. Decre, R. Smits, H. Bruyninckx, and J. D. Schutter, "Extending iTaSC to support inequality constraints and non-instantaneous task specification", in *2009 IEEE International Conference on Robotics and Automation*.

[65] W. DecrÃľ, H. Bruyninckx, and J. D. Schutter, "Extending the iTaSC constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons", in *IEEE International Conference on Robotics and Automation*, 2013.

[66] G. Borghesan and J. De Schutter, "Constraint-based specification of hybrid position-impedance-force tasks", in *IEEE International Conference on Robotics and Automation*, 2014.

[67] C. Schindlbeck and S. Haddadin, "Unified passivity-based Cartesian force / impedance control for rigid and flexible joint robots via task-energy tanks", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[68] S. Farsoni, C. T. Landi, F. Ferraguti, C. Secchi, and M. BonfÃĺ, "Compensation of load dynamics for admittance controlled interactive industrial robots using a quaternion-based kalman filter", *IEEE Robotics and Automation Letters*, vol. 2, 2017.

[69] D. Kubus, T. Kroger, and F. M. Wahl, "Improving force control performance by computational elimination of non-contact forces/torques", in *IEEE International Conference on Robotics and Automation*, 2008.

[70] A. F. Vereshchagin, "Computer simulation of the dynamics of complicated mechanisms of robot-manipulators", *Engineering Cybernetics, 12(6)*, pp. 65–70, 1974.

[71] E. P. Popov, "Control of robots-manipulators", *Engineering Cybernetics*, 1974.

[72] H. Bruyninckx and O. Khatib, "Gauss' principle and the dynamics of redundant and constrained manipulators", in *IEEE International Conference on Robotics and Automation*, 2000.

[73] C. F. Gauß, "Über ein neues allgemeines Grundgesetz der Mechanik.", *Journal für die reine und angewandte Mathematik*, vol. 4, pp. 232–235, 1829.

[74] E. Ramm, "Principles of least action and of least constraint", *GAMM-Mitteilungen*, vol. 34, pp. 164–182, 2011.

[75] J. L. Lagrange, "Mécanique analytique", *Mallet-Bachelier*, vol. 1, 1853.

[76] R. Bellman, "On the theory of dynamic programming", *Proceedings of the National Academy of Sciences*, vol. 38, pp. 716–719, 1952.

[77] J. De Schutter, D. Torfs, H. Bruyninckx, and S. Dutré, "Invariant hybrid force/position control of a velocity controlled robot with compliant end effector using modal decoupling", *The International Journal of Robotics Research*, vol. 16, pp. 340–356, 1997.

[78] W. Langson, I. Chryssochoos, S. Rakovic, and D. Mayne, "Robust model predictive control using tubes", *Automatica*, vol. 40, pp. 125 –133, 2004.

[79] H. Wang and Y. Xie, "Prediction error based adaptive jacobian tracking for free-floating space manipulators", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, pp. 3207–3221, 2012.

[80] S. Mason, N. Rotella, S. Schaal, and L. Righetti, "An MPC walking framework with external contact forces", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

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

[82] M. T. Mason, "Compliance and force control for computer controlled manipulators", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[83] J. Luh, M. Walker, and R. Paul, "Resolved-acceleration control of mechanical manipulators", *IEEE Transactions on Automatic Control*, vol. 25, no. 3, pp. 468–474, 1980.

[84] L. Sciavicco and B. Siciliano, *Modelling and control of robot manipulators*. Springer Science & Business Media, 2000.

[85] S. Grazioso, G. Di Gironimo, and B. Siciliano, "From differential geometry of curves to helical kinematics of continuum robots using exponential mapping", in *International Symposium on Advances in Robot Kinematics*, 2018.

[86] F. S. Grassia, "Practical parameterization of rotations using the exponential map", *Journal of graphics tools*, vol. 3, no. 3, pp. 29–48, 1998.

[87] T. D. Laet, S. Bellens, R. Smits, E. Aertbelien, H. Bruyninckx, and J. D. Schutter, "Geometric relations between rigid bodies (part 1): Semantics for standardization", *IEEE Robotics Automation Magazine*, vol. 20, no. 1, pp. 84–93, 2013.

[88] M Klotzbücher, "Domain specific languages for hard real-time safe coordination of robot and machine tool systems", PhD thesis, Departement Werktuigkunde, Faculty of Engineering, KU Leuven, 2013.

[89] K. S. Pratt and R. R. Murphy, "Protection from human error: Guarded motion methodologies for mobile robots", *IEEE Robotics Automation Magazine*, vol. 19, pp. 36–47, 2012.

[90] Institute for Intelligent Systems and Robotics (ISIR) - Sorbonne University, *OROCOS/ROS Components for Light Weight Robots*, Accessed: 2019-10-21, [Online] Available: http://rtt-lwr.rtfd.io.

[91] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[92] Peter Soetens, *OROCOS Real-Time Toolkit*, Accessed: 2019-03-10, [Online] Available: http://www.orocos.org/rtt.

[93] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, "The KUKA-DLR Lightweight Robot arm-a new reference platform for robotics research and manufacturing", in *ISR (International Symposium on Robotics) and ROBOTIK (6th German Conference on Robotics)*, 2010.

[94] R. Bischoff, U. Huggenberger, and E. Prassler, "KUKA youBot - a mobile manipulator for research and education", in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[95] *youBot-Store*, Accessed: 2019-10-24, [Online] Available: http://www.youbot-store.com/developers/kuka-youbot-kinematics-dynamics-and-3d-model-81.

[96] T. Lefebvre, H. Bruyninckx, and J. De Schutter, *Nonlinear Kalman filtering for force-controlled robot tasks*. Springer, 2005.