SAPIENZA
UNIVERSITÀ DI ROMA

Optimization-Based Methods for Stable and Robust Motion Generation and Control in Mobile Robots

Sapienza University of Rome

Dottorato di Ricerca in Automatica, Bioingegneria e Ricerca Operativa - XXXVI Ciclo

Tommaso Belvedere

ID number 1705326

Advisor

Prof. Giuseppe Oriolo

Academic Year 2023/2024

Thesis defended on 29 May 2024
in front of a Board of Examiners composed by:

Prof.ssa Gaia Nicosia, Università di Roma Tre (chairman)

Prof. Andrea Pacifici, Università di Roma Tor Vergata

Prof.ssa Federica Pascucci, Università di Roma Tre

Optimization-Based Methods for Stable and Robust Motion Generation and Control in Mobile Robots
Ph.D. thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: belvedere@diag.uniroma1.it

Abstract

While Robotics is seeing an ever-increasing adoption in both industrial and service applications, truly autonomous robots are still far from being widespread. One of the limiting factors is in the availability of adaptable, performant and robust control methods to generate complex motions for such systems. At the moment, Model Predictive Control (MPC) stands out among the most promising techniques to fill this gap. In fact, with its ability to minimize a cost function while respecting a set of constraints that represent physical and operational limitations of a system, MPC is capable of outperforming classic approaches, and its predictive nature makes it adaptable to a large range of tasks.

This thesis presents a series of motion generation methods based on Model Predictive Control with the aim of improving over the current methodologies in two aspects: the first being the treatment of systems that exhibit a non-minimum phase behavior and thus pose the challenge of generating motions that are *stable*; the second being the *robustness* against uncertainties in the model parameters, which will inevitably make the robot deviate from the planned trajectory.

The first part presents IS-MPC (Intrinsically Stable MPC), demonstrating its effectiveness in the stable inversion of non-minimum phase systems. We showcase the applications of IS-MPC in stabilizing balancing robots performing non-trivial navigation and loco-manipulation tasks, and in preventing the jackknife phenomenon in autonomous Tractor-Trailer vehicles.

The second part of the thesis addresses the problem of robustifying motions against parametric uncertainties, focusing on aerial robots. We make use of the recent notion of closed-loop sensitivity and explore its application for robust flight control in Quadrotors with an experimental validation. We also present a novel computationally efficient Robust MPC scheme, named ST-MPC (Sensitivity-aware Tube MPC), for controlling nonlinear systems affected by parametric uncertainties, demonstrating its effectiveness through an extensive simulation campaign.

The contributions of this thesis extend to both stability and robustness in motion generation, offering valuable insights and practical applications across diverse robotic systems.

Acknowledgments

As I reflect on this thesis, I am reminded of the many people who have shared this journey with me in various ways.

I am deeply grateful to Prof. Giuseppe Oriolo and Prof. Leonardo Lanari for their invaluable guidance on my path to becoming a researcher. I also extend my heartfelt thanks to Dr. Paolo Robuffo Giordano for welcoming me into his team and engaging me in so many stimulating discussions.

I have been fortunate to meet many individuals who have made this experience truly enjoyable. I am especially thankful to everyone at the Robotics Lab in Rome, particularly Michele, Nicola, Filippo, and Spyros, for the wonderful times we shared and the pleasure of working together. Likewise, I am indebted to the Rainbow Team in Rennes, which became a second home to me. There are too many people to thank individually, but I would like to mention Nicola, Salvatore, Esteban, Marco, Maxime(s), and Pierre, among others.

Finally, I cannot overstate the unwavering support and love I have always received from my family throughout these years.

List of Symbols

Unless stated otherwise or evident from context, throughout this thesis we adhere to the following notation.

$\mathbb{I}_j^k = \{i \in \mathbb{N} \mid j \leq i \leq k\}$	Set of indices from j to k
a	Scalar
$\boldsymbol{a}$	Vector
$\boldsymbol{A}$	Matrix
A_{ij}	i-th row, j-th column element of $\boldsymbol{A}$
$\boldsymbol{a}(t)$	Time-varying vector
$\boldsymbol{a}_k = \boldsymbol{a}(t_k)$	Vector $\boldsymbol{a}$ evaluated at time t_k
$\boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times \boldsymbol{b}$	Skew-symmetric cross product
$\boldsymbol{a} \leq \boldsymbol{b}$	Element-wise inequality between the two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$(\boldsymbol{a}, \boldsymbol{b})$	Vertical concatenation of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{I}_n$	Identity matrix of dimension $n \times n$
$\mathbf{1}_n$, $\mathbf{0}_n$	n-dimensional (column) vector of ones/zeros
$\mathbf{1}_{n \times m}$, $\mathbf{0}_{n \times m}$	$(n \times m)$-dimensional matrix of ones/zeros
$\boldsymbol{A}^\dagger$	Moore-Penrose inverse of $\boldsymbol{A}$
$\lVert \cdot \rVert_{\boldsymbol{W}}$	2-norm weighted by the matrix $\boldsymbol{W}$
$\boldsymbol{q} = (q_w, q_x, q_y, q_z)^T$	Quaternion
$\boldsymbol{q}_\mathbf{v} = (q_x, q_y, q_z)^T$	Vector part of the quaternion
$\otimes$	Quaternion product's operator
$\lVert \boldsymbol{q}_a - \boldsymbol{q}_b \rVert = \lVert (\boldsymbol{q}_a \otimes \boldsymbol{q}_b^{-1})_\mathbf{v} \rVert$	Error quaternion norm

Contents

Chapter 1

Introduction

Over the past decades, the Robotics field has experienced a series of advancements that have allowed robots to become more widespread across different levels of our societies.

The first wave witnessed the use of *robot manipulators* for industrial purposes, particularly in the manufacturing industry. In these applications, robots were confined to enclosed spaces away from humans and lacked flexibility due to their limited sensing capabilities. Later, thanks to the introduction of force sensing technologies and related advancements in safety, there has been a paradigm shift towards collaborative robots, or cobots. Unlike their predecessors, cobots are designed to work alongside human operators, fostering a collaborative environment in which humans and robots share tasks (see Fig. 1.1).

In parallel, the proliferation of *mobile robots* has experienced unprecedented growth driven by the development of sensing, localization, and mapping, along with more advanced control techniques. Similar to robot manipulators, Wheeled Mobile Robots (WMR) were initially adopted in industrial and structured environments, such as in the logistics business. With recent improvements in safety standards and a decrease in the cost of development and components, WMR have emerged in many civil applications (the so-called service robots field), including last-mile delivery, stocktaking in grocery stores, vacuum cleaning, and autonomous mobility, among others.

Figure 1.1. Robot manipulators operating in an industrial environment. Note how in the traditional setting (left) robots are placed in a cage where humans are not allowed in during operation, while cobots (right) can safely operate alongside humans.

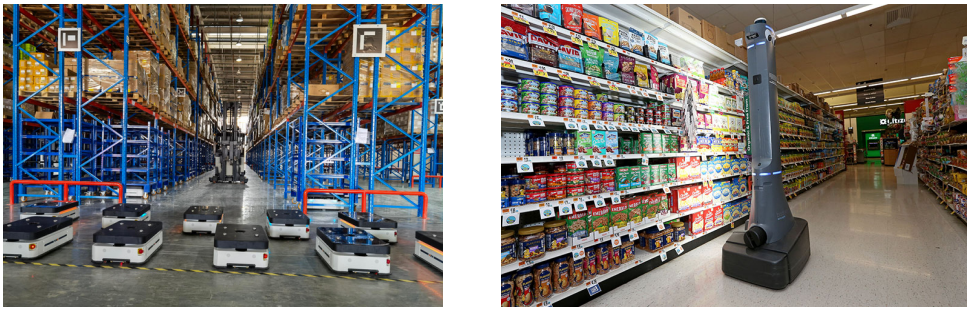

Figure 1.2. Wheeled mobile robots used for pallet transportation in logistics (left) and for stocktaking in grocery stores (right).

Figure 1.3. Examples of more complex articulated mobile robots. A quadruped robot equipped with a array of sensors for patrolling (left) and a fully actuated aerial manipulator (right).

In recent years, various types of more complex mobile robots have been developed for specialized applications (see Fig. 1.3). For instance, legged robots, including humanoids and quadrupeds, excel in navigating uneven terrain and adapting to human-centric environments better than simple WMR. On the other hand, WMR has evolved to incorporate robot manipulators, giving rise to Wheeled Mobile Manipulators (WMM), capable of expanding the robot workspace compared to fixed-base manipulators. Balancing WMR has also garnered attention in the past decade, merging the manipulation capabilities of WMM with increased agility. The field of aerial robots has also experienced substantial growth, with recent research focusing on aerial manipulators capable of interacting with the environment.

While each of these systems presents its own unique set of challenges, they share a commonality in facing the non-trivial problem of *motion generation*, i.e., that of synthesizing movements and related commands to perform some task.

One of the issues arising when controlling such complex dynamical systems is that of *instability*, which can be present when the system exhibits a non-minimum phase behavior, and which has to be addressed to achieve controlled and safe motions. In such cases, the performance in executing the desired task might be affected by the method used to generate stable motions, and no clear analytical solution might be available.

Another one of the many challenges that still need to be solved for an effective deployment of robots in real-world scenarios is that of *robustness* against the (unavoidable) uncertainties of the robot/environment models. In fact, any advanced planning and control algorithm relies on a suitable model of the robot/environment

for generating the motion plan or actions for realizing a task of interest. However, any model is only an approximation of the real world and, thus, planning/control schemes must exhibit some degree of robustness against model uncertainties.

Due to the possibly complex nature of the motion generation problem, arguably the most successful and widespread methods to perform motion generation are based on optimization. In fact, being able to solve these problems by designing appropriate cost functions and constraints, in conjunction with the availability of efficient numerical optimization algorithms, has proven to be one of the enabling factors in the development of solutions for complex articulated robots.

An idea that has proven to be extremely valuable has been that of combining trajectory optimization and feedback control in order to combine the capability of synthesizing complex motions of trajectory optimization with the reactiveness and robustness to disturbances of feedback control. This technique, that has seen enormous interest in recent decades also thanks to the continuous advancements in computing, is Model Predictive Control (MPC), a form of real-time optimization-based motion generation. Nowadays, MPC is a widely adopted control approach in robotics, with many examples of its application to Quadrotor control [1, 2], autonomous racing [3], WMR navigation and control [4, 5], and legged locomotion [6–8]. Its predictive nature, which leverages a *nominal* model of the robot/environment, and the possibility to explicitly handle constraints and optimization of performance indexes over the future evolution of the system, has made it an attractive choice for solving complex control problems. For instance, one of the main key advantages of MPC over traditional robot controllers is the ability to account for input constraints [2] so as to ensure the feasibility of the planned motion also in the presence of limited actuation. Yet, an effective use of MPC is subject to the capability of effectively dealing with the possible instability of the controlled system, and to the robustness to uncertainties despite the unavoidable use of a nominal model.

1.1 Contribution and overview

In this thesis, we address the real-time motion generation problem focusing on two distinct aspect arising in the proposed applications. For this reason, the manuscript is divided in two parts, the first being concerned with the problem of generating *stable* motions, while the second being focused on the problem of generating motions that are *robust* to uncertainties in the model parameters.

The majority of this thesis is centered around the use of Model Predictive Control, so Chapter 2 provides an overview of the essential components of optimization-based motion generation in the form of MPC. After some preliminaries on optimal control and constrained optimization, the focus shifts to MPC for real-time applications to nonlinear systems. Due to the strict time requirements to perform real-time control, we discuss in a general formulation the different approaches that will be used throughout this thesis to appropriately approximate the possibly too complex nonlinear optimization problem arising from the trajectory optimization performed in an MPC controller at each control instant. Specifically, we discuss the Linearized Time Varying MPC, in which the system is first linearized around a suitable auxil-

iary trajectory, and only then the Optimal Control Problem is formulated over this linearized system, and the Real-Time Iteration scheme which, being closely tied with the Sequential Quadratic Programming method, transforms the Optimal Control Problem into a finite-dimensional Nonlinear Program via direct transcription, and approximately solves the resulting problem through linearization.

In the first part of the thesis, we introduce IS-MPC, an Intrinsically Stable MPC scheme which is effective for stable inversion of non-minimum phase systems.

In more detail, Chapter 3 introduces the problem of controlling systems that exhibit a non-minimum phase behavior, discussing the zero dynamics and its relation with the stability of the system when forced to track a desired output. Then, we discuss the boundedness condition for the evolution of LTI systems, an anti-causal condition on the initialization of the unstable component of the state that depends on the future evolution of the forcing input. This condition will then be used to establish a stability constraint for the MPC controller — in the form of a terminal constraint on a subset of the state — that can be applied to nonlinear system using a suitable approximate linearization. The resulting IS-MPC method is then presented in the form of a tutorial with the application to a planar Wheeled Inverted Pendulum, before being adopted for different applications in the following two chapters.

In Chapter 4, we present a whole-body control architecture for the generation of stable task-oriented motions in Wheeled Inverted Pendulum (WIP) robots. Controlling WIP systems is challenging because the successful execution of tasks is subordinate to the ability to maintain balance. Our feedback control approach relies both on partial feedback linearization and MPC. The partial feedback linearization reshapes the system into a convenient form, while the MPC computes inputs to execute the desired task by solving a constrained optimization problem. Input constraints account for actuation limits and a stability constraint is in charge of stabilizing the unstable body pitch angle dynamics. The proposed approach is validated by simulations on an ALTER-EGO robot performing navigation and loco-manipulation tasks. The contents of this chapter have been originally presented in [9].

In Chapter 5 we discuss the application of IS-MPC to autonomous Tractor-Trailer vehicles, which are affected by jackknifing, a phenomenon that consists in the divergence of the trailer hitch angle and ultimately causes the vehicle to fold up. This phenomenon is particularly severe in the case of backward motions, where it appears also at low speeds. With reference to this specific context, we present a control method that drives the vehicle along generic reference Cartesian trajectories while avoiding the divergence of the hitch angle. This is obtained thanks to a feedback control law that combines two actions: a tracking term, computed using input-output linearization, and a corrective term, generated via IS-MPC. The successful performance of the proposed anti-jackknifing control is verified through simulations and experiments on a purposely built one-trailer prototype. To show the generality of the approach, we also apply and test the proposed method on a two-trailer vehicle. The contents of this chapter have been originally presented in [10].

The second part of the thesis is concerned with the problem of robustifying motions against parametric uncertainties. The application of choice is that of aerial robots (Quadrotors), although the method can be applied to generic nonlinear systems.

In Chapter 6, we introduce the notion of *closed-loop sensitivity* suitable for describing the effect of parametric uncertainties over the state and input trajectories of a system. This notion, which takes the label *closed-loop* due to the explicit modeling of the effect of the feedback action, can be used to robustify the system behavior in several ways. We provide an experimental validation of these concepts in the context of robust flight control for a quadrotor equipped with the popular PX4 controller. In particular, we assess how the optimization of the reference trajectory w.r.t. these sensitivity metrics and the use of *input tubes* (to guarantee the feasibility of the actuation constraints) can improve the closed-loop system performance against model uncertainties commonly affecting the quadrotor systems. To accomplish this, we present a series of experiments designed to validate our optimization approach on two distinct trajectories, with the primary aim of assessing its precision in guiding the quadrotor through the center of a window at relatively high speeds. The contents of this chapter have been originally presented in [11].

Chapter 7 then introduces a computationally efficient Robust MPC scheme for controlling nonlinear systems affected by parametric uncertainties in their models. The approach leverages the notion of *closed-loop sensitivity* and the associated ellipsoidal tubes of perturbed trajectories for taking into account online time-varying restrictions on state and input constraints. This makes the MPC controller "aware" of potential additional requirements needed to cope with parametric uncertainty, thus significantly improving the tracking performance and success rates during navigation in constrained environments. One key contribution lies in the introduction of a computationally efficient robust MPC formulation with a *comparable computational complexity* to a standard MPC (i.e., an MPC not explicitly dealing with parametric uncertainty). An extensive simulation campaign is presented to demonstrate the effectiveness of the proposed approach in handling parametric uncertainties and enhancing task performance, safety, and overall robustness. The versatility and efficiency of the proposed method make it therefore a valuable tool for real-time control of robots subject to non-negligible uncertainty in their models. The contents of this chapter have been submitted in [12].

Finally, Chapter 8 serves as a summary of the applications described in Chapters 4–7 and discusses a series of possible future research directions.

Chapter 2

The Optimization-Based Approach to Motion Generation

In this chapter, we review the general formulation of Optimal Control Problems that are aimed at motion generation. Moreover, we delve into some of the *direct methods* for solving such problems and discuss their use in the context of real-time control of trajectories through Model Predictive Control (MPC).

2.1 Preliminaries

Consider a continuous-time dynamical system with state $\boldsymbol{x}_c \in \mathbb{R}^{n_x}$ and control input $\boldsymbol{u}_c \in \mathbb{R}^{n_u}$ whose evolution is described by the system of first-order ordinary differential equations

$$\dot{\boldsymbol{x}}_c(t) = \boldsymbol{f}_c(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t), t), \quad \forall t \in [t_0, \infty), \tag{2.1}$$

with initial condition $\boldsymbol{x}_c(t_0) = \boldsymbol{x}_{c,0}$.

Being system (2.1) a controlled non-autonomous system, the evolution of $\boldsymbol{x}_c(t)$ starting from the initial condition, also named *state trajectory*, is governed by the *input trajectory* $\boldsymbol{u}_c(t)$.

Optimal Control deals with the problem of generating trajectories through the optimization of performance criteria, such as tracking error norm, energy consumption, or time required to execute the motion, among others. The desired behavior is encoded by a *cost functional* that assigns to each control $\boldsymbol{u}_c(t)$ a scalar *cost*

$$L(\boldsymbol{u}_c(t)) = \int_{t_0}^{t_f} \ell(\tau, \boldsymbol{x}_c(\tau), \boldsymbol{u}_c(\tau))d\tau + \ell_f(t_f, \boldsymbol{x}_c(t_f)),$$

where ℓ and ℓ_f, termed *running* and *final* cost respectively, are functions designed to have minimum value in correspondence of the desired behaviors, and t_f is the final time, which will be assumed fixed —- a common assumption in the context of real-time optimal control —- but which can in principle be optimized as well.

While the optimization of the performance criteria encoded by the cost function can be used to synthesize complex behaviors [13], the ability to constrain the state and input evolution is what enables modern optimization-based control to excel

when compared to traditional control methods. Many systems of interest are subject to actuation constraints, meaning that the control trajectory is bounded to a set $U \subset \mathbb{R}^{n_u}$, which can often be encoded via double-sided box constraint:

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_c \leq \boldsymbol{u}_{\max}, \tag{2.2}$$

with $\boldsymbol{u}_{\min}$ and $\boldsymbol{u}_{\max}$ encoding the lower an upper bounds for the control action.

Similarly, it can be desirable to impose limits on (part of) the state vector as well, e.g., to set the maximum longitudinal or lateral velocity of an autonomous vehicle for safety reasons, or to account for physical limitations of the system such as joint limits. In this case, it is possible to set lower and upper bounds $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ for the state, imposing the constraint:

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_c \leq \boldsymbol{x}_{\max}. \tag{2.3}$$

More in general, constraints can be used to encode complex behaviors such as avoiding obstacles, enforcing passivity, or avoiding self-collision in articulated robots, which can often be formulated through a possibly time-varying condition of the kind

$$\boldsymbol{g}(\boldsymbol{x}_c, \boldsymbol{u}_c, t) \leq \boldsymbol{0}. \tag{2.4}$$

The combination of these conditions, along with the optimization of the cost functional, defines the Optimal Control Problem (OCP):

$$\begin{cases} \underset{\boldsymbol{u_c}(\cdot)}{\text{minimize}} & \displaystyle\int_{t_0}^{t_f} \ell(\tau, \boldsymbol{x}_c(\tau), \boldsymbol{u}_c(\tau))d\tau + \ell_f(t_f, \boldsymbol{x}_c(t_f)) & & \text{(2.5a)} \\ \text{subject to} & \boldsymbol{x}_c(t_0) = \boldsymbol{x}_{c,0} & & \text{(2.5b)} \\ & \dot{\boldsymbol{x}}_c = \boldsymbol{f}_c(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t), t) & \forall t \in [t_0,\ t_f] & \text{(2.5c)} \\ & \boldsymbol{x}_{\min} \leq \boldsymbol{x}_c(t) \leq \boldsymbol{x}_{\max} & \forall t \in [t_0,\ t_f] & \text{(2.5d)} \\ & \boldsymbol{u}_{\min} \leq \boldsymbol{u}_c(t) \leq \boldsymbol{u}_{\max} & \forall t \in [t_0,\ t_f] & \text{(2.5e)} \\ & \boldsymbol{g}(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t), t) \leq \boldsymbol{0} & \forall t \in [t_0,\ t_f] & \text{(2.5f)} \\ & \boldsymbol{g}_f(\boldsymbol{x}_c(t_f)) \leq \boldsymbol{0} & & \text{(2.5g)} \end{cases}$$

In this OCP, the input trajectory $\boldsymbol{u}_c(t)$ is optimized to minimize the cost function (2.5a) while satisfying constraints (2.5b)–(2.5g). Among these constraints, it is always necessary to specify the initial state (2.5b) and the dynamics constraint (2.5c) linking the evolution of the input and state trajectories. Depending on the particular problem, the aforementioned box constraints (2.5d) and (2.5e) on the state and inputs can be introduced, as well as the general nonlinear (path) constraints (2.5f) and the terminal constraint (2.5g) acting on the final state $\boldsymbol{x}(t_f)$. Although in general (2.5f), (2.5g) can be made of equality and/or inequality constraints, here we prefer the inequality constrained formulation since it can easily incorporate equality constraints as well. In general, the objective function and the constraints are assumed to be at least twice continuously differentiable.

Note that the existence of an input trajectory $\boldsymbol{u}_c^*(t)$ solution to problem (2.5), i.e., the so-called *feasibility*, is not always guaranteed and the problem has to be designed to be feasible by construction or modified, for instance by relaxing the constraints, to make it feasible.

2.1.1 Examples

Depending on the specific application, OCP (2.5) can assume many different forms which differ in terms that are present in the cost function and/or the constraints. We propose two illustrative examples here for reference.

Point-to-point planning with obstacle avoidance

Consider the problem of steering a mobile robot from the initial state $\boldsymbol{x}_{c,0}$ to a final state $\boldsymbol{x}_f$ in the fixed time interval $[t_0, t_f]$. We may be interested in doing so while using a minimal amount of control effort to limit energy consumption, component wear, or to provide some intrinsic degree of smoothness to the resulting motion. To this end, one can minimize the integral of the squared 2-norm of the control input $\boldsymbol{u}_c$, providing a smooth convex running cost. Assuming that the environment is occupied by obstacles and that the robot has to keep a minimum clearance $\rho_{\min}$ to be safe, one can impose an inequality constraint on the distance $d(\boldsymbol{x}_c)$ between the robot and the closest obstacle to generate a safe motion. Additionally, one may impose input constraints to account for saturation. This can be summarized in the following OCP, providing the input trajectory that steers the system to the desired state:

$$\begin{cases} \underset{\boldsymbol{u}_c(\cdot)}{\text{minimize}} & \displaystyle\int_{t_0}^{t_f} \|\boldsymbol{u}_c(\tau)\|^2 d\tau \\ \text{subject to} & \boldsymbol{x}_c(t_0) = \boldsymbol{x}_{c,0} \\ & \dot{\boldsymbol{x}}_c = \boldsymbol{f}_c(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t), t) & \forall t \in [t_0,\ t_f] \\ & \boldsymbol{u}_{\min} \leq \boldsymbol{u}_c(t) \leq \boldsymbol{u}_{\max} & \forall t \in [t_0,\ t_f] \\ & d(\boldsymbol{x}_c(t)) \geq \rho_{\min} & \forall t \in [t_0,\ t_f] \\ & \boldsymbol{x}_c(t_f) = \boldsymbol{x}_f \end{cases}$$

Output tracking

Instead of specifying only a final state $\boldsymbol{x}_f$ and letting the state trajectory evolve with no other criteria, in many applications it is of interest for a function of the state $\boldsymbol{y}(t) = \boldsymbol{h}(\boldsymbol{x}_c(t))$ — often termed as *output* — to track a possibly time varying reference $\boldsymbol{y}_d(t)$. In this case, the performance criteria is to minimize the so-called tracking error, i.e., the norm of the difference between the actual and the desired output. If a *feedforward* input $\boldsymbol{u}_d(t)$ is available, its tracking error can also be incorporated in the cost function. In its most basic form the output tracking problem can be expressed as

$$\begin{cases} \underset{\boldsymbol{u}_c(\cdot)}{\text{minimize}} & \displaystyle\int_{t_0}^{t_f} \|\boldsymbol{y}(\tau) - \boldsymbol{y}_d(\tau)\|^2 + \|\boldsymbol{u}_c(\tau) - \boldsymbol{u}_d(\tau)\|^2 d\tau \\ \text{subject to} & \boldsymbol{x}_c(t_0) = \boldsymbol{x}_{c,0} \\ & \dot{\boldsymbol{x}}_c = \boldsymbol{f}_c(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t), t) & \forall t \in [t_0,\ t_f] \end{cases}$$

but it is clearly possible to include additional constraints. If the desired output can be followed exactly from the initial state $\boldsymbol{x}_{c,0}$, then the solution to the problem is trivially $\boldsymbol{u}_c(t) = \boldsymbol{u}_d(t)$. More often however, this kind of formulation is used

in the absence of a feedforward input $\boldsymbol{u}_d$, which has then to be generated. In other instances, if the system does not start from an initial condition for which $\boldsymbol{h}(\boldsymbol{x}_{c,0}) = \boldsymbol{y}_d(0)$, the solution to the problem will be a trajectory that converges to the desired one, according to the cost function. Finally, the desired trajectory might not feasible for the system dynamics, in which case the OCP would provide the best approximation of the desired output, according to the cost function.

2.2 Model Predictive Control

At its core, MPC revolves around the idea of making predictions about the future behavior of a system and using these predictions to determine the optimal control actions. Unlike traditional control methods that operate reactively, MPC considers a finite prediction horizon, allowing for the explicit incorporation of a prediction of the system dynamics and constraints into the control process. This forward-looking approach endows MPC with the ability to address complex, time-varying problems, making it particularly well-suited for applications in robotics.

More in detail, the MPC framework aims at formulating a controller that, at each time t_k and with a fixed rate f_t, computes the control action $\boldsymbol{u}_k$ solving an OCP of the type (2.5) from the current state $\hat{\boldsymbol{x}}_k$. Clearly, this provides a feedback action from the current state that allows the system to evolve in closed-loop, providing the disturbance rejection and reactive capabilities typical of feedback controllers. The problem is solved over a finite time interval $[t_k,\ t_k + T_c]$, with T_c being the so-called *control horizon*. As time passes, the "window" of time that is concerned in the trajectory optimization moves while keeping the same duration T_c following the *receding horizon* principle. Thus, the OCP solved at each control instant t_k is:

$$
\begin{cases}
\underset{\boldsymbol{u}_c(\cdot)}{\text{minimize}} & \displaystyle\int_{t_k}^{t_k+T_c} \ell(\boldsymbol{x}_c(\tau), \boldsymbol{u}_c(\tau))d\tau + \ell_f(\boldsymbol{x}_c(t_k + T_c)) & & \text{(2.6a)} \\
\text{subject to} & \boldsymbol{x}_c(t_k) = \hat{\boldsymbol{x}}_k & & \text{(2.6b)} \\
& \dot{\boldsymbol{x}}_c = \boldsymbol{f}_c(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t)) & \forall t \in [t_k,\ t_k + T_c] & \text{(2.6c)} \\
& \boldsymbol{g}(\boldsymbol{x}_c(t), \boldsymbol{u}_c(t)) \leq \boldsymbol{0} & \forall t \in [t_k,\ t_k + T_c] & \text{(2.6d)} \\
& \boldsymbol{g}_f(\boldsymbol{x}_c(t_k + T_c)) \leq \boldsymbol{0}, & & \text{(2.6e)}
\end{cases}
$$

where we dropped the explicit dependence on time and grouped all state and input constraints over the control horizon in (2.6d). Once the problem has been solved, the optimal input $\boldsymbol{u}_c^*(t)$ is applied to the system for a duration $\delta_t = 1/f_t$, while waiting for the next feedback measurement to become available for repeating the procedure. It should be noted that the choice of f_t — and of δ_t as a byproduct — is of paramount importance in the design of an MPC controller. Since the time available to carry out the computations is δ_t, it is necessary to compromise between a frequency f_t that is high enough (short δ_t) to capture the evolution of the system's fast dynamics and low enough (long δ_t) so that the computations required to solve the problem are carried out in time.

As the MPC action is computed in discrete time intervals, it can be convenient to formulate the problem directly in discrete-time. Moreover, as will be discussed in Sect. 2.4, the infinite-dimensional continuous-time OCP (2.6) is typically solved

numerically using a finite-dimensional approximation — obtained discretizing over time. To this aim, we divide the control horizon T_c in C sampling intervals of duration δ_t, such that $T_c = C \cdot \delta_t$. Consider then the discretized dynamics obtained from the continuous time model (2.1) by integrating it numerically over $[t_k, t_{k+1})$ with $t_{k+1} = t_k + \delta_t$ and with constant inputs over the interval:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{2.7}$$

Then, the state and input trajectories $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$ over the control horizon $[t_k, t_{k+C}]$ are determined by the sequences $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_C)$ and $\boldsymbol{u} = (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{C-1})$.

The discrete-time OCP solved at each control instant is:

$$\begin{cases} \underset{\boldsymbol{u}}{\text{minimize}} & \displaystyle\sum_{i=0}^{C-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_C) & & \text{(2.8a)} \\ \text{subject to} & \boldsymbol{x}_0 = \hat{\boldsymbol{x}}_k & & \text{(2.8b)} \\ & \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) & \forall i \in \mathbb{I}_0^{C-1} & \text{(2.8c)} \\ & \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq \boldsymbol{0} & \forall i \in \mathbb{I}_0^{C-1} & \text{(2.8d)} \\ & \boldsymbol{g}_f(\boldsymbol{x}_C) \leq \boldsymbol{0} & & \text{(2.8e)} \end{cases}$$

Once the problem has been solved, the first input $\boldsymbol{u}_0^*$ of the optimal sequence is selected and commanded to the system until the next feedback measurement $\hat{\boldsymbol{x}}_{k+1}$ at time $t_{k+1} = t_k + \delta_t$ becomes available for repeating the procedure. In this formulation, we make the assumption that the sampling time δ_t is small enough such that any constraint of type (2.8d) being imposed at the discrete time samples $(t_k, t_{k+1}, \ldots, t_{k+C})$ is also satisfied over the continuous time interval $[t_k, t_{k+C}]$.

Remark 2.2.1. *In the MPC formulation, note how the subscript i refers to the* predicted *time of the trajectory over the control horizon with $t_{k+i} \in [t_k,\ t_{k+C}]$ (for $i = 0, \ldots, C$), that is always reset to zero at each MPC iteration. On the other hand, the subscript k on the time t_k and the current state $\hat{\boldsymbol{x}}_k$ refer to the* actual *time of the closed-loop system evolving with the dynamics* (2.7) *under the action of the MPC controller.*

With the exception of some particular formulations, such as the unconstrained Linear Quadratic Regulator [14], finding an analytical solution to the MPC problems (2.6), (2.8) is virtually impossible. For this reason, it is common to resort to numerical (possibly approximate) methods.

There are three main classes of methods used for the numerical solution of OCP. The first class consists of the Dynamic Programming (DP) and the Hamilton-Jacobi-Bellman (HJB) [15] approaches. Both methods are based on the computation of the level sets of the OCP value function in discrete and continuous time, respectively. These methods suffer from the so called curse of dimensionality as they are practically applicable only to systems with a small number of states.

The second class consists of the *indirect methods*. These methods are based on the application of Pontryagin's minimum principle [16], which provides optimality conditions for the solution of the OCP. The application of such methods usually

consists in finding the (numerical) solution of a multi-point boundary value problem [17]. Although they can lead to highly accurate numerical solutions, indirect methods are not normally applied to on-line implementations, such as MPC [18].

The third class consists of the *direct methods* or *transcription methods*. These methods first transcribe the infinite-dimensional continuous-time OCP to a Non-Linear Program (NLP) of finite dimensions, which is then solved using tailored numerical optimization algorithms [18]. Direct methods are more popular for the on-line solution of an OCP, allowing to utilize high-performance implementations of optimization algorithms designed for broader classes of problems after a suitable transcription of the OCP has been applied. Direct methods are classified by the way in which they transcribe the OCP into an NLP, their differences consisting of the way in which the state and inputs are discretized, as well as the choice of which quantities will be selected as decision variables in the resulting NLP. The three approaches are direct collocation [19], direct single shooting [20] and direct multiple shooting [21].

In general, the NLP resulting from the application of a direct transcription method can be solved, for instance, with Interior Point (IP) or Sequential Quadratic Programming (SQP) algorithms [22], both of which would solve a sequence of appropriately linearized approximations of the problem up to convergence to a local minimizer. In an MPC context however, the time-budget to solve the optimization problem is at most the sampling time δ_t, and reducing the control delay is of paramount importance. For this reason, a number of techniques that are not designed to solve the problem up to convergence, but to solve a sufficiently accurate approximation of it in a reduced amount of time, have been developed [23–26]. In essence, most of these techniques are designed to find a linear approximation of the problem, whose solution can be found in a reasonable and predictable amount of time. In fact, if the prediction model and constraints are linear and the cost function quadratic, the OCP can be transcribed into a Quadratic Program (QP), which can be solved using efficient algorithms even on embedded hardware.

Throughout this thesis, we make use of transcription methods and formulate a suitable approximation of the original problem that can be solved in real-time. In particular, two different methods have been used to transform a the OCP into a QP. At the most basic level, all revolve around the linearization of the problem around a trajectory or setpoint, but they differ in the way the problem is first formulated. In one case, the problem is first linearized, formulated as an OCP and then transcribed into a QP. In the second case, the nonlinear OCP is formulated and then appropriately linearized to yield a QP. It is worth noting that, depending on the way the transcription and linearization are performed, the two approaches can result in the same QP.

2.3 Constrained optimization

Before illustrating the formulation of approximate MPC problems, we review the basics of constrained (nonlinear) optimization. The treatment follows that of [22].

Consider the compact-form NLP:

$$\begin{cases} \underset{\boldsymbol{w}}{\text{minimize}} & \varphi(\boldsymbol{w}) \\ \text{subject to} & \boldsymbol{E}(\boldsymbol{w}) = \boldsymbol{0} \\ & \boldsymbol{G}(\boldsymbol{w}) \leq \boldsymbol{0}, \end{cases} \tag{2.9}$$

where $\boldsymbol{w} \in \mathbb{R}^{n_w}$ are the decision variables — or primal variables — of the problem, $\varphi(\boldsymbol{w})$ is the scalar cost function and $\boldsymbol{E}(\boldsymbol{w}) \in \mathbb{R}^{n_e}$, $\boldsymbol{G}(\boldsymbol{w}) \in \mathbb{R}^{n_g}$ are the vectors stacking equality and inequality constraints, respectively. Functions $\varphi(\cdot)$, $\boldsymbol{E}(\cdot)$ and $\boldsymbol{G}(\cdot)$ are assumed twice continuously differentiable.

Definition 2.3.1 (Feasible set)**.** *The feasible set Ω is the set that contains all points satisfying the constraints:*

$$\Omega = \{\boldsymbol{w} \in \mathbb{R}^{n_w} \,:\, \boldsymbol{E}(\boldsymbol{w}) = \boldsymbol{0},\, \boldsymbol{G}(\boldsymbol{w}) \leq \boldsymbol{0}\}\,.$$

Definition 2.3.2 (Feasible point)**.** *The point $\bar{\boldsymbol{w}} \in \mathbb{R}^{n_w}$ is a feasible point if and only if $\bar{\boldsymbol{w}} \in \Omega$.*

When solving the optimization problem (2.9), we are interested in finding feasible points that locally minimize the cost function $\varphi(\boldsymbol{w})$.

Definition 2.3.3 (Local minimizer)**.** *The point $\boldsymbol{w}^*$ is a local minimizer iff it is a feasible point and there exists a neighborhood $\mathcal{N}$ of $\boldsymbol{w}^*$, such that*

$$\forall \boldsymbol{w} \in \Omega \cap \mathcal{N} \,:\, \varphi(\boldsymbol{w}) \geq \varphi(\boldsymbol{w}^*).$$

It turns out that the set $\mathcal{N}$ needed to verify if a point is a local minimizer or not depends on the inequality constraints which are active locally.

Definition 2.3.4 (Active constraint)**.** *An inequality constraint is termed active at a feasible point $\bar{\boldsymbol{w}} \in \Omega$ iff*

$$G_i(\bar{\boldsymbol{w}}) = 0.$$

Definition 2.3.5. *(Active set) The active set is an index set*

$$\mathcal{A}(\bar{\boldsymbol{w}}) \subset \{1, \ldots, n_g\},$$

where the indices specify the inequality constraints that are active.

These definitions are needed to establish the constraint qualification conditions for which the First-Order Necessary Conditions (FONC) can be stated.

Definition 2.3.6 (LICQ)**.** *The linear independence constraint qualification (LICQ) holds at a feasible point $\bar{\boldsymbol{w}}$ iff all vectors $\nabla E_i(\bar{\boldsymbol{w}})$, for $i = 1, \ldots, n_e$ and $\nabla G_i(\bar{\boldsymbol{w}})$ for $i \in \mathcal{A}(\bar{\boldsymbol{w}})$ are linearly independent.*

This condition can also equivalently stated as the Jacobian of $\tilde{\boldsymbol{G}}(\boldsymbol{w})$ — i.e., the stack of the equality and active inequality constraints — being full row rank.

An essential ingredient for constrained optimization in the so-called Lagrangian function.

Definition 2.3.7. *The Lagrangian function of* (2.9) *is defined as*

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \varphi(\boldsymbol{w}) + \boldsymbol{\lambda}^T \boldsymbol{E}(\boldsymbol{w}) + \boldsymbol{\mu}^T \boldsymbol{G}(\boldsymbol{w}),$$

with $\boldsymbol{\lambda} \in \mathbb{R}^{n_e}$ *and* $\boldsymbol{\mu} \in \mathbb{R}^{n_g}$ *being the Lagrange multipliers — or dual variables.*

The Lagrangian function is used to formulate the famous Karush-Kuhn-Tucker (KKT) conditions, providing first order necessary conditions for the optimality of the solution of problem (2.9).

Theorem 2.3.1 (KKT conditions)**.** *Let* $\boldsymbol{w}^*$ *be a local minimizer of NLP* (2.9) *for which the LICQ holds. Then, there must exist Lagrange multipliers* $\boldsymbol{\lambda}^*$ *and* $\boldsymbol{\mu}^*$ *for which*

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \nabla_{\boldsymbol{w}} \varphi(\boldsymbol{w}^*) + \nabla_{\boldsymbol{w}} \boldsymbol{E}(\boldsymbol{w}^*)\boldsymbol{\lambda}^* + \nabla_{\boldsymbol{w}} \boldsymbol{G}(\boldsymbol{w}^*)\boldsymbol{\mu}^* = \boldsymbol{0} \tag{2.10a}$$

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \boldsymbol{E}(\boldsymbol{w}^*) = \boldsymbol{0} \tag{2.10b}$$

$$\nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \boldsymbol{G}(\boldsymbol{w}^*) \leq \boldsymbol{0} \tag{2.10c}$$

$$\boldsymbol{\mu}^* \geq \boldsymbol{0} \tag{2.10d}$$

$$\mu_i^* G_i(\boldsymbol{w}^*) = 0 \quad i \in \mathbb{I}_1^{n_g}, \tag{2.10e}$$

holds. If some $\boldsymbol{y}^* = (\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *satisfy the KKT conditions, then it is also a solution to the primal and dual problems.*

In constrained optimization, Eq. (2.10a) is denoted as *stationarity condition*, Eq. (2.10b) and (2.10c) as the *primal feasibility*, Eq. (2.10d) as the *dual feasibility* and Eq. (2.10e) as the *complementarity slackness*. These condition define a nonsmooth manifold in which the solution lives. In particular, the non-smoothness is characterized by the activation or not of the inequality constraints at the solution. For this reason, we denote with the term *strictly active* constraints that are active with $\mu_i^* > 0$. Conversely, a constraint which is active with $\mu_i^* = 0$ is called *weakly active*.

Definition 2.3.8 (Strict complementarity)**.** *Let* $\boldsymbol{y}^* = (\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *be a KKT point. Strict complementarity holds at* $\boldsymbol{y}^*$ *iff all active constraints are strictly active.*

When a problem is strictly convex, the KKT conditions are necessary and sufficient for the existence of a unique solution to the problem [27]. In general, it is instead necessary to resort to Second-Order Optimality Conditions, composed of two results: the Second-Order Necessary Conditions (SONC) and the Second-Order Sufficient Conditions (SOSC).

Theorem 2.3.2 (Second-Order Optimality Conditions)**.** *Let* $\boldsymbol{y}^*$ *be a KKT point and assume that LICQ and SC hold. Regard as* $\boldsymbol{Z}$ *the basis matrix of the null space of* $\nabla_{\boldsymbol{w}} \tilde{\boldsymbol{G}}(\boldsymbol{w}^*)^T$*.*

The two following statements hold:

- *(SONC) If* $\boldsymbol{w}^*$ *is a local minimizer, then*

$$\boldsymbol{Z}^T \nabla_{\boldsymbol{w}}^2 \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \boldsymbol{Z} \geq 0.$$

- *(SOSC) If* $\boldsymbol{Z}^T \nabla_{\boldsymbol{w}}^2 \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \boldsymbol{Z} > 0$*, then* $\boldsymbol{w}^*$ *is a local minimizer. This minimizer is unique in its neighborhood, i.e., a strict local minimizer, and stable against small differentiable perturbations of the problem data.*

The matrix $\nabla_{\boldsymbol{w}}^2 \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is referred to as the *Hessian*, while $\boldsymbol{Z}^T \nabla_{\boldsymbol{w}}^2 \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \boldsymbol{Z}$ is usually termed *reduced Hessian*.

2.4 Approximate methods for real-time MPC

In the following, we illustrate the two approaches for formulating approximate MPC problems, providing rather generic formulations which can be used as reference for the following chapters.

2.4.1 Linearized Time Varying MPC

Let us first concentrate on the case of Linearized Time Varying MPC, which is obtained from a nonlinear problem by linearizing the dynamics and constraints over a suitable *auxiliary* trajectory. The intermediate result is an OCP with linear (time-varying) dynamics and constraints, which can then easily be transcribed into a QP.

Consider the prediction model (2.7) and possibly an output function $\boldsymbol{y}(t) = \boldsymbol{h}(\boldsymbol{x}(t))$. Let $(\boldsymbol{x}_{\text{aux}}(t), \boldsymbol{u}_{\text{aux}}(t))$, $t \in [t_k, t_{k+C}]$ be the auxiliary trajectory around which the system is going to be linearized. This trajectory can be, for instance, a reference trajectory for the state (if available), it could be obtained by some online trajectory generation module, or it could be obtained at each iteration of the algorithm by considering the predicted trajectory at the previous MPC iteration, possibly shifted in time.

We now analyze the choice of the cost function, of the prediction model and of the constraints to formulate the final problem, and the different ways of transcribing the OCP into a QP.

Prediction model

Starting from continuous time dynamics (2.1), the model can be linearized with two different approaches, depending if the linearization is performed before or after the discretization. For convenience, we work on the variation with respect to the auxiliary trajectory, however everything can be easily recasted in the original coordinates if deemed more convenient. Define the variations $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{u}$ by applying the change of coordinates $\Delta\boldsymbol{x} = \boldsymbol{x}_c - \boldsymbol{x}_{\text{aux}}$ and $\Delta\boldsymbol{u} = \boldsymbol{u}_c - \boldsymbol{u}_{\text{aux}}$.

First discretize, then linearize Starting from the already discretized prediction model (2.7), the linearized model is simply:

$$\boldsymbol{A}_{k+i} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i}), \boldsymbol{u}_{\text{aux}}(t_{k+i})} \quad \boldsymbol{B}_{k+i} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i}), \boldsymbol{u}_{\text{aux}}(t_{k+i})}. \tag{2.11}$$

First linearize, then discretize Alternatively, one can first linearize the continuous time dynamics (2.1):

$$\boldsymbol{A}_c(t_{k+i}) = \left.\frac{\partial \boldsymbol{f}_c}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i}),\boldsymbol{u}_{\text{aux}}(t_{k+i})} \qquad \boldsymbol{B}_c(t_{k+i}) = \left.\frac{\partial \boldsymbol{f}_c}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i}),\boldsymbol{u}_{\text{aux}}(t_{k+i})}.$$

Then, assuming piecewise constant inputs over the sampling interval, apply for instance the discretization:

$$\begin{aligned} \boldsymbol{A}_{k+i} &= e^{\boldsymbol{A}_c(t_{k+i})\delta_t}, \\ \boldsymbol{B}_{k+i} &= \int_0^{\delta_t} e^{\boldsymbol{A}_c(t_{k+i})\tau} d\tau \boldsymbol{B}_c(t_{k+i}). \end{aligned} \tag{2.12}$$

For the output function $\boldsymbol{y}(t) = \boldsymbol{h}(\boldsymbol{x}(t))$, it is sufficient to apply the chain rule to obtain a linear time-varying approximation:

$$\boldsymbol{y}_i = \underbrace{\left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i})}}_{\boldsymbol{H}_{k+i}} \Delta\boldsymbol{x}_i + \underbrace{\boldsymbol{h}(\boldsymbol{x}_{\text{aux}}(t_{k+i}))}_{\boldsymbol{h}_{k+i}}. \tag{2.13}$$

Finally, applying either method (2.11) or (2.12) and linearizing the output as in (2.13), the linearized prediction model for the state and output over the prediction horizon $[t_k,\, t_{k+C}]$ is:

$$\begin{aligned} \Delta\boldsymbol{x}_{i+1} &= \boldsymbol{A}_{k+i}\Delta\boldsymbol{x}_i + \boldsymbol{B}_{k+i}\Delta\boldsymbol{u}_i \\ \boldsymbol{y}_i &= \boldsymbol{H}_{k+i}\Delta\boldsymbol{x}_i + \boldsymbol{h}_{k+i}, \end{aligned}$$

Constraints

For the generic constraint $\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq \boldsymbol{0}$ and the terminal constraint $\boldsymbol{g}_f(\boldsymbol{x}_C) \leq \boldsymbol{0}$, apply the chain rule to obtain the linear constrains

$$\begin{aligned} \boldsymbol{C}_{k+i}\Delta\boldsymbol{x}_i + \boldsymbol{D}_{k+i}\Delta\boldsymbol{u}_i + \boldsymbol{g}_{k+i} \leq \boldsymbol{0}, \\ \boldsymbol{C}_{k+C}\Delta\boldsymbol{x}_C + \boldsymbol{g}_{k+C} \leq \boldsymbol{0}, \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{g}_{k+i} &= \boldsymbol{g}(\boldsymbol{x}_{\text{aux}}(t_{k+i}), \boldsymbol{u}_{\text{aux}}(t_{k+i})) \\ \boldsymbol{C}_{k+i} &= \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i}),\boldsymbol{u}_{\text{aux}}(t_{k+i})} \qquad \boldsymbol{D}_{k+i} = \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+i}),\boldsymbol{u}_{\text{aux}}(t_{k+i})} \\ \boldsymbol{g}_{k+C} &= \boldsymbol{g}_f(\boldsymbol{x}_{\text{aux}}(t_{k+C})) \\ \boldsymbol{C}_{k+C} &= \left.\frac{\partial \boldsymbol{g}_f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{\text{aux}}(t_{k+C})}. \end{aligned}$$

Concerning the initial state condition $\boldsymbol{x}_0 = \hat{\boldsymbol{x}}_k$, we have

$$\Delta\boldsymbol{x}_0 = \underbrace{\hat{\boldsymbol{x}}_k - \boldsymbol{x}_{\text{aux}}(t_k)}_{\Delta\boldsymbol{x}_k}.$$

Cost function

In order to construct a QP, we limit ourselves to the choice of a quadratic cost function. For example, consider a cost

$$L(\Delta \boldsymbol{u}) = \sum_{i=0}^{C-1} \left(\|\boldsymbol{y}_i - \boldsymbol{y}_d(t_{k+i})\|^2_{\boldsymbol{W}_y} + \|\Delta \boldsymbol{u}_i\|^2_{\boldsymbol{W}_u} \right) + \|\boldsymbol{y}_C - \boldsymbol{y}_d(t_{k+C})\|^2_{\boldsymbol{W}_y} \tag{2.14}$$

in which we minimize the tracking error for the output $\boldsymbol{y}(t)$ and the variation with respect to the auxiliary input $\boldsymbol{u}_{\text{aux}}(t)$ over the control horizon with some positive-definite weight matrices $\boldsymbol{W_y}$, $\boldsymbol{W_u}$, which can be time-dependent or not.

The resulting problem

Finally, we can formulate the resulting OCP, consisting of the cost function, the prediction model for the state and output, and the constraints

$$\begin{cases} \underset{\Delta \boldsymbol{u}(\cdot)}{\text{minimize}} & \frac{1}{2} L(\Delta \boldsymbol{u}) & & \text{(2.15a)} \\ \text{subject to} & \Delta \boldsymbol{x}_0 = \Delta \boldsymbol{x}_k & & \text{(2.15b)} \\ & \Delta \boldsymbol{x}_{i+1} = \boldsymbol{A}_{k+i} \Delta \boldsymbol{x}_i + \boldsymbol{B}_{k+i} \Delta \boldsymbol{u}_i & \forall i \in \mathbb{I}_0^{C-1} & \text{(2.15c)} \\ & \boldsymbol{y}_i = \boldsymbol{H}_{k+i} \Delta \boldsymbol{x}_i + \boldsymbol{h}_{k+i} & \forall i \in \mathbb{I}_0^{C} & \text{(2.15d)} \\ & \boldsymbol{C}_{k+i} \Delta \boldsymbol{x}_i + \boldsymbol{D}_{k+i} \Delta \boldsymbol{u}_i + \boldsymbol{g}_{k+i} \leq \boldsymbol{0} & \forall i \in \mathbb{I}_0^{C-1} & \text{(2.15e)} \\ & \boldsymbol{C}_{k+C} \Delta \boldsymbol{x}_C + \boldsymbol{g}_{k+C} \leq \boldsymbol{0} & & \text{(2.15f)} \end{cases}$$

At each control instant, the current state $\hat{\boldsymbol{x}}_k$ is used to compute the initial condition $\Delta \boldsymbol{x}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}_{\text{aux}}(t_k)$ in (2.15b). Once the problem has been solved, the optimal input trajectory equates to $\boldsymbol{u}^*(t_{k+i}) = \boldsymbol{u}_{\text{aux}}(t_{k+i}) + \Delta \boldsymbol{u}_i^*$, for $i = 0, \dots, C-1$ and the current input $\boldsymbol{u}^*(t_k)$ is commanded to the system.

Transcription of the OCP into a QP

Let us now analyze the different ways how OCP (2.15) can be recasted into a QP. We consider a *sparse* and a *dense* formulation that differ in the choice of decision variables. We first illustrate the sparse formulation, as the dense one can be obtained from the sparse through the elimination of dynamic constraints, an operation referred to as *condensing* [28].

Sparse formulation In the sparse formulation, we transcribe the problem by defining the decision variables vector as the sequences of both states and inputs. Let the decision variables be

$$\boldsymbol{w}_s = (\Delta \boldsymbol{x}, \Delta \boldsymbol{u}) = (\Delta \boldsymbol{x}_0, \dots, \Delta \boldsymbol{x}_C, \Delta \boldsymbol{u}_0, \dots, \Delta \boldsymbol{u}_{C-1}),$$

consisting of the state and the input trajectories. With this choice, only the output has to be expressed as a function of the decision variables and substituted. To this aim, let the output trajectory be

$$\boldsymbol{y} = (\boldsymbol{y}_0, \dots, \boldsymbol{y}_C)$$

and the output drift $\boldsymbol{h} = (\boldsymbol{h}_k, \dots, \boldsymbol{h}_{k+C})$. Then the output trajectory $\boldsymbol{y}$ can be expressed in terms of the decision variables in matrix form as

$$\boldsymbol{y} = \underbrace{\begin{pmatrix} \boldsymbol{H}_k & & \\ & \ddots & \\ & & \boldsymbol{H}_{k+C} \end{pmatrix}}_{\boldsymbol{H}} \Delta\boldsymbol{x} + \boldsymbol{h}$$

The cost function (2.14) can also be rewritten in matrix form. Let

$$\boldsymbol{Q}_y = \begin{pmatrix} \boldsymbol{W}_y & & \\ & \ddots & \\ & & \boldsymbol{W}_y \end{pmatrix}, \quad \boldsymbol{Q}_u = \begin{pmatrix} \boldsymbol{W}_u & & \\ & \ddots & \\ & & \boldsymbol{W}_u \end{pmatrix}.$$

Then

$$L(\boldsymbol{w}_s) = \frac{1}{2}\boldsymbol{w}_s^T \boldsymbol{Q}_s \boldsymbol{w}_s + \boldsymbol{c}_s^T \boldsymbol{w}_s,$$

where

$$\boldsymbol{Q}_s = \begin{pmatrix} \boldsymbol{H}^T \boldsymbol{Q}_y \boldsymbol{H} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_u \end{pmatrix},$$
$$\boldsymbol{c}_s = \begin{pmatrix} \boldsymbol{H}^T (\boldsymbol{h} - \boldsymbol{y}_d) \\ \boldsymbol{0} \end{pmatrix}.$$

The constraints can be easily written in matrix form as well. In fact the stack of equality constraints (2.15b) and (2.15c) is:

$$\underbrace{\begin{pmatrix} \boldsymbol{I}_{n_x} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ -\boldsymbol{A}_k & \boldsymbol{I}_{n_x} & \ddots & \vdots & -\boldsymbol{B}_k & \ddots & \vdots \\ & \ddots & \ddots & \boldsymbol{0} & & \ddots & \boldsymbol{0} \\ & & -\boldsymbol{A}_{k+C-1} & \boldsymbol{I}_{n_x} & & & -\boldsymbol{B}_{k+C-1} \end{pmatrix}}_{\boldsymbol{E}} \boldsymbol{w}_s = \underbrace{\begin{pmatrix} \boldsymbol{I}_{n_x} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{pmatrix} \Delta\boldsymbol{x}_k}_{\boldsymbol{\eta}}. \quad (2.16)$$

Similarly, for the inequality constraints (2.15e) and (2.15f):

$$\underbrace{\begin{pmatrix} \boldsymbol{C}_k & & & \boldsymbol{D}_k & & \\ & \ddots & & & \ddots & \\ & & \boldsymbol{C}_{k+C-1} & & & \boldsymbol{D}_{k+C-1} \\ & & & \boldsymbol{C}_{k+C} & & \boldsymbol{0}_{n_x \times n_u} \end{pmatrix}}_{\boldsymbol{G}_s} \boldsymbol{w}_s + \underbrace{\begin{pmatrix} \boldsymbol{g}_k \\ \vdots \\ \boldsymbol{g}_{k+C-1} \\ \boldsymbol{g}_{k+C} \end{pmatrix}}_{\boldsymbol{g}_s} \leq \boldsymbol{0}. \quad (2.17)$$

The QP problem can then be written in compact form

$$\begin{cases} \underset{\boldsymbol{w}_s}{\text{minimize}} & \frac{1}{2}\boldsymbol{w}_s^T \boldsymbol{Q}_s \boldsymbol{w}_s + \boldsymbol{c}_s^T \boldsymbol{w}_s \\ \text{subject to} & \boldsymbol{E}\boldsymbol{w}_s + \boldsymbol{\eta} = \boldsymbol{0} \\ & \boldsymbol{G}_s \boldsymbol{w}_s + \boldsymbol{g}_s \leq \boldsymbol{0}. \end{cases} \quad (2.18)$$

This Quadratic Program is said to be *sparse* since the Hessian $\boldsymbol{Q}_s$ and the constraint matrices $\boldsymbol{E}$, $\boldsymbol{G}_s$ are by construction sparse. In particular, it can be noted how they present block diagonal sparsity patterns due to the link between successive states though the dynamics. This is a characteristic of OCPs transcribed using multiple shooting, which can be exploited to solve the problem by applying a backward Riccati recursion [29].

Clearly, the use of $n_x \cdot (C+1) + n_u \cdot C$ decision variables $\boldsymbol{w}_s$ in this formulation is non minimal, as the sequence of inputs $\boldsymbol{u}$ uniquely determines the state evolution $\boldsymbol{x}$ from the initial condition $\hat{\boldsymbol{x}}_k$. While this can in principle increase the cost of solving the problem, sparse QP solvers like OSQP [30] or ProxQP [31] can exploit the sparsity to reduce the computational cost to a level comparable of that of solving a reduced problem with the dense formulation, to be illustrated next.

Dense formulation From the sparse formulation, it is possible to eliminate the dynamics constraint (2.16) and explicitly express the dependence of the state trajectory $\Delta\boldsymbol{x}$ on the input trajectory $\Delta\boldsymbol{u}$ and the initial value $\Delta\boldsymbol{x}_k$. Let

$$\boldsymbol{\Phi}_{m,n} = \boldsymbol{A}_{k+n-1} \cdots \boldsymbol{A}_{k+m},$$

with $\boldsymbol{\Phi}_{m,m} = \boldsymbol{I}_{n_x}$ and

$$\boldsymbol{\Psi}_n = \sum_{j=0}^{n-1} \boldsymbol{\Phi}_{j+1,n} \boldsymbol{B}_{k+j} \Delta\boldsymbol{u}_j.$$

From linear system theory, it is well known that the sequence of states from the initial condition $\Delta\boldsymbol{x}_0 = \Delta\boldsymbol{x}_k$ satisfies

$$\Delta\boldsymbol{x}_i = \boldsymbol{\Phi}_{0,i}\Delta\boldsymbol{x}_k + \boldsymbol{\Psi}_i, \quad \text{for } i = 0, \ldots, C.$$

This can also be rewritten in matrix form as

$$\Delta\boldsymbol{x} = \underbrace{\begin{pmatrix} \boldsymbol{0}_{n_x\times n_u} & \boldsymbol{0}_{n_x\times n_u} & \cdots & \boldsymbol{0}_{n_x\times n_u} \\ \boldsymbol{\Phi}_{1,1}\boldsymbol{B}_k & \boldsymbol{0}_{n_x\times n_u} & \cdots & \boldsymbol{0}_{n_x\times n_u} \\ \boldsymbol{\Phi}_{1,2}\boldsymbol{B}_k & \boldsymbol{\Phi}_{2,2}\boldsymbol{B}_{k+1} & \cdots & \boldsymbol{0}_{n_x\times n_u} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Phi}_{1,C}\boldsymbol{B}_k & \boldsymbol{\Phi}_{2,C}\boldsymbol{B}_{k+1} & \cdots & \boldsymbol{\Phi}_{C,C}\boldsymbol{B}_{k+C-1} \end{pmatrix}}_{\boldsymbol{\Psi}} \Delta\boldsymbol{u} + \underbrace{\begin{pmatrix} \boldsymbol{I}_{n_x} \\ \boldsymbol{\Phi}_{0,1} \\ \boldsymbol{\Phi}_{0,2} \\ \vdots \\ \boldsymbol{\Phi}_{0,C} \end{pmatrix}}_{\boldsymbol{\Phi}} \Delta\boldsymbol{x}_k. \tag{2.19}$$

This expression can be used to eliminate the dynamics constraint from the QP problem (2.18) and rewrite the cost function and remaining constraints as a function of the inputs only. In fact, let $\boldsymbol{w}_d = \boldsymbol{u}$ and substitute

$$\boldsymbol{w}_s = \begin{pmatrix} \boldsymbol{\Psi}\boldsymbol{w}_d + \boldsymbol{\Phi}\Delta\boldsymbol{x}_k \\ \boldsymbol{w}_d \end{pmatrix}.$$

Then the cost function becomes

$$L(\boldsymbol{w}_d) = \frac{1}{2}\boldsymbol{w}_d^T \boldsymbol{Q}_d \boldsymbol{w}_d + \boldsymbol{c}_d^T \boldsymbol{w}_d + \boldsymbol{d},$$

where

$$\begin{aligned}
\boldsymbol{Q}_d &= \boldsymbol{\Psi}^T \boldsymbol{H}^T \boldsymbol{Q}_y \boldsymbol{H} \boldsymbol{\Psi} + \boldsymbol{Q}_u \\
\boldsymbol{c}_d &= \boldsymbol{\Psi}^T \boldsymbol{H}^T \boldsymbol{Q}_y \left(\boldsymbol{h} - \boldsymbol{y}_d + \boldsymbol{H} \boldsymbol{\Phi} \Delta \boldsymbol{x}_k \right) \\
\boldsymbol{d} &= \Delta \boldsymbol{x}_k^T \boldsymbol{\Phi}^T \left(\boldsymbol{H}^T \boldsymbol{Q}_y \boldsymbol{H} \boldsymbol{\Phi} \Delta \boldsymbol{x}_k + \boldsymbol{H}^T \boldsymbol{Q}_y \left(\boldsymbol{h} - \boldsymbol{y}_d \right) \right) + \frac{1}{2} \left(\boldsymbol{h} - \boldsymbol{y}_d \right)^T \boldsymbol{Q}_y \left(\boldsymbol{h} - \boldsymbol{y}_d \right).
\end{aligned}$$

Again, the same can be done for the constraints:

$$\boldsymbol{G}_s \begin{pmatrix} \boldsymbol{\Psi} \boldsymbol{w}_d + \boldsymbol{\Phi} \Delta \boldsymbol{x}_k \\ \boldsymbol{w}_d \end{pmatrix} + \boldsymbol{g}_s = \underbrace{\boldsymbol{G}_s \begin{pmatrix} \boldsymbol{\Psi} \\ \boldsymbol{I} \end{pmatrix}}_{\boldsymbol{G}_d} \boldsymbol{w}_d + \underbrace{\boldsymbol{G}_s \begin{pmatrix} \boldsymbol{\Phi} \Delta \boldsymbol{x}_k \\ \boldsymbol{0} \end{pmatrix} + \boldsymbol{g}_s}_{\boldsymbol{g}_d}.$$

Finally, the dense QP problem becomes

$$\begin{cases} \underset{\boldsymbol{w}_d}{\text{minimize}} & \frac{1}{2} \boldsymbol{w}_d^T \boldsymbol{Q}_d \boldsymbol{w}_d + \boldsymbol{c}_d^T \boldsymbol{w}_d \\ \text{subject to} & \boldsymbol{G}_d \boldsymbol{w}_d + \boldsymbol{g}_d \leq \boldsymbol{0}. \end{cases} \tag{2.20}$$

In this case, matrices $\boldsymbol{Q}_d$ and $\boldsymbol{G}_d$ are dense as they are obtained from the product of sparse and dense matrices. Compared to the sparse formulation, the number of decision variables is reduced to $n_u \cdot C$ as only the input sequence is free to be optimized, and the $n_x \cdot (C+1)$ equality constraints of the dynamics and initial condition have been eliminated.

Again, it is worth pointing out that the two formulations provide the same result[1] and that the computational cost is comparable if the sparsity of the first formulation is exploited by, e.g., the use of sparse QP solvers. Ultimately, the choice between one of the two boils down to the availability of a specific solver or to specific hardware requirements, e.g., to limit memory usage on embedded platforms.

2.4.2 The Real-Time Iteration scheme

The second approximate method for MPC we describe is the Real-Time Iteration scheme (RTI), originally introduced by Diehl [32,33]. A tutorial on the method and its similarities with the linearized MPC approach can be found in [26].

The basis of RTI is the application of SQP to the NLP resulting from a multiple shooting transcription of the OCP. RTI then performs *one iteration* of SQP per sampling instant, solving the QP subproblem obtained from the linearization around the current solution guess, possibly using a Hessian approximation. In this way, the solution time is drastically reduced with respect to performing SQP iterations up to convergence and the method becomes applicable to real-time control.

First, we perform a multiple shooting transcription of the OCP into a NLP. If the OCP has already been formulated in discrete time, this procedure is straightforward; if the problem is instead formulated in continuous time, the first step is to discretize the dynamics to obtain (2.7).

[1] The state trajectory $\boldsymbol{x}^*$ can be obtained in the dense formulation by applying the optimal input $\boldsymbol{u}^*$ to equation (2.19) or simply by integrating the dynamics recursively.

Then, let

$$\boldsymbol{w} = (\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x}_0, \dots, \boldsymbol{x}_C, \boldsymbol{u}_0, \dots, \boldsymbol{u}_{C-1})$$

be the decision variables of the NLP, consisting of the state and the input trajectories over the time horizon. Then, the transcription of OCP (2.8) into an NLP takes the form:

$$\begin{cases} \underset{\boldsymbol{w}}{\text{minimize}} & \varphi(\boldsymbol{w}) \\ \text{subject to} & \boldsymbol{E}(\boldsymbol{w}, \hat{\boldsymbol{x}}_k) = \boldsymbol{0} \\ & \boldsymbol{G}(\boldsymbol{w}) \leq \boldsymbol{0}, \end{cases} \tag{2.21}$$

where

$$\varphi(\boldsymbol{w}) = \sum_{i=0}^{C-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_C),$$

$$\boldsymbol{E}(\boldsymbol{w}, \hat{\boldsymbol{x}}_k) = \begin{pmatrix} \boldsymbol{x}_0 - \hat{\boldsymbol{x}}_k \\ \boldsymbol{x}_1 - \boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0) \\ \vdots \\ \boldsymbol{x}_C - \boldsymbol{f}(\boldsymbol{x}_{C-1}, \boldsymbol{u}_{C-1}) \end{pmatrix},$$

and

$$\boldsymbol{G}(\boldsymbol{w}) = \begin{pmatrix} \boldsymbol{g}(\boldsymbol{x}_0, \boldsymbol{u}_0) \\ \vdots \\ \boldsymbol{g}(\boldsymbol{x}_{C-1}, \boldsymbol{u}_{C-1}) \\ \boldsymbol{g}_f(\boldsymbol{x}_C) \end{pmatrix}.$$

Note the dependence of the equality constraints vector $\boldsymbol{E}(\boldsymbol{w}, \hat{\boldsymbol{x}}_k)$ on the initial condition $\hat{\boldsymbol{x}}_k$. In fact, the NLP can be interpreted as a *parametric* optimization problem with respect to the current state [28].

The SQP method solves the general NLP (2.21) by iterating over the solution of a QP obtained linearizing functions $\varphi(\boldsymbol{w})$, $\boldsymbol{E}(\boldsymbol{w}, \hat{\boldsymbol{x}}_k)$ and $\boldsymbol{G}(\boldsymbol{w})$ around a solution guess. Starting with an initial guess $(\boldsymbol{w}_0, \boldsymbol{\lambda}_0, \boldsymbol{\mu}_0)$, at each step of the algorithm a new candidate solution is computed through the update rule:

$$\boldsymbol{w}_{\kappa+1} = \boldsymbol{w}_\kappa + \alpha\, \Delta\boldsymbol{w}_\kappa, \quad \boldsymbol{\lambda}_{\kappa+1} = \boldsymbol{\lambda}_\kappa + \alpha \left(\boldsymbol{\lambda}_\kappa^{\text{QP}} - \boldsymbol{\lambda}_\kappa\right), \quad \boldsymbol{\mu}_{\kappa+1} = \boldsymbol{\mu}_\kappa + \alpha \left(\boldsymbol{\mu}_\kappa^{\text{QP}} - \boldsymbol{\mu}_\kappa\right),$$

where $\alpha \in (0,\, 1]$ is the step length, determined using a globalization strategy such as trust region [34], line search [35], or filter methods [36], and $\left(\Delta\boldsymbol{w}_\kappa, \boldsymbol{\lambda}_\kappa^{\text{QP}}, \boldsymbol{\mu}_\kappa^{\text{QP}}\right)$ are the primal-dual solution of the following QP:

$$\begin{cases} \underset{\Delta\boldsymbol{w}}{\text{minimize}} & \frac{1}{2}\Delta\boldsymbol{w}^T \boldsymbol{B}_\kappa \Delta\boldsymbol{w} + \nabla_{\boldsymbol{w}}\varphi(\boldsymbol{w}_\kappa)^T \Delta\boldsymbol{w} \\ \text{subject to} & \boldsymbol{E}(\boldsymbol{w}_\kappa, \hat{\boldsymbol{x}}_k) + \nabla_{\boldsymbol{w}}\boldsymbol{E}(\boldsymbol{w}_\kappa, \hat{\boldsymbol{x}}_k)\Delta\boldsymbol{w} = \boldsymbol{0} \\ & \boldsymbol{G}(\boldsymbol{w}_\kappa) + \nabla_{\boldsymbol{w}}\boldsymbol{G}(\boldsymbol{w}_\kappa)\Delta\boldsymbol{w} \leq \boldsymbol{0}. \end{cases} \tag{2.22}$$

Here, $\nabla_{\boldsymbol{w}}\boldsymbol{E}(\cdot)$ and $\nabla_{\boldsymbol{w}}\boldsymbol{G}(\cdot)$ denote the constraint Jacobians, being evaluated at $\boldsymbol{w}_\kappa$. The matrix $\boldsymbol{B}_\kappa$ denotes the exact Hessian of the Lagrangian, that is $\nabla^2_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_\kappa, \boldsymbol{\lambda}_\kappa, \boldsymbol{\mu}_\kappa)$, or an Hessian approximation, and $\nabla_{\boldsymbol{w}}\varphi(\cdot)$ is the gradient of the cost function. The procedure is in general repeated up to convergence with an adaptive step length.

It is interesting to note how one SQP iteration essentially consists in solving a linearized version of the original problem, with the constraints being satisfied

to their first-order after one iteration. This is a particularly enticing proposition in the context of real-time control, where often a guess based on the solution to the problem at the previous control instant is close to the optimal solution, and computing such an approximate solution provides a sufficiently accurate policy for feedback control, considering that the solution is going to be refined after a short amount of time.

Concerning the Hessian matrix of the QP subproblem (2.22), while using the exact Hessian $\nabla^2_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_\kappa, \boldsymbol{\lambda}_\kappa, \boldsymbol{\mu}_\kappa)$ is possible, it is often avoided due to the need for computing expensive second-order derivatives of the Lagrangian and for regularization strategies to ensure that the resulting QP is convex [22]. On the other hand, using the exact Hessian has the advantage of providing quadratic convergence in the vicinity of the optimal solution, and has been successfully used in the context of robot motion generation, for instance in [37, 38]. Another popular choice is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update, which consists in iteratively approximating the exact Hessian using only first-order derivatives of the Lagrangian, thus reducing the computational cost. While this technique only provides a superlinear convergence rate, it has been successfully employed in a variety of works [39–41].

Another common Hessian approximation, that will be used in the following, is based on the so-called Gauss-Newton method. This techniques provides a multiplier-free approximation, thus not being affected by their initialization. The method applies to cost functions expressed in a nonlinear least squares form:

$$\varphi(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{R}(\boldsymbol{w})\|^2,$$

with $\boldsymbol{R}(\boldsymbol{w})$ being called the *residual* vector. Then, the Gauss-Newton Hessian approximation is $\boldsymbol{B}_\kappa = \nabla_{\boldsymbol{w}}\boldsymbol{R}(\boldsymbol{w}_\kappa)\nabla_{\boldsymbol{w}}\boldsymbol{R}(\boldsymbol{w}_\kappa)^T$, being obtained using only first-order derivatives. This approximation provides linear convergence in a neighborhood of the solution, and has proven to be a computationally efficient and valid choice in a range of robotic applications [42, 43].

The RTI method consists in performing one iteration of SQP with full step length ($\alpha = 1$) and a Gauss-Newton Hessian approximation. The current solution guess around which the problem is linearized, that we denote with $\bar{\boldsymbol{w}} = (\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$, is obtained from the solution of the MPC at the previous control cycle, possibly shifting the trajectory of one step in time [29].

Define the decision variables vector for the QP as $\boldsymbol{\Delta w} = \boldsymbol{w} - \bar{\boldsymbol{w}}$. The MPC controller then solves the problem:

$$\begin{cases} \underset{\boldsymbol{\Delta w}}{\text{minimize}} & \frac{1}{2}\boldsymbol{\Delta w}^T\boldsymbol{H}\boldsymbol{\Delta w} + \boldsymbol{c}^T\boldsymbol{\Delta w} \\ \text{subject to} & \boldsymbol{\Delta x}_0 + \bar{\boldsymbol{x}}_0 - \hat{\boldsymbol{x}}_k = \boldsymbol{0} \\ & \boldsymbol{\Delta x}_{i+1} - \boldsymbol{A}_i\boldsymbol{\Delta x}_i - \boldsymbol{B}_i\boldsymbol{\Delta u}_i + \boldsymbol{f}_i = \boldsymbol{0} \quad \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{C}_i\boldsymbol{\Delta x}_i + \boldsymbol{D}_i\boldsymbol{\Delta u}_i + \boldsymbol{g}_i \leq \boldsymbol{0} \quad \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{C}_C\boldsymbol{\Delta x}_C + \boldsymbol{g}_C \leq \boldsymbol{0} \end{cases}$$

where

$$\begin{aligned}
\boldsymbol{H} &= \left.\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\right|_{\bar{\boldsymbol{w}}}^{T} \left.\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\right|_{\bar{\boldsymbol{w}}} & \boldsymbol{c} &= \left.\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\right|_{\bar{\boldsymbol{w}}}^{T} \boldsymbol{R}(\bar{\boldsymbol{w}}) \\
\boldsymbol{A}_i &= \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}_i,\bar{\boldsymbol{u}}_i} \boldsymbol{B}_i = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\bar{\boldsymbol{x}}_i,\bar{\boldsymbol{u}}_i} & \boldsymbol{f}_i &= \bar{\boldsymbol{x}}_{i+1} - \boldsymbol{f}(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{u}}_i) \\
\boldsymbol{C}_i &= \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}_i,\bar{\boldsymbol{u}}_i} \boldsymbol{D}_i = \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right|_{\bar{\boldsymbol{x}}_i,\bar{\boldsymbol{u}}_i} & \boldsymbol{g}_i &= \boldsymbol{g}(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{u}}_i) \\
\boldsymbol{C}_C &= \left.\frac{\partial \boldsymbol{g}_f}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}_C} & \boldsymbol{g}_C &= \boldsymbol{g}_f(\bar{\boldsymbol{x}}_C).
\end{aligned} \tag{2.23}$$

Finally, let us rewrite the QP problem in compact form by stacking and grouping the equality and inequality constraints into matrices $\boldsymbol{E}$ and $\boldsymbol{G}$, respectively, by letting $\boldsymbol{\eta}(\hat{\boldsymbol{x}}_k) = (\bar{\boldsymbol{x}}_0 - \hat{\boldsymbol{x}}_k, \boldsymbol{f}_0, \dots, \boldsymbol{f}_{C-1})$ and $\boldsymbol{g} = (\boldsymbol{g}_0, \dots, \boldsymbol{g}_C)$, in order to express everything with respect to the decision variables vector $\boldsymbol{\Delta w}$:

$$\begin{cases}
\underset{\boldsymbol{\Delta w}}{\text{minimize}} & \frac{1}{2}\boldsymbol{\Delta w}^T \boldsymbol{H} \boldsymbol{\Delta w} + \boldsymbol{c}^T \boldsymbol{\Delta w} \\
\text{subject to} & \boldsymbol{E}\boldsymbol{\Delta w} + \boldsymbol{\eta}(\hat{\boldsymbol{x}}_k) = \boldsymbol{0} \\
& \boldsymbol{G}\boldsymbol{\Delta w} + \boldsymbol{g} \leq \boldsymbol{0}.
\end{cases} \tag{2.24}$$

Here, matrices $\boldsymbol{E}$ and $\boldsymbol{G}$ are sparse [28] with block diagonal non-zero entries, having the same structure as matrices $\boldsymbol{E}$ and $\boldsymbol{G}_s$ in (2.16) and (2.17), which can be exploited by QP solvers to efficiently compute the solution to the problem.

By solving this problem at each control instant, the MPC algorithm computes an optimal state and input trajectory $\boldsymbol{w}^* = \bar{\boldsymbol{w}} + \boldsymbol{\Delta w}^*$ from the current state at time t_k to the end state at time t_{k+C}.

Since the optimization problem (2.21) depends linearly on the current state $\hat{\boldsymbol{x}}_k$, it is possible to construct the RTI approximation (2.24) of the problem, except for the vector $\boldsymbol{\eta}(\hat{\boldsymbol{x}}_k)$ without the need for knowing the initial state. This allows the split in the so-called *preparation* phase and *feedback* phase aimed at reducing the control delay introduced by the algorithm [26]. In fact, in the preparation phase, all the derivatives can be evaluated before the state $\hat{\boldsymbol{x}}_k$ is made available. Then, when the state is measured, the feedback phase computes the solution of the already constructed problem. This can provide a substantial reduction in the control delay since the preparation phase is reported to be up to an order of magnitude more computationally expensive than the feedback phase.

Part I

Motion Generation Using Intrinsically Stable MPC

Chapter 3

Intrinsically Stable MPC

In many applications, the main objective of the control system is to generate commands to realize a desired motion. Very often, this desired motion is expressed by some low-dimensional function (or subset) of the states. This is the case, for instance, of redundant articulated robots, e.g., a 7 degrees of freedom robot manipulator controlling the position of its end-effector [44], or a legged robot whose feet have to follow the desired footsteps to locomote [45], where by definition the output function has a dimension lower that the degrees of freedom of the system. This problem, denoted as *output tracking*, is typically intended *asymptotically*; that is, if due to the initial conditions or to disturbances the system is not able to instantaneously perform exact tracking of the desired motion, this is at least achieved after a sufficiently long time.

In the control literature, the notion of zero dynamics, akin to that of zeros of a linear system, serves the purpose of describing the unobservable internal subsystem that could arise when tracking a desired output [46]. This notion, that has been introduced in the context of input-output linearization of nonlinear systems, is not only relevant in the case of inversion using feedback linearization, but is actually useful in describing any system that is forced to track a desired output. For this reason, the question of controlling the behavior of the zero dynamics of the system becomes relevant also in the case of optimization-based motion generation.

Systems whose zero dynamics is unstable are called non-minimum phase. Controlling non-minimum phase nonlinear systems poses additional challenges over minimum phase systems [47, 48], as the evolution of the internal dynamics can be unbounded [49], possibly leading to control saturation or to the complete failure of the control system. In this thesis, we explore the use of methods where a stability condition based on the theory of linear Stable Inversion is imposed on the MPC optimization problem after having appropriately linearized the nonlinear dynamics. While providing only a local approximation, this proved to be an efficient means of providing stability while controlling non-minimum phase systems using MPC.

The main methodology, i.e., the so-called Intrinsically Stable Model Predictive Control (IS-MPC), has been introduced in the context of gait generation of Humanoid Robots (we refer to [50] for an extensive presentation of this application), where the Linear Inverted Pendulum (LIP) model is used to describe the evolution of the Center of Mass of a humanoid with respect to the Zero Moment Point (ZMP)

— or alternatively the Center of Pressure — at the foot. In the LIP model, which can be seen as the inverse of the Cart-Table model [51], the ZMP acts as the forcing input, whose trajectory is typically determined from the position of the footsteps. The CoM trajectory is however unstable, with the LIP model having an unstable eigenvalue. IS-MPC was then introduced to generate gaits (ZMP trajectories and footstep positions) in which the CoM trajectory remains bounded with respect to the ZMP trajectory, allowing the humanoid robot to walk without falling.

The chapter is organized as follows. Sect. 3.1 provides a short overview of the theory of nonlinear systems, input-output feedback linearization, and the zero dynamics. In Sect. 3.2 we discuss the origin of the stability condition at the center of the proposed method. Then, Sect. 3.3 contains a concise and general overview of the IS-MPC method, which is finally showcased through the application to a Wheeled Inverted Pendulum robot in Sect. 3.3.1.

3.1 Preliminaries

In this section, we provide the necessary preliminaries on the input-output linearization of nonlinear systems and the zero dynamics arising when tracking a desired output. The treatment follows that of [52, Chapter 4] and is proposed in the case of single input single output (SISO) systems for brevity and ease of exposition. The discussion and results however can be easily replicated for the multiple input multiple output (MIMO) case, for which we refer the reader to [52, Chapter 5].

Consider the nonlinear input-affine SISO system

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})u \\ y &= h(\boldsymbol{x}), \end{aligned} \tag{3.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $u, y \in \mathbb{R}$.

Definition 3.1.1 (Relative Degree)**.** *System* (3.1) *is said to have relative degree r at $\boldsymbol{x}'$ if*

1. $L_{\boldsymbol{g}} L_{\boldsymbol{f}}^k h(\boldsymbol{x}) = 0$ *for all $\boldsymbol{x}$ in a neighborhood of $\boldsymbol{x}'$ and all $k < r-1$,*
2. $L_{\boldsymbol{g}} L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{x}') \neq 0$,

where $L_{\boldsymbol{a}} b(\boldsymbol{x})$ denotes the Lie derivative of function $b(\boldsymbol{x})$ along the vector field $\boldsymbol{a}(\boldsymbol{x})$, that is, $L_{\boldsymbol{a}} b(\boldsymbol{x}) = \frac{\partial b}{\partial x}\boldsymbol{a}$.

The notion of relative degree plays a pivotal role in defining a change of coordinates that transforms the system in the so-called Byrnes-Isidori *normal form*. This particular output-focused normal form will highlight the input-output structure of the system, allowing to apply straightforward control laws to control the output evolution and eventually characterizing the behavior of the (possible) internal unobservable dynamics.

Proposition 3.1.1. *Suppose the system has relative degree $r \leq n$ at $\boldsymbol{x}'$. Set*

$$\begin{array}{rcl} \phi_1(\boldsymbol{x}) & = & h(\boldsymbol{x}) \\ \phi_2(\boldsymbol{x}) & = & L_{\boldsymbol{f}} h(\boldsymbol{x}) \\ & \vdots & \\ \phi_r(\boldsymbol{x}) & = & L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{x}). \end{array} \tag{3.2}$$

The row vectors

$$d\phi_1(\boldsymbol{x}'), d\phi_2(\boldsymbol{x}'), \ldots, d\phi_r(\boldsymbol{x}')$$

are linearly independent. If r is strictly less than n, it is always possible to find $n - r$ more functions $\phi_{r+1}(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})$ such that the mapping

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \begin{pmatrix} \phi_1(\boldsymbol{x}) \\ \vdots \\ \phi_n(\boldsymbol{x}) \end{pmatrix} \tag{3.3}$$

has a Jacobian which is non-singular at $\boldsymbol{x}'$ and therefore qualifies as a local coordinate transformation in a neighborhood of $\boldsymbol{x}'$. Moreover, it is always possible to choose $\phi_{r+1}(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})$ in such a way that

$$L_{\boldsymbol{g}} \phi_i(\boldsymbol{x}) = 0 \quad \textit{for all } r+1 \leq i \leq n \textit{ and all } \boldsymbol{x} \textit{ around } \boldsymbol{x}'.$$

Proof. The proposition is an adaptation of Lemma 4.1.1 and Proposition 4.1.3 of [52], the proof of which can be found therein. ∎

We now apply the change of coordinates defined by mapping (3.3) to system (3.1). Let the new coordinates be $\boldsymbol{z} = (z_1, \ldots, z_n)$ with $z_i = \phi_i(\boldsymbol{x})$ for $1 \leq i \leq n$. Then, for the first r coordinates, recalling (3.2) and Definition 3.1.1 we obtain

$$\begin{array}{rcl} \dfrac{dz_1}{dt} = \dfrac{\partial \phi_1}{\partial \boldsymbol{x}} \dfrac{d\boldsymbol{x}}{dt} = \dfrac{\partial h}{\partial \boldsymbol{x}} \dfrac{d\boldsymbol{x}}{dt} = L_{\boldsymbol{f}} h(\boldsymbol{x}) = \phi_2(\boldsymbol{x}) = z_2 \\ \vdots \\ \dfrac{dz_{r-1}}{dt} = \dfrac{\partial \phi_{r-1}}{\partial \boldsymbol{x}} \dfrac{d\boldsymbol{x}}{dt} = \dfrac{\partial L_{\boldsymbol{f}}^{r-2} h}{\partial \boldsymbol{x}} \dfrac{d\boldsymbol{x}}{dt} = L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{x}) = \phi_r(\boldsymbol{x}) = z_r \\ \dfrac{dz_r}{dt} = \dfrac{\partial \phi_r}{\partial \boldsymbol{x}} \dfrac{d\boldsymbol{x}}{dt} = \dfrac{\partial L_{\boldsymbol{f}}^{r-1} h}{\partial \boldsymbol{x}} \dfrac{d\boldsymbol{x}}{dt} = L_{\boldsymbol{f}}^{r} h(\boldsymbol{x}) + L_{\boldsymbol{g}} L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{x}) u. \end{array} \tag{3.4}$$

For the last equation, it is still necessary to apply the inverse coordinate transformation to express it as a function of $\boldsymbol{z}$, i.e., $\boldsymbol{x} = \boldsymbol{\Phi}^{-1}(\boldsymbol{z})$. Then, by defining

$$\begin{aligned} a(\boldsymbol{z}) &= L_{\boldsymbol{g}} L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{\Phi}^{-1}(\boldsymbol{z})) \\ b(\boldsymbol{z}) &= L_{\boldsymbol{f}}^{r} h(\Phi^{-1}(\boldsymbol{z})) \end{aligned}$$

the equation becomes

$$\frac{dz_r}{dt} = b(\boldsymbol{z}) + a(\boldsymbol{z}) u.$$

Concerning the last $n - r$ equations, no special structure arises in general. In fact, for $r + 1 \leq i \leq n$:

$$\frac{dz_i}{dt} = \frac{\partial \phi_i}{\partial \boldsymbol{x}} \left(\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x}) u \right) = L_{\boldsymbol{f}} \phi_i(\boldsymbol{x}) + L_{\boldsymbol{g}} \phi_i(\boldsymbol{x}) u.$$

Again, we must express the right-hand side of the equation as a function of z. To this aim, for $r+1 \leq i \leq n$ define

$$\begin{aligned} p_i(\boldsymbol{z}) &= L_{\boldsymbol{g}}\phi_i(\boldsymbol{\Phi}^{-1}(\boldsymbol{z})) \\ q_i(\boldsymbol{z}) &= L_{\boldsymbol{f}}\phi_i(\boldsymbol{\Phi}^{-1}(\boldsymbol{z})) \end{aligned}$$

such that

$$\frac{dz_i}{dt} = q_i(\boldsymbol{z}) + p_i(\boldsymbol{z})u. \tag{3.5}$$

Finally, the full state-space representation obtained by combining (3.4) and (3.5), that we denote as *normal form* is:

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= z_3 \\ &\vdots \\ \dot{z}_{r-1} &= z_r \\ \dot{z}_r &= b(\boldsymbol{z}) + a(\boldsymbol{z})u \\ \dot{z}_{r+1} &= q_{r+1}(\boldsymbol{z}) + p_{r+1}(\boldsymbol{z})u \\ &\vdots \\ \dot{z}_n &= q_n(\boldsymbol{z}) + p_n(\boldsymbol{z})u, \end{aligned} \tag{3.6}$$

with output $y = z_1$. As stated in Proposition 3.1.1, it is always possible to find a change of coordinates such that $L_{\boldsymbol{g}}\phi_i(\boldsymbol{x}) = 0$ for $r+1 \leq i \leq n$, which would result in the last $n-r$ equations of the dynamics not depending explicitly on the input[1]. However, since constructing such functions involves solving a system of $n-r$ partial differential equations, it is often not easy to find a solution which satisfied this additional requirement, which is often neglected.

3.1.1 Linearization via feedback

By analyzing the normal form (3.6), it is readily apparent how in these new coordinates the input-output relationship can be made linear by means of (static) state feedback. This procedure, denoted with *Input-Output Feedback Linearization*, allows to impose any arbitrary r-order dynamics to the output, posing the basis to solving the output regulation problem.

Consider the state feedback law

$$u = \frac{1}{a(\boldsymbol{z})}\left(-b(\boldsymbol{z}) + v\right), \tag{3.7}$$

where v is an external reference input. Note that by construction $a(\boldsymbol{z}) \neq 0$ in a neighborhood of $\boldsymbol{x}'$, so the control law is always (locally) well-defined if the system

[1] They would still be, in general, coupled with the rest of the dynamics.

has relative degree r. Applying this control law to system (3.6), one obtains

$$\begin{aligned}
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\vdots \\
\dot{z}_{r-1} &= z_r \\
\dot{z}_r &= v \\
\dot{z}_{r+1} &= q_{r+1}(\boldsymbol{z}) + \frac{p_{r+1}(\boldsymbol{z})}{a(\boldsymbol{z})}\left(-b(\boldsymbol{z}) + v\right) \\
&\vdots \\
\dot{z}_n &= q_n(\boldsymbol{z}) + \frac{p_n(\boldsymbol{z})}{a(\boldsymbol{z})}\left(-b(\boldsymbol{z}) + v\right).
\end{aligned} \tag{3.8}$$

This system is clearly composed of a *linear subsystem* of dimension r which determines the input-output behavior and of a (in general) *nonlinear subsystem* that albeit being forced by the input v does not affect the output — being, in fact, not observable. In particular, the linear subsystem now has the trivial transfer function

$$H(s) = \frac{1}{s^r}.$$

From this, it is possible to apply any linear control technique to impose a desired behavior to the output.

Remark 3.1.1. *If the system has relative degree $r = n$, then the feedback law* (3.7) *transforms the full system into a linear and controllable one. In fact, it is possible to do so if and only if there exists an output $\lambda(\boldsymbol{x})$ with respect to which the system has relative degree n [52, Lemma 4.2.1].*

Remark 3.1.2. *Note that the feedback law* (3.7) *can also be expressed directly in the original coordinates, taking the form*

$$u = \frac{1}{L_{\boldsymbol{g}} L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{x})}\left(-L_{\boldsymbol{f}} h(\boldsymbol{x}) + v\right).$$

3.1.2 The zero dynamics

Having analyzed the input-output relationship highlighted by the coordinate transformation and how this can be controlled by means of state feedback, we now analyze the role of the residual nonlinear dynamics described by the last $n - r$ equations of the normal form. For convenience, we partition the coordinates of the normal form as:

$$\boldsymbol{z} = (\boldsymbol{\xi}, \boldsymbol{\eta}) = (\underbrace{z_1, \ldots, z_r}_{\boldsymbol{\xi}}, \underbrace{z_{r+1}, \ldots, z_n}_{\boldsymbol{\eta}}).$$

With this notation, the normal form can be rewritten as

$$\begin{aligned}
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\vdots \\
\dot{z}_{r-1} &= z_r \\
\dot{z}_r &= b(\boldsymbol{\xi}, \boldsymbol{\eta}) + a(\boldsymbol{\xi}, \boldsymbol{\eta}) u \\
\dot{\boldsymbol{\eta}} &= \boldsymbol{q}(\boldsymbol{\xi}, \boldsymbol{\eta}) + \boldsymbol{p}(\boldsymbol{\xi}, \boldsymbol{\eta}) u.
\end{aligned} \tag{3.9}$$

Without loss of generality, one can assume that the mapping (3.3) is such that $\boldsymbol{\xi} = \mathbf{0}$ and $\boldsymbol{\eta} = \mathbf{0}$ at $\boldsymbol{x}'$. Then, if $\boldsymbol{x}'$ is an equilibrium point, $(\boldsymbol{\xi}, \boldsymbol{\eta}) = (\mathbf{0}, \mathbf{0})$ is an equilibrium point as well.

Consider the case in which the system is supposed to track an identically zero output. We are interested in finding all the initial states $\boldsymbol{x}^\circ$ and input functions $u^\circ(\cdot)$ such that the corresponding output $y(t)$ is identically zero for all t in a neighborhood of $t = 0$.

Recalling that in normal form the output is $y(t) = z_1(t)$ and the first r equations of the coordinate transformation (3.2), the constraint $y(t) = 0$ for all t clearly implies

$$\dot{z}_1(t) = \dot{z}_2(t) = \cdots = \dot{z}_r(t) = 0,$$

that is $\boldsymbol{\xi}(t) = \mathbf{0}$ for all t.

Then, the input $u(t)$ must be the *unique* solution to the equation $\dot{\boldsymbol{\xi}}(t) = \mathbf{0}$ for all t, with initial condition $\boldsymbol{\xi}(0) = \boldsymbol{\xi}^\circ = \mathbf{0}$:

$$0 = b(\mathbf{0}, \boldsymbol{\eta}(t)) + a(\mathbf{0}, \boldsymbol{\eta}(t))u(t). \tag{3.10}$$

The remaining part of the state $\boldsymbol{\eta}(t)$ is governed by the dynamics

$$\dot{\boldsymbol{\eta}} = \boldsymbol{q}(\mathbf{0}, \boldsymbol{\eta}) + \boldsymbol{p}(\mathbf{0}, \boldsymbol{\eta})u, \tag{3.11}$$

where the initial condition $\boldsymbol{\eta}(0) = \boldsymbol{\eta}^\circ$ can be chosen arbitrarily as it does not affect the output evolution.

Equation (3.10) has the solution

$$u(t) = -\frac{b(\mathbf{0}, \boldsymbol{\eta}(t))}{a(\mathbf{0}, \boldsymbol{\eta}(t))},$$

which, plugged into (3.11) yields the dynamics describing the evolution of $\boldsymbol{\eta}(t)$:

$$\dot{\boldsymbol{\eta}}(t) = \boldsymbol{q}(\mathbf{0}, \boldsymbol{\eta}(t)) - \boldsymbol{p}(\mathbf{0}, \boldsymbol{\eta}(t))\frac{b(\mathbf{0}, \boldsymbol{\eta}(t))}{a(\mathbf{0}, \boldsymbol{\eta}(t))}, \quad \boldsymbol{\eta}(0) = \boldsymbol{\eta}^\circ. \tag{3.12}$$

This dynamics is the so-called *zero dynamics* of the system, describing the internal behavior — unobservable from the output point of view — that the system exhibits when forced to follow the desired output.

It is worth noting that the unique input that satisfies condition (3.10) depends on $\boldsymbol{\eta}(t)$ and, ultimately, on the initial condition $\boldsymbol{\eta}^\circ$.

This derivation of the zero dynamics has followed a "closed-loop" approach, in which a feedback control law is devised to track the desired output from a suitable initial condition. On the other hand, it is important to notice that the concept of zero dynamics allows also for an "open-loop" definition as the dynamical system that characterizes the internal behavior of a system once initial conditions and inputs are chosen in such a way as to constrain the output to be identically zero [53]. This definition implies that, in fact, a zero dynamics might arise whenever the system is following an output trajectory independently on the way the commanded input has been generated. For this reason, it is natural to consider the effect of the zero dynamics on the system also when controlled using optimization-based methods such as MPC.

All the discussions around the zero dynamics are not limited to the case in which the output is $y(0) = 0$. In fact the concept of zero dynamics can be extended to the case in which the system is forced to track any arbitrary function $y_d(t)$. In such case, this forced dynamics describes the *internal* behavior of the system when the input and initial conditions are those such that the output is exactly equal to $y_d(t)$.

Definition 3.1.2 (Minimum and non-minimum phase system)**.** *A system is said to be minimum phase if its zero dynamics is stable. Conversely, a system is said to be non-minimum phase if its zero dynamics is unstable.*

It is well known that non-minimum phase systems pose a variety of challenges and limitations when designing feedback controllers [54]. In our context, it is important to stress the fact that, when trying to regulate the output of a non-minimum phase system two undesired things might occur:

- The internal dynamics, being unstable, might lead $\boldsymbol{\eta}(t)$ to diverge;
- As a consequence to the divergence of $\boldsymbol{\eta}(t)$, the input $u(t)$ required to follow the desired output might be unbounded.

In the following, we will try to address these problems by analyzing under which conditions the evolution of the zero dynamics and of the input remain bounded, and ultimately exploit these conditions to appropriately design MPC algorithms that are able to control systems which are non-minimum phase.

The following two remarks provide an additional perspective into the naming of the zero dynamics, which can be justified by an interesting parallel with linear systems:

Remark 3.1.3. *It can be shown that the zeros of the transfer function of a linear system coincide with the eigenvalues of its linear zero dynamics — when the system is converted in normal form.*

Remark 3.1.4. *The linear approximation at $\boldsymbol{\eta} = \mathbf{0}$ of the zero dynamics of a system coincides with the zero dynamics of the linear approximation at $\boldsymbol{x} = \mathbf{0}$, thus to the zeros of its transfer function. That is, the operations of linearization and the computation of the zero dynamics commute.*

In light of these two remarks, it also becomes apparent how the application of linear systems theory over the linearized zero dynamics can provide an effective, albeit local, tool to deal with systems which posses a zero dynamics.

Subsequently, the focus now shifts to linear time-invariant (LTI) systems, exploring the conditions under which the system evolution remains bounded. These conditions will play a pivotal role in devising stability constraints for implementing MPC in systems featuring an unstable zero dynamics.

3.2 Boundedness condition for LTI systems

In this section, we analyze the condition under which the evolution of a forced linear system remains bounded, even if the system is unstable. This condition will pose

the basis to formulate a so-called *stability constraint* to be imposed in the MPC problem in order to ensure that the evolution of the system does not diverge.

The following results are based on the contributions regarding the so-called *stable inversion* of linear systems [55]. These results have been used to generate stable feedforward trajectories, e.g., in the context of tracking control of flexible link robots [56], and have been later extended to control non-minimum phase nonlinear systems [57–60]. In [61], the concept was used for gait generation and was named as the *boundedness condition*, consisting in a proper initial condition for the LIPM that prevents the state from diverging, given a desired ZMP trajectory. Further investigations have also been carried in [62] for the time-varying dynamics of the Variable Height Inverted Pendulum model, where it was possible to find an exponential dichotomy to decouple the stable and unstable components of the dynamics.

While for the linear time invariant case the main result is a closed-form – albeit anticausal – condition, in the nonlinear case the most successful works have only been able to produce a iterative procedure to numerically generate such trajectories, also requiring a (possibly difficult to find) dichotomic split [63] of the system into stable and unstable subsystems for the time-varying case [60]. Arguably, this has narrowed the range of applicability of such methods to the case of real-time planning, and justifies our study of the linear version of the condition which will then be applied to the approximate linearization of a nonlinear system.

Consider a time-invariant linear system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$$

with $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and $\boldsymbol{u} \in \mathbb{R}^{n_u}$.

Without loss of generality, we can suppose that the input $\boldsymbol{u}$ is generated to track a desired output $\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}$. In such case, the system represents the *inverse* of the original one, possibly possessing a (linear) zero dynamics.

Assume that $\boldsymbol{A}$ has no eigenvalues with zero real part (i.e., the origin is a hyperbolic equilibrium point). Then, there exists a change of coordinates

$$\begin{pmatrix} \boldsymbol{x}_\mathrm{s} \\ \boldsymbol{x}_\mathrm{u} \end{pmatrix} = \begin{pmatrix} \boldsymbol{T}_\mathrm{s} \\ \boldsymbol{T}_\mathrm{u} \end{pmatrix} \boldsymbol{x} = \boldsymbol{T}\boldsymbol{x}$$

such that the stable and unstable subsystems are decoupled:

$$\begin{pmatrix} \dot{\boldsymbol{x}}_\mathrm{s} \\ \dot{\boldsymbol{x}}_\mathrm{u} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\Lambda}_\mathrm{s} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Lambda}_\mathrm{u} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_\mathrm{s} \\ \boldsymbol{x}_\mathrm{u} \end{pmatrix} + \begin{pmatrix} \boldsymbol{G}_\mathrm{s} \\ \boldsymbol{G}_\mathrm{u} \end{pmatrix} \boldsymbol{u},$$

with $\boldsymbol{\Lambda}_\mathrm{s}$ and $\boldsymbol{\Lambda}_\mathrm{u}$ characterized by the eigenvalues of $\boldsymbol{A}$ with negative and positive real part, respectively.

Clearly, in the decoupled system, the stable component $\boldsymbol{x}_\mathrm{s}$ has an evolution that is bounded if the input is bounded, and converges to the origin in free evolution. On the contrary, the unstable component $\boldsymbol{x}_\mathrm{u}$ will in general diverge[2].

The following lemma is essentially an adaptation of a result given in [66].

[2]The unstable component $\boldsymbol{x}_\mathrm{u}$ plays in our context a similar role to that of the capture point [64] (also known as the divergent component of motion [65]) in humanoid locomotion.

Lemma 3.2.1. *Assume that the input $\boldsymbol{u}(t)$ is such that $\|\boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(t)\|_\infty \leq \mu$, for some $\mu > 0$. Then, the state evolution $\boldsymbol{x}^*_{\mathrm{u}}(t)$ starting from the initial state*

$$\boldsymbol{x}^*_{\mathrm{u}}(t_0) = -\int_{t_0}^{\infty} e^{-\boldsymbol{\Lambda}_{\mathrm{u}}(\tau - t_0)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) d\tau \tag{3.13}$$

is bounded, i.e., there exists $\beta > 0$ such that $\|\boldsymbol{x}^*_{\mathrm{u}}(t)\|_\infty \leq \beta$.

Proof. The generic evolution of the unstable component $\boldsymbol{x}_{\mathrm{u}}$ starting from $\boldsymbol{x}_{\mathrm{u}}(t_0)$ with input $\boldsymbol{u}(t)$ can be written as

$$\boldsymbol{x}_{\mathrm{u}}(t) = e^{\boldsymbol{\Lambda}_{\mathrm{u}}(t-t_0)} \left(\boldsymbol{x}_{\mathrm{u}}(t_0) + \int_{t_0}^{t} e^{-\boldsymbol{\Lambda}_{\mathrm{u}}(\tau - t_0)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) d\tau \right).$$

Choosing the initial condition (3.13), the evolution becomes

$$\boldsymbol{x}^*_{\mathrm{u}}(t) = -\int_{t}^{\infty} e^{\boldsymbol{\Lambda}_{\mathrm{u}}(t-\tau)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) d\tau.$$

Since all eigenvalues of $\boldsymbol{\Lambda}_{\mathrm{u}}$ have strictly positive real part, there exist $\gamma, \alpha > 0$ such that

$$\|e^{\boldsymbol{\Lambda}_{\mathrm{u}}(t-\tau)}\|_\infty \leq \gamma\, e^{\alpha(t-\tau)} \quad \text{for} \quad t \leq \tau.$$

We can then write

$$\begin{aligned}
\|\boldsymbol{x}^*_{\mathrm{u}}(t)\|_\infty &\leq \int_{t}^{\infty} \left\| e^{\boldsymbol{\Lambda}_{\mathrm{u}}(t-\tau)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) \right\|_\infty d\tau \\
&\leq \int_{t}^{\infty} \left\| e^{\boldsymbol{\Lambda}_{\mathrm{u}}(t-\tau)} \right\|_\infty \|\boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau)\|_\infty d\tau \\
&\leq \gamma\mu \int_{t}^{\infty} e^{\alpha(t-\tau)} d\tau = \frac{\gamma\mu}{\alpha} = \beta.
\end{aligned}$$

∎

The interpretation of this lemma is the following: although the system includes an unstable part, for any bounded input there exists a particular initial condition from which the state evolution remains bounded. Such initial condition is anticausal, in the sense that it depends on the future values of the input.

The initial condition being anticausal also allows for an inverted perspective on the problem. Suppose that the initial condition $\boldsymbol{x}^*_{\mathrm{u}}(t_0)$ is fixed, which is typically the case in the context of feedback control. Then, the future evolution of the input must satisfy condition (3.13) in order to make the state trajectory bounded. This can be used, for instance, in the context of MPC, where the algorithm has to find the future evolution of the system that satisfies the constraints and minimizes the cost function, starting from the initial state at time t_k.

In this case, since the control horizon is limited, the integral can be split over two time intervals, the control horizon, and the so-called tail:

$$\begin{aligned}
\boldsymbol{x}^*_{\mathrm{u}}(t_k) &= -\int_{t_k}^{\infty} e^{-\boldsymbol{\Lambda}_{\mathrm{u}}(\tau - t_k)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) d\tau \\
&= \underbrace{-\int_{t_k}^{t_{k+C}} e^{-\boldsymbol{\Lambda}_{\mathrm{u}}(\tau - t_k)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) d\tau}_{\text{control horizon}} \underbrace{-\int_{t_{k+C}}^{\infty} e^{-\boldsymbol{\Lambda}_{\mathrm{u}}(\tau - t_k)} \boldsymbol{G}_{\mathrm{u}}\boldsymbol{u}(\tau) d\tau}_{\text{tail}}.
\end{aligned}$$

Denote with $\boldsymbol{u}_{\text{MPC}}$ the input $\boldsymbol{u}(t) \in [t_k, t_{k+C}]$ and with $\boldsymbol{u}_{\text{tail}}$ the input $\boldsymbol{u}(t) \in [t_{k+C}, \infty]$ after the end of the control horizon. From this, it is clear how this anti-causal condition depends, in principle, on a term that can be directly manipulated by the MPC controller via $\boldsymbol{u}_{\text{MPC}}$, being a function of the inputs during the control horizon, plus a term which depends on the future evolution of the input after the control horizon $\boldsymbol{u}_{\text{tail}}$, about which we can only conjecture.

On the other hand, the condition at time t_k can be integrated over the control horizon, yielding the equivalent condition

$$\boldsymbol{x}_{\text{u}}^*(t_{k+C}) = -\int_{t_{k+C}}^{\infty} e^{-\boldsymbol{\Lambda}_{\text{u}}(\tau - t_{k+C})} \boldsymbol{G}_{\text{u}} \boldsymbol{u}_{\text{tail}}(\tau) d\tau \tag{3.14}$$

on the terminal state $\boldsymbol{x}_{\text{u}}(t_{k+C})$. Notably, the integral condition here depends only on $\boldsymbol{u}_{\text{tail}}$, with the dependence of $\boldsymbol{x}_{\text{u}}(t_{k+C})$ on $\boldsymbol{u}_{\text{MPC}}$ being implicit.

3.3 The IS-MPC approach

We now introduce the IS-MPC method in general, before examining its usage through a series of application examples. Suppose we want to design an MPC controller able to perform output tracking while generating bounded state trajectories for a system

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_c(\boldsymbol{x}, \boldsymbol{u}). \tag{3.15}$$

While not being a requirement, one can assume that system (3.15) may have been obtained after having performed some form of feedback linearization over the original system, possibly decoupling its zero dynamics with respect to the linearized outputs. In such case, IS-MPC will be designed over this transformed dynamics and the input $\boldsymbol{u}$ will be mapped to the original input using the appropriate feedback transformation.

The MPC controller has to be designed to satisfy the two objectives:

- execute the main task with minimum tracking error;
- ensure that the system trajectories remain bounded.

Concerning the first point, it is typically sufficient to design the cost function to minimize deviations from the reference output. In some instances, if a preliminary feedback controller is used and IS-MPC is tasked with applying only a stabilizing corrective action — as is the case in the application described in Chapter 5 — it is sufficient to minimize the IS-MPC correction. As for the second point, we are interested in ensuring that the closed-loop trajectories of the system do not diverge for all future times. In the context of receding horizon control, simply imposing constraints on the limited control horizon might not be sufficient, as the terminal state might be such that no bounded feasible trajectory exists after the control horizon; In fact, in the case of simple regulation, terminal set constraints have been used to prove the stability of the closed-loop system [29, Chapter 2]. Such arguments are however difficult to apply in the case of tracking problems without relying on excessively conservative conditions, which might not be compatible with the first goal, or on the knowledge of a reference *state* trajectory, which might not be

available. Some methods that regulate the full *state* rely on a terminal cost obtained from the value function of an equivalent LQ problem. However, to obtain practical stabilization, artificial bounds on the controlled variables might still be introduced, leading to overly conservative limitations to the resulting motion (see Sect. 5.5.2 for an example of this in the Tractor-Trailer system). For this reasons, IS-MPC relies on Lemma 3.2.1 to impose a (terminal) condition that is exact — in the linear case — and that only constraints the unstable component of the dynamics, thus minimizing the impact on the tracking error.

In order to apply the method to a nonlinear system, we use the Linearized Time Varying MPC approach described in Sect. 2.4.1, linearizing the dynamics and the constraints around an auxiliary trajectory $(\boldsymbol{x}_{\text{aux}}(t), \boldsymbol{u}_{\text{aux}}(t))$. Moreover, we define a LTI approximation of the dynamics for $t \in [t_{k+C}, \infty]$:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}^* \boldsymbol{x} + \boldsymbol{B}^* \boldsymbol{u},$$

that will be used to impose the stability condition. This linearization can be chosen, for example, as the linearization around the auxiliary trajectory at the end of the control horizon $(\boldsymbol{x}_{\text{aux}}(t_{k+C}), \boldsymbol{u}_{\text{aux}}(t_{k+C}))$ or, more in general, around a suitable equilibrium.

Assuming that the conditions for the application of Lemma 3.2.1 hold, namely that the resulting dynamics is hyperbolic[3], we can decouple the unstable components of the dynamics with the projection matrix $\boldsymbol{T}_{\text{u}}$, satisfying

$$\dot{\boldsymbol{x}}_{\text{u}} = \boldsymbol{\Lambda}_{\text{u}}^* \boldsymbol{x}_{\text{u}} + \boldsymbol{G}_{\text{u}}^* \boldsymbol{u}, \quad t \in [t_{k+C}, \infty],$$

where $\boldsymbol{\Lambda}_{\text{u}}^*$ has all unstable eigenvalues. Then, impose the terminal condition (3.14) to obtain the *stability constraint*:

$$\boldsymbol{x}_{\text{u}}(t_{k+C}) = \boldsymbol{T}_{\text{u}} \boldsymbol{x}_{k+C} = -\int_{t_{k+C}}^{\infty} e^{-\boldsymbol{\Lambda}_{\text{u}}^*(\tau - t_{k+C})} \boldsymbol{G}_{\text{u}}^* \boldsymbol{u}_{\text{tail}}(\tau) d\tau \tag{3.16}$$

at the final time of the predicted MPC trajectory.

This condition imposes that the unstable component $\boldsymbol{x}_{\text{u}}$ at the end of the control horizon lands on a state for which its future evolution (after the control horizon) forced by the input $\boldsymbol{u}_{\text{tail}}$ remains bounded. Clearly, compared to any constraint on the evolution of the state over the control horizon only, this provides additional guarantees over the future evolution of the system and on the fact that the action planned by the MPC controller will make the evolution of the closed-loop system bounded. This becomes especially relevant when the control horizon of the MPC is relatively short, which could otherwise result in myopic actions that can eventually lead the system to diverge, even it the predicted trajectory is bounded (over the control horizon). We summarize the IS-MPC approach as follows:

1. Start by designing a controller to execute the desired task. This can be a preliminary feedback controller or can be part of the MPC itself.

[3] As will be shown later through an example, if the system can be decoupled into an hyperbolic and non-hyperbolic — but stabilizable through other means — part the procedure can be applied on the hyperbolic subsystem only.

2. Design the cost function of the MPC and the state/input constraints to execute the task.
3. Find an appropriate LTI approximation of the dynamics for $t \in [t_{k+C}, \infty]$;
4. Identify the stable and unstable components of the dynamics.
5. Impose the terminal stability constraint on the unstable component using the most suitable tail.

To best illustrate the method, we apply IS-MPC in the simple setting of a planar Wheeled Inverted Pendulum, an underactuated and unstable system that manifests its non-minimum phase nature when controlling the base position.

3.3.1 Application to a Wheeled Inverted Pendulum

Consider a planar Wheeled Inverted Pendulum robot (see Fig. 3.1) consisting of a wheel of radius R moving without slipping over the flat horizontal ground and of a rigid body hinged at the wheel axis. The system is described by the two generalized coordinates x, representing the position of the center of the wheel or equivalently of the contact point with the ground, and ϕ, representing the angle of the body with respect to the vertical position.

The wheel has a mass m_w and moment of inertia I_w with respect to its axis, where the center of mass is also assumed to be placed; the body has a mass m_b and moment of inertia I_b with respect to its center of mass, which is assumed to be placed at a distance l from the wheel axis.

Assume that the wheel is actuated by a motor and that the body is attached to the motor casing[4]. For convenience, we consider a control input $u = \tau/R$, where τ is the motor torque.

The equations of motion of the system can be obtained using the Lagrangian formalism with respect to the generalized coordinates $\boldsymbol{q} = (x, \phi)$, yielding:

$$\underbrace{\begin{pmatrix} m_b + m_w + \frac{I_w}{R^2} & m_b l\cos(\phi) \\ m_b l\cos(\phi) & I_b + m_b l^2 \end{pmatrix}}_{\boldsymbol{M}(\boldsymbol{q})} \ddot{\boldsymbol{q}} = \underbrace{\begin{pmatrix} m_b l \sin(\phi)\dot{\phi}^2 \\ m_b l g \sin(\phi) \end{pmatrix}}_{\boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}})} + \underbrace{\begin{pmatrix} 1 \\ -R \end{pmatrix}}_{\boldsymbol{b}} u, \tag{3.17}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is the symmetric and positive-definite inertia matrix and $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is vector of Coriolis and gravity forces acting on the system.

Being system (3.17) a second-order mechanical system, it can be easily transformed into its state-space formulation by setting $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}}) = (x, \phi, v_x, v_\phi) \in \mathbb{R}^{n_x}$, with $n_x = 4$, and left-multiplying the Lagrangian dynamics for the inverse of the inertia matrix:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \begin{pmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})\left(\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{b}u\right) \end{pmatrix} \\ &= \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})u, \end{aligned} \tag{3.18}$$

[4]This choice makes the body directly affected by the reaction forces. Alternatively, the body could be assumed to be on an axial bearing, making the joint passive — yet still coupled with the rest of the dynamics.

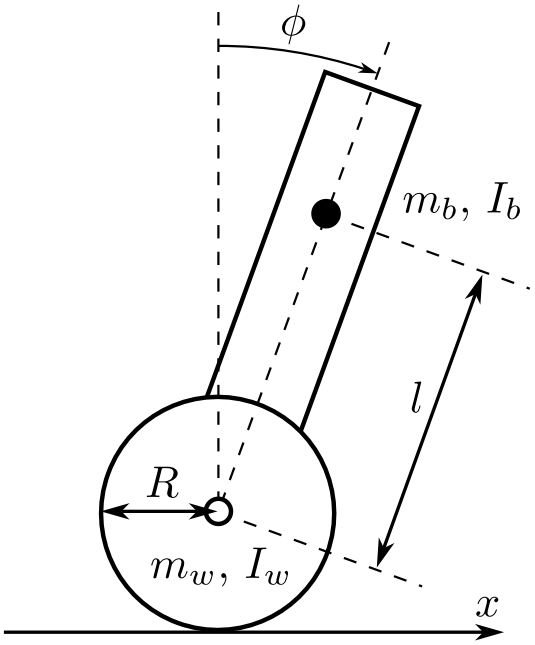

Figure 3.1. Graphical description of the Planar Wheeled Inverted Pendulum system.

where

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{pmatrix} v_x \\ v_\phi \\ \dfrac{m_b l(I_b + m_b l^2)\sin\phi\,\dot{\phi}^2 - m_b^2 l^2 g\cos\phi\sin\phi}{\det\boldsymbol{M}} \\ \dfrac{(m_b + m_w + I_w/R^2)m_b l g\sin\phi - m_b^2 l^2\cos\phi\sin\phi\,\dot{\phi}^2}{\det\boldsymbol{M}} \end{pmatrix},$$

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{pmatrix} 0 \\ 0 \\ \dfrac{I_b + m_b l^2 + m_b l R\cos\phi}{\det\boldsymbol{M}} \\ -\dfrac{(m_b + m_w + I_w/R^2)R + m_b l\cos\phi}{\det\boldsymbol{M}} \end{pmatrix},$$

and

$$\det\boldsymbol{M} = (I_b + m_b l^2)(m_w + I_w/R^2) + m_b I_b + m_b^2 l^2\sin^2\phi.$$

It can be easily verified that the origin $\boldsymbol{x} = \boldsymbol{0}$ is an unstable equilibrium of the system.

The task of choice is the tracking of a time-varying reference $y_d(t)$ for the base position x. Being the system underactuated, this is not a trivial task to perform if we are also interested in keeping the main body upright — i.e., in a vicinity of the upward equilibrium.

To this end, we will first perform the input-output linearization described in Sect. 3.1.1, linearize the zero dynamics around the upward equilibrium, and apply IS-MPC to the linearized system by designing a terminal stability constraint that stabilizes the system around the upward equilibrium.

Input-output linearization

It is straightforward to notice that the system has relative degree $r = 2$ with respect to the single output $y = h(\boldsymbol{x}) = x$. Being $r = n_x - 2$, we know that a two-dimensional zero dynamics will arise. Moreover, it is possible to put the system in normal form via the simple reordering of state variables $(\boldsymbol{\xi}, \boldsymbol{\eta}) = (x, v_x, \phi, v_\phi)$, and the input transformation

$$u = \frac{1}{L_{\boldsymbol{f}} L_{\boldsymbol{g}} h(\boldsymbol{x})} \left(v - L_{\boldsymbol{f}}^2 h(\boldsymbol{x}) \right),$$

where $v \in \mathbb{R}$ is the virtual input, resulting in

$$\dot{\boldsymbol{\xi}} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \boldsymbol{\xi} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} v, \tag{3.19}$$

$$\dot{\boldsymbol{\eta}} = \begin{pmatrix} \eta_2 \\ \dfrac{m_b l (g + R\eta_2^2) \sin \eta_1}{I_b + m_b l (l + R \cos \eta_1)} \end{pmatrix} + \begin{pmatrix} 0 \\ -\dfrac{(m_b + m_w + I_w/R^2) R + m_b l \cos \eta_1}{I_b + m_b l (l + R \cos \eta_1)} \end{pmatrix} v. \tag{3.20}$$

From this, it would be possible to make the base position converge exponentially to the desired trajectory $y_d(t)$ using the linear feedback

$$v(t) = \ddot{y}_d(t) + k_p (y_d(t) - \xi_1(t)) + k_d (\dot{y}_d(t) - \xi_2(t)),$$

with k_p, $k_d > 0$. However, as intuition suggests, this would result in the WIP falling to the ground as a result of the forced zero dynamics being unstable, thus diverging uncontrollably from the upward equilibrium.

To complete the task while keeping the body upright, it is instead necessary to make sure that the evolution of the (linearized) zero dynamics remains bounded.

Linearization around the upright equilibrium

For simplicity, we will linearize the dynamics around the upright equilibrium. That is, the auxiliary trajectory equates to $(\boldsymbol{x}_{\text{aux}}(t), \boldsymbol{u}_{\text{aux}}(t)) = (\boldsymbol{0}_4, 0)$, thus $\Delta \boldsymbol{x} = \boldsymbol{x}$, $\Delta \boldsymbol{u} = v$, so the MPC formulation can be equivalently expressed in the original coordinates. We use this linearization point for both the MPC prediction model and for the stability condition.

As a result of the input-output linearization, the dynamics of the base position (3.19) are that of a double integrator controlled by the virtual input v.

As far as the zero dynamics (3.20) is concerned, we perform a linearization around the equilibrium $\boldsymbol{\eta} = \boldsymbol{0}$, which results in a linearized dynamics

$$\dot{\boldsymbol{\eta}} = \begin{pmatrix} 0 & 1 \\ \alpha^2 & 0 \end{pmatrix} \boldsymbol{\eta} + \begin{pmatrix} 0 \\ \beta \end{pmatrix} v = \boldsymbol{A}_{\boldsymbol{\eta}} \boldsymbol{\eta} + \boldsymbol{B}_{\boldsymbol{\eta}} v, \tag{3.21}$$

where $\alpha = \sqrt{\frac{m_b l g}{I_b + m_b l^2}}$ and $\beta = -\frac{m_b l + (m_b + m_w + I_w/R^2) R}{I_b + m_b l (l + R)}$. The matrix $\boldsymbol{A}_{\boldsymbol{\eta}}$ has two eigenvalues $\lambda_{1,2} = \pm \alpha$, confirming that $\boldsymbol{\eta} = \boldsymbol{0}$ is an unstable equilibrium for the zero dynamics and that the system is non-minimum phase.

IS-MPC formulation

The goal of the MPC is to generate a sequence of inputs that minimize the position tracking error and that maintain the WIP upright during the motion. Concerning the tracking task, we consider a cost function

$$L(\boldsymbol{v}) = \sum_{i=0}^{C-1} \left(\|y_i - y_d(t_{k+i})\|^2 + Q_v \|v_i - \ddot{y}_d(t_{k+i})\|^2 \right) + \|y_C - y_d(t_{k+C})\|^2,$$

where Q_v is the input weight used to penalize deviations from the feedforward input.

To stabilize the system we impose a stability constraint on the unstable component of the zero dynamics. In fact, since the zero dynamics is completely decoupled from the output dynamics, we can impose the terminal condition (3.16) simply on the linearized zero dynamics (3.21). First, we note that we can apply the coordinate transformation

$$\begin{pmatrix} \eta_\mathrm{s} \\ \eta_\mathrm{u} \end{pmatrix} = \boldsymbol{T}\boldsymbol{\eta} = \begin{pmatrix} \boldsymbol{T}_\mathrm{s} \\ \boldsymbol{T}_\mathrm{u} \end{pmatrix} \boldsymbol{\eta} = \begin{pmatrix} 1 & -1/\alpha \\ 1 & 1/\alpha \end{pmatrix} \boldsymbol{\eta}$$

to obtain the diagonalized dynamics

$$\begin{pmatrix} \dot{\eta}_\mathrm{s} \\ \dot{\eta}_\mathrm{u} \end{pmatrix} = \begin{pmatrix} -\alpha & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} \eta_\mathrm{s} \\ \eta_\mathrm{u} \end{pmatrix} + \begin{pmatrix} -\beta/\alpha \\ \beta/\alpha \end{pmatrix} v.$$

Applying Lemma 3.2.1, we obtain that the unique bounded solution of the unstable subsystem has to satisfy the terminal condition

$$\eta_\mathrm{u}(t_{k+C}) = -\frac{\beta}{\alpha} \int_{t_{k+C}}^{\infty} e^{-\alpha(\tau - t_{k+C})} v_\mathrm{tail}(\tau) d\tau.$$

As discussed, being this integral the product of the evolution of the input $v_\mathrm{tail}(t)$ outside the time horizon, the MPC problem does not have a direct way to control this evolution, so we must perform some assumption on it. We consider two cases:

- *Anticipative tail:* if the evolution of the feedforward input $\ddot{y}_d(t)$ is assumed to be known for at least a preview window t_p after the control horizon[5], than we can set $v_\mathrm{tail}(t) = \ddot{y}_d(t)$ to compute the anticipative tail

  $$\eta_\mathrm{u}^* = -\frac{\beta}{\alpha} \int_{t_{k+C}}^{t_{k+C}+t_p} e^{-\alpha(\tau - t_{k+C})} \ddot{y}_d(\tau) d\tau,$$

  by assuming that the input will be equal to the desired one with no correction;

- *Truncated tail:* if no particular assumption can be made on the input, we consider $v_\mathrm{tail}(t) = 0$, yielding the truncated tail

  $$\eta_\mathrm{u}^* = 0.$$

[5]Thanks to the exponentially decreasing term in the integral, there is always in practice a preview time t_p after which the integral can be safely truncated with minimal impact on the result.

Regardless of the choice of tail, we impose on the predicted state at the end of the control horizon a stability constraint of the kind

$$\boldsymbol{T}_{\mathrm{u}}\boldsymbol{\eta}_C = \eta_{\mathrm{u}}^*.$$

It is worth noting that this condition does not constrain the state evolution to end at one particular state, as there are infinite combinations of $\boldsymbol{\eta} = (\phi, v_\phi)$ that could satisfy the constraint. For this reason, this is less restrictive than simply imposing a terminal state constraint $\boldsymbol{\eta} = \mathbf{0}$ to stabilize the system. Moreover, although the resulting evolution of $\boldsymbol{\eta}$ is bounded, we do not set an explicit hard constraint on the pitch angle ϕ nor on its velocity, which would unnecessarily restrict the range of motion of the system.

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be the state and input matrices describing the discrete-time prediction model obtained by the discretization of (3.19) and (3.21), so that $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}v_k$. The resulting IS-MPC formulation for this example is:

$$\begin{cases} \underset{\boldsymbol{v}}{\text{minimize}} & L(\boldsymbol{v}) & \\ \text{subject to} & \boldsymbol{x}_0 = \boldsymbol{x}_k & \\ & \boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}v_i & \forall i \in \mathbb{I}_0^{C-1} \\ & v_{\min} \leq v_i \leq v_{\max} & \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{T}_{\mathrm{u}}\boldsymbol{\eta}_C = \eta_{\mathrm{u}}^* & \end{cases}$$

If not for the stability constraint, the MPC controller would simply track the desired output, leaving the zero dynamics to diverge. With the introduction of the terminal condition, the MPC will be able to correct the motion so that the evolution of the unstable component remains bounded, and the robot body remains close to the upright configuration.

In order to showcase the peculiarities of this method, we compare it with what is the typical naive approach used to stabilize systems of this kind in the context of MPC, i.e., instead of using the stability constraint, we simply add to the cost function a regularization term penalizing the deviation of the pitch angle ϕ from the equilibrium:

$$\begin{cases} \underset{\boldsymbol{v}}{\text{minimize}} & L(\boldsymbol{v}) + \sum_{i=0}^{C-1} Q_\phi \phi_i^2 + Q_{\phi,C}\phi_C^2 & \\ \text{subject to} & \boldsymbol{x}_0 = \boldsymbol{x}_k & \\ & \boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}v_i & \forall i \in \mathbb{I}_0^{C-1} \\ & v_{\min} \leq v_i \leq v_{\max} & \forall i \in \mathbb{I}_0^{C-1} \end{cases}$$

Here, we consider two weights Q_ϕ, for the pitch angle over the control horizon, and $Q_{\phi,C}$ for the pitch angle at the terminal state to allow the possibility of having a comparatively larger $Q_{\phi,C}$ in order to reduce the penalization of the tracking error over the first part of the control horizon, improving on the closed-loop performance.

Results

We simulate system (3.18) under the action of both controllers in MATLAB, with a control frequency of 100 Hz. The dynamic parameters of the model are: $m_w =$

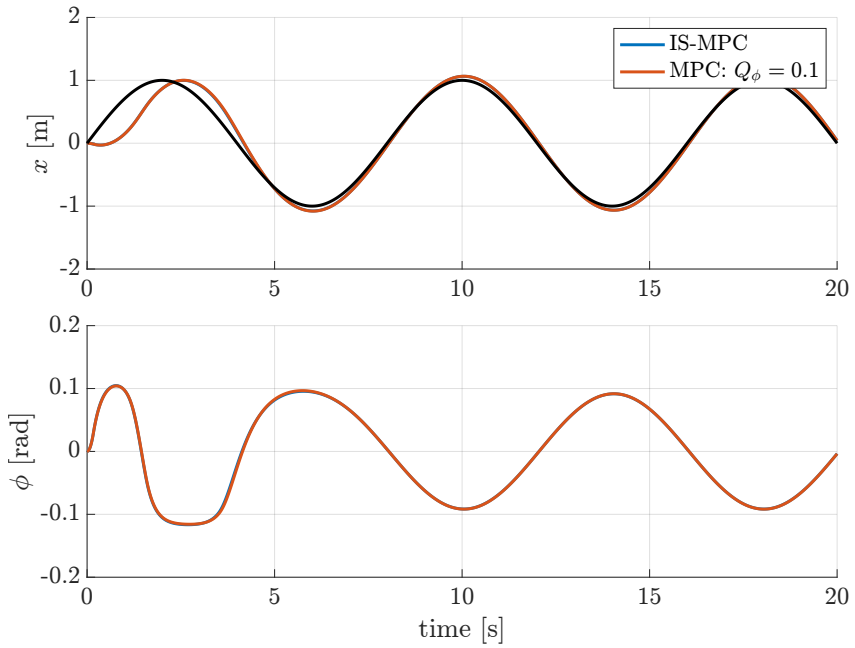

Figure 3.2. Comparison between IS-MPC and MPC with a control horizon $T_c = 1.5$ s. Reference $y_d(t)$ in black.

2.4 kg, $I_w = 0.0106$ kg m^2, $m_b = 18.36$ kg, $I_b = 1.5$ kg m^2, $l = 0.35$ m, $R = 0.13$ m. The desired task is the tracking of the sinusoidal reference $y_d(t) = \sin(2\pi f_s t)$ with frequency $f_s = 0.125$ Hz. The input is assumed to be limited to $(v_{\min}, v_{\max}) = (-1,\ 1)$ m/s^2

In the first test, we use a relatively long control horizon $T_c = 1.5$ s. For the cost functions, we set $Q_v = 0.01$, $Q_\phi = Q_{\phi,C} = 0.1$. As for the tail, we first consider the conservative choice of a truncated tail. The resulting (x, ϕ) trajectory, reported in Fig. 3.2, shows how in the case of a long control horizon, both methods provide essentially the same performance. This can easily be justified by the fact that — with long horizons — for IS-MPC the effect of a terminal constraint is small over the first steps of the trajectory and using a truncated tail is sufficient, while for MPC the regularization term over the pitch angle ϕ is enough to stabilize the motion around the unstable equilibrium while not perturbing the tracking of the reference position.

When a shorter control horizon $T_c = 0.8$ s is considered, the fundamental difference between the two methods starts to emerge from the results (see Fig. 3.3): In this case, IS-MPC experiences a larger tracking error, but the trajectory remains bounded. On the other hand, the standard MPC with $Q_\phi = 0.1$ diverges and the WIP falls to the ground. Only increasing the regularization cost to $Q_\phi = 1$ yields again a bounded trajectory, albeit with an unsatisfactory tracking error. It is then natural to try to tune the cost function to improve the tracking performance. For instance, this can be done by lowering the regularization cost Q_ϕ to reduce the effect on the tracking error and increasing the terminal cost $Q_{\phi,C}$ to penalize diverging behaviors. However, as can be observed in Fig. 3.4, we are not able to obtain a satisfactory performance by following this route, even though all cases result in a

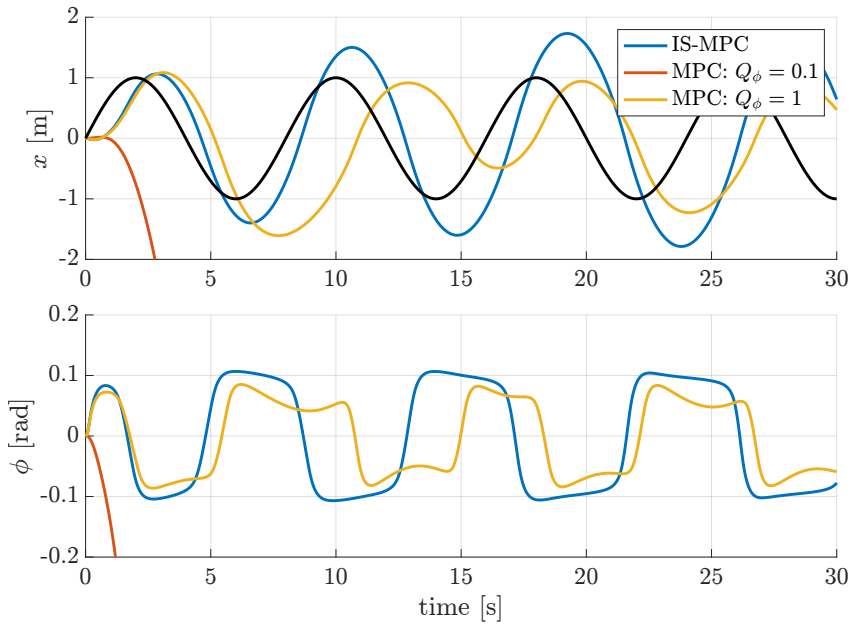

Figure 3.3. Comparison between IS-MPC and MPC with a control horizon $T_c = 0.8$ s. Reference $y_d(t)$ in black.

bounded trajectory. Indeed, this is one of the pitfalls of optimization-based control when the cost function has to be tuned to account for conflicting objectives.

For IS-MPC, on the other hand, it is possible to introduce a stability constraint based on an anticipative tail. In this case, reported in Fig. 3.5, it is possible to obtain a satisfactory tracking performance, with a residual error being the effect of linearization[6].

In conclusion, while IS-MPC does not provide substantial benefits over a more naive approach as adding a regularization term when the control horizon is long enough, it becomes quite clear that, when the horizon shortens, the ability to impose a constraint on the evolution of the unstable component can be extremely effective. The strength of the approach is twofold:

1. Since the stability is enforced via a constraint, it is not affected by the cost function weights and no tuning is involved, simplifying the MPC design.

2. As the constraint is imposed over the unstable component $\boldsymbol{x}_\mathrm{u}$, i.e., a lower-dimensional linear combination of the states, the effect of the constraint is minimal, since there exist infinite combinations of the state that can satisfy the constraint, allowing for the tracking error to be simultaneously minimized.

[6] In fact it can be shown that if the linearized model is used to simulate the closed-loop system, IS-MPC is able to achieve zero tracking error, while MPC has a residual error at all times.

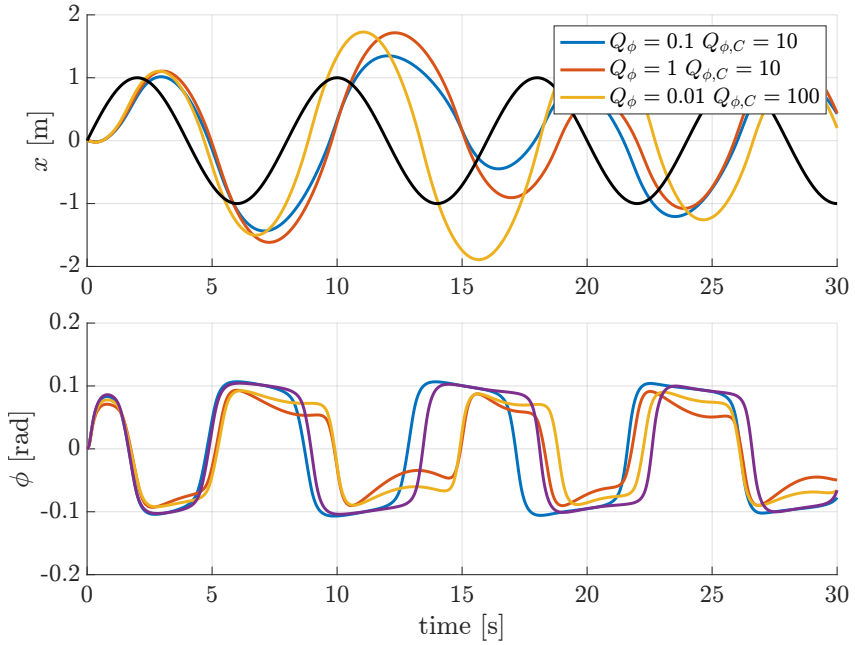

Figure 3.4. Result of tuning the cost function of the MPC problem with a control horizon $T_c = 0.8$ s. Reference $y_d(t)$ in black.

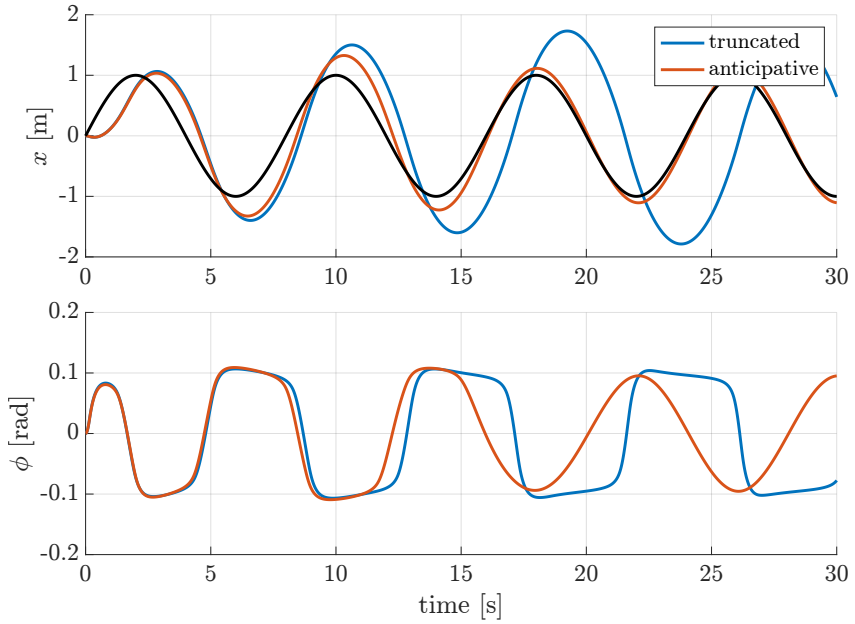

Figure 3.5. Result of the use of an anticipative tail as opposed to a truncated tail in IS-MPC, with a control horizon $T_c = 0.8$ s. Reference $y_d(t)$ in black.

Chapter 4

Stable Tracking Control of Articulated Balancing Robots

Balancing mobile robots are constituted by a statically unstable body, typically mounted on wheels, which can perform manipulation tasks by leveraging one or more actuated arms with multiple degrees of freedom. These platforms offer several advantages with respect to statically stable mobile robots. They are more suitable to traverse uneven terrains and they can perform fast and dynamic motions while carrying weights, moving at speeds comparable to those of humans. However, these capabilities implicitly require that the robot is able to maintain dynamic balance at all times in order to prevent falls. The underactuation of the dynamics renders this a rather complex control problem.

Early research studies on balancing mobile robots, both including transportation vehicles [67] and manipulators [68–70], highlighted the potentialities of these platforms which encouraged research groups to develop articulated robots that can also deliver manipulation capabilities, such as Golem Krang [71], or the more recent ALTER-EGO [72] shown in Fig. 4.1, and Boston Dynamics' Handle[1].

4.1 Related works

In the literature, several approaches have been proposed to control robots of this kind. It is possible for instance to model the platform as an inverted pendulum with the only goal of maintaining balance, and account for the manipulator motion as a disturbance to be compensated by the mobile base, e.g., via sliding mode control [73]. In [74] a PD control law has been used for balancing coupled with a PID controller for navigation while performance has been improved with a disturbance observer. Others, such as [75], propose a whole-body control law for *balancing*, which also allows the robot to gently interact with the environment. Task-space control was demonstrated to be applicable to WIP-based platforms in [76], allowing to maintain balance while controlling the end effector. This approach was further investigated by [77], where task-space control has been extended to the class of *planar* WIP robots in which the wheel motors are connected to the robot base

[1] https://youtu.be/5iV_hB08Uns?si=_v2DYWOJHXW396Of

Figure 4.1. ALTER-EGO, a prototype WIP robot developed at the University of Pisa [72].

link that is then subject to the wheel reaction torque. Being these methods based on classic (nonlinear) control, they do not allow to incorporate constraints, e.g., to account for input saturation, which can be an issue when performing dynamic motions.

A few works have also explored the use of MPC. Feedback linearization in conjunction with MPC was used in [78] for a wheeled inverted pendulum without arms. In [79] a hierarchical structure is proposed, composed by an MPC-based reference trajectory generator and an inverse dynamics controller to track the reference trajectories. Whole-body MPC algorithms instead allow to jointly perform re-planning and control, but their implementation can be cost-demanding due to the complex dynamics of the robots. Remarkable results were achieved in [80] for a Ballbot-like omnidirectional mobile manipulator, where real-time performance was achieved by exploiting custom nonlinear solvers. These methods, although proven to be effective, do not directly address the stability problem related to the unstable balancing dynamics. In fact, it is often required for the MPC to have a long prediction horizon in order to generate stable behaviors, either by regulating to zero the body pitch angle — possibly limiting performance — or by tracking a pre-computed pitch trajectory — which unavoidably requires an additional level of trajectory planning which might limit the flexibility in the application of the method to dynamic scenarios.

4.2 Contribution

In this chapter, we illustrate a whole-body MPC controller for WIP robots which explicitly addresses the instability problem using the IS-MPC method. The scheme has a general formulation that can accommodate different objectives. In particular, we report examples of navigation and loco-manipulation tasks. In order to reshape

the system in a convenient way, we perform MPC on a partially feedback-linearized system. This structure allows us to address the instability problem by introducing an explicit stability constraint on the pitch dynamics based on the boundedness condition of Sect. 3.2. The key advantages of the proposed method can be summarized as follows:

- it can be applied to 3D WIP robots with any number of arms and degrees of freedom;
- the motion generation is achieved in real-time and fully *online*, only requiring a desired task trajectory;
- the use of the explicit stability constraint on the unstable component of the dynamics allows for short prediction times and reduces the effect of tuning on the stability of the system;
- being based on MPC, it is possible to include state/input constraints or other general task constraints, such as collision avoidance.

The chapter is organized as follows. Section 4.3 gives some background concepts by introducing the robot model, the partial feedback linearization and the generic task definition. Section 4.4 contains an overview of the proposed approach and describes in detail the IS-MPC algorithm. Simulation results obtained performing two different tasks are reported in Sect. 4.5.

4.3 The control problem

In this section we introduce the dynamic model of the considered WIP robot, describe the partial feedback linearization procedure and define the task to be executed.

4.3.1 Modeling

The configuration of a WIP robot is defined as $\boldsymbol{q} = (x, y, \theta, \phi, \boldsymbol{q}_r, \boldsymbol{q}_l)$, where (x, y) is the position of the differential-drive robot base in the world frame, θ and ϕ are respectively the yaw angle and the pitch angle of the body, and $\boldsymbol{q}_r \in \mathbb{R}^{n_r}$ and $\boldsymbol{q}_l \in \mathbb{R}^{n_l}$ are the right and left arm joint angles, respectively. The total number of dofs in the arms is $n_a = n_r + n_l$. Figure 4.2 shows the schematic of an example WIP robot having $n_r = n_l = 2$.

Denote the velocity vector by $\boldsymbol{\nu} = (v, \omega, v_\phi, \boldsymbol{v}_r, \boldsymbol{v}_l)$, where v is the pseudovelocity of the robot base, $\omega = \dot{\theta}$, $v_\phi = \dot{\phi}$, $\boldsymbol{v}_r = \dot{\boldsymbol{q}}_r$ and $\boldsymbol{v}_l = \dot{\boldsymbol{q}}_l$. The robot state is then $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{\nu})$.

A reduced-order dynamic model of the WIP robot can be expressed in compact form as [81]

$$\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\,\boldsymbol{\nu} \tag{4.1}$$

$$\dot{\boldsymbol{\nu}} = -\boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{\nu}) - \boldsymbol{E}\,\boldsymbol{\tau}), \tag{4.2}$$

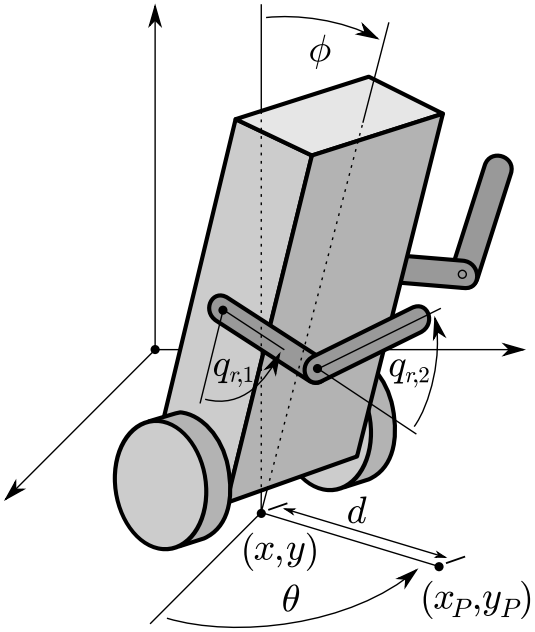

Figure 4.2. Generalized coordinates for a WIP robot with $n_r = n_l = 2$ (left arm coordinates are omitted for clarity). Note the displaced point on the ground (x_P, y_P) used for navigation tasks.

where $\boldsymbol{G}$ is a matrix whose columns span the null space of the Pfaffian nonholonomic constraint on the robot base, $\boldsymbol{M}$ is the inertia matrix, $\boldsymbol{c}$ is a force vector containing the gravity, centrifugal and Coriolis contributions, $\boldsymbol{E}$ is the actuator selection matrix and $\boldsymbol{\tau} \in \mathbb{R}^{2+n_a}$ are the torques acting on the base wheels and the arm joints.

Due to the presence of the articulated arms and the associated inertial couplings, the above model is considerably more complicated than in the case of a WIP with no arms (see [82] for a model of the latter). Therefore, we omit the explicit expression of the various matrices in eqs. (4.1–4.2), with the exception of the actuator selection matrix which takes the form

$$\boldsymbol{E} = \begin{pmatrix} 1/R & 1/R & \mathbf{0}_{n_a}^T \\ a/R & -a/R & \mathbf{0}_{n_a}^T \\ -1 & -1 & \mathbf{0}_{n_a}^T \\ \mathbf{0}_{n_a} & \mathbf{0}_{n_a} & \boldsymbol{I}_{n_a} \end{pmatrix},$$

where R is the wheel radius and a is the semi-distance between the wheels. The linear dependence of the first and third rows is a consequence of the fact that the pitch angle ϕ is not independently actuated — which makes the WIP robot an underactuated system. In general, full expressions for the dynamics can be obtained in symbolic form by the use of tools such as the MATLAB Symbolic Toolbox or Wolfram Mathematica. Alternatively, libraries such as Pinocchio [83] can be used to efficiently compute the full dynamics and its derivatives numerically — from exact analytical expressions — starting from a representation of the system dynamics and kinematics in a text file.

4.3.2 Partial feedback linearization

To simplify the design of our control scheme, we perform a partial feedback linearization of model (4.1–4.2).

Let us focus on the following subset of equations from (4.2):

$$\begin{pmatrix} \dot{v} \\ \dot{\omega} \\ \dot{\boldsymbol{v}}_r \\ \dot{\boldsymbol{v}}_l \end{pmatrix} = \boldsymbol{\alpha}(\boldsymbol{x}) + \boldsymbol{\Psi}(\boldsymbol{x})\boldsymbol{\tau}.$$

Define the following input transformation

$$\boldsymbol{\tau} = \boldsymbol{\Psi}^{-1}(\boldsymbol{x})(\boldsymbol{u} - \boldsymbol{\alpha}(\boldsymbol{x})), \tag{4.3}$$

where $\boldsymbol{u}$ are the new transformed inputs. Using (4.3) in (4.1–4.2) results in a partially linearized dynamics which is conveniently reordered as

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \\ \dot{v} &= u_1 \\ \dot{\omega} &= u_2 \\ \dot{\boldsymbol{q}}_r &= \boldsymbol{v}_r \\ \dot{\boldsymbol{v}}_r &= \boldsymbol{u}_3 \\ \dot{\boldsymbol{q}}_l &= \boldsymbol{v}_l \\ \dot{\boldsymbol{v}}_l &= \boldsymbol{u}_4 \\ \dot{\boldsymbol{\eta}} &= \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{x}, \boldsymbol{u}), \end{aligned} \tag{4.4}$$

having set $\boldsymbol{\eta} = (\phi, v_\phi)$. The vector field describing the pitch dynamics is now

$$\boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{S}(\boldsymbol{x})\boldsymbol{\Psi}^{-1}(\boldsymbol{x})\left(\boldsymbol{u} - \boldsymbol{\alpha}(\boldsymbol{x})\right),$$

where $\boldsymbol{r}$ and $\boldsymbol{S}$ are respectively the drift and the input matrix characterizing the original pitch dynamics in (4.1–4.2).

As intuition suggests, the pitch dynamics is inherently unstable in this kind of system [84]. For instance, one may easily verify that in a WIP robot the linearized pitch dynamics around the static equilibrium (CoM over the wheel axis) has a positive eigenvalue. A similar situation holds when the linearization of the pitch dynamics is computed around generic trajectories.

4.3.3 Task definition

We will consider general task functions defined as

$$\boldsymbol{r} = \boldsymbol{h}(\boldsymbol{q}), \tag{4.5}$$

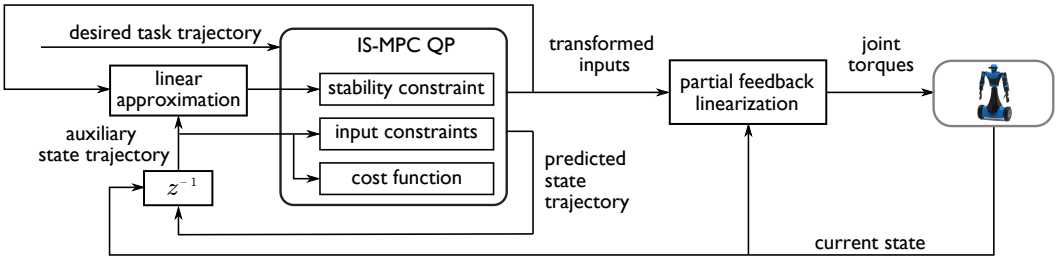

Figure 4.3. A block scheme of the proposed approach. The z^{-1} block stores the solution of each iteration and makes it available at the next, after replacing its first sample with the current state, see (4.6).

with $\boldsymbol{r} \in \mathbb{R}^m$. By differentiating (4.5) we obtain

$$\dot{\boldsymbol{r}} = \frac{\partial \boldsymbol{h}(\boldsymbol{q})}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = \frac{\partial \boldsymbol{h}(\boldsymbol{q})}{\partial \boldsymbol{q}} \boldsymbol{G}(\boldsymbol{q}) \boldsymbol{\nu} = \boldsymbol{J}(\boldsymbol{q}) \boldsymbol{\nu},$$

where $\boldsymbol{J}(\boldsymbol{q})$ is the $m \times (3 + n_a)$ task Jacobian.

For the simulations, we will consider two specific task functions. The first (*navigation* task) describes the coordinates (x_P, y_P) of a point P on the ground which is displaced[2] from the base by a distance d along the sagittal axis (see Fig. 4.2). The task function and the associated task Jacobian are easily computed as

$$\boldsymbol{h}(\boldsymbol{q}) = \begin{pmatrix} x_P \\ y_P \end{pmatrix} = \begin{pmatrix} x + d\cos\theta \\ y + d\sin\theta \end{pmatrix},$$

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{pmatrix} \cos\theta & -d\sin\theta & \mathbf{0}^T_{n_a+1} \\ \sin\theta & d\cos\theta & \mathbf{0}^T_{n_a+1} \end{pmatrix}.$$

While the values of the arm generalized coordinates have no instantaneous effect on the navigation task, arm motions can dynamically contribute to the stabilization of the pitch dynamics.

The second task function used in the simulations will be the position of the end-effector, chosen as one of the two hands (*loco-manipulation* task). In this case, all generalized coordinates directly contribute to the task.

4.4 The proposed approach

In this section we discuss the proposed method for controlling WIP robots. We will first provide a general overview of the solution approach and then proceed to a detailed discussion of its main component, the IS-MPC controller.

4.4.1 Overview

The problem we address is that of generating in real-time a whole-body motion of the WIP robot such that:

[2] A nonzero displacement guarantees that $\boldsymbol{J}(\boldsymbol{q})$ will be full rank, which is an implicit requirement of our MPC-based approach.

- the task function $\boldsymbol{r}(t)$ tracks a desired reference trajectory $\boldsymbol{r}_d(t)$;
- balance is maintained;
- constraints on both states (joint limits, velocity limits) and inputs (torque limits) are satisfied.

A block scheme of the proposed solution approach is shown in Fig. 4.3. The input is the task trajectory $\boldsymbol{r}_d(t)$ to be tracked. At each iteration, the IS-MPC algorithm solves an optimization problem over a receding control horizon to compute the transformed inputs $\boldsymbol{u}$, from which the original torque inputs $\boldsymbol{\tau}$ will be reconstructed. To achieve real-time performance, the optimization problem is formulated as a Quadratic Program (QP) using the Linearized Time Varying MPC approach, with an auxiliary trajectory obtained from the solution predicted at the previous iteration, and using the same trajectory to compute the cost function as well as to map torque limits to linear constraints on the transformed inputs.

An essential component of the IS-MPC algorithm is the explicit stability constraint included in the QP. Such constraint will avoid the onset of instability in the pitch dynamics, ultimately guaranteeing that the robot maintains balance while executing the assigned task.

In the next section we describe in detail the IS-MPC algorithm and its various components.

4.4.2 IS-MPC

Below we introduce the prediction model, the stability constraint, the input constraints and the resulting MPC formulation.

Prediction model

At the k-th iteration, the prediction model is obtained by linearizing (4.4) around an auxiliary trajectory defined as follows. Denote by

$$(\boldsymbol{x}_{k-1}, \boldsymbol{x}_{k|k-1}, \boldsymbol{x}_{k+1|k-1} \ldots, \boldsymbol{x}_{k+C-1|k-1})$$

the predicted state trajectory at the $(k-1)$-th iteration, where $\boldsymbol{x}_{k-1}$ is the current state at t_{k-1} and the following samples $\boldsymbol{x}_{k|k-1}, \boldsymbol{x}_{k+1|k-1} \ldots, \boldsymbol{x}_{k+C-1|k-1}$ are obtained by injecting the MPC solution inputs at t_{k-1} into the corresponding prediction model[3]. From this trajectory, we can build the *auxiliary* trajectory $(\bar{\boldsymbol{x}}_k, \ldots, \bar{\boldsymbol{x}}_{k+C})$ by letting

$$\bar{\boldsymbol{x}}_{k+i} = \begin{cases} \boldsymbol{x}_k & i = 0, \\ \boldsymbol{x}_{k+i|k-1} & i = 1, \ldots, C-1 \\ \bar{\boldsymbol{x}}_{k+C-1} & i = C. \end{cases} \tag{4.6}$$

The current state $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$ is used as first sample of the auxiliary trajectory to increase its precision and therefore the accuracy of the subsequent linearization procedure. Note also that the auxiliary trajectory is prolonged up to t_{k+C} by

[3] At the first iteration, when the system is at rest and a previous solution is not available, the inputs are simply set to zero.

replicating the last sample $\boldsymbol{x}_{k+C-1|k-1}$ of the predicted state trajectory at the $(k-1)$-th iteration. The auxiliary input is the MPC solution at t_{k-1}:

$$(\bar{\boldsymbol{u}}_k, \ldots, \bar{\boldsymbol{u}}_{k+C-1}).$$

The prediction model at t_k can now be computed as the linear approximation of the partially feedback-linearized dynamics (4.4) around the auxiliary trajectory. Since the base position will not appear explicitly in the MPC prediction, we introduce the reduced state vector $\boldsymbol{\xi} = (v, \theta, \omega, \boldsymbol{q}_r, \boldsymbol{v}_r, \boldsymbol{q}_l, \boldsymbol{v}_l, \boldsymbol{\eta})$ to define the discrete time prediction model used in the MPC.

In particular, for the pitch dynamics we will use the following model

$$\boldsymbol{\eta}_{i+1} = \bar{\boldsymbol{A}}_{\boldsymbol{\eta},k+i}\boldsymbol{\eta}_i + \bar{\boldsymbol{B}}_{\boldsymbol{\eta},k+i}\,\boldsymbol{u}_i + \bar{\boldsymbol{f}}_{\boldsymbol{\eta},k+i}, \tag{4.7}$$

for $i = 0, \ldots, C-1$, where $\bar{\boldsymbol{A}}_{\boldsymbol{\eta},k+i}$, $\bar{\boldsymbol{B}}_{\boldsymbol{\eta},k+i}$ and $\bar{\boldsymbol{f}}_{\boldsymbol{\eta},k+i}$ are obtained discretizing the linearization:

$$\begin{aligned}
\bar{\boldsymbol{A}}_{\eta,c}(t_{k+i}) &= \left.\frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}}{\partial \boldsymbol{\eta}}\right|_{\bar{\boldsymbol{x}}_{k+i},\bar{\boldsymbol{u}}_{k+i}}, \quad \bar{\boldsymbol{B}}_{\eta,c}(t_{k+i}) = \left.\frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}}{\partial \boldsymbol{u}}\right|_{\bar{\boldsymbol{x}}_{k+i},\bar{\boldsymbol{u}}_{k+i}}, \\
\bar{\boldsymbol{f}}_{\eta,c}(t_{k+i}) &= \boldsymbol{f}_{\boldsymbol{\eta}}(\bar{\boldsymbol{x}}_{k+i}, \bar{\boldsymbol{u}}_{k+i}) - \bar{\boldsymbol{A}}_{\eta,c}(t_{k+i})\bar{\boldsymbol{\eta}}_{k+i} - \bar{\boldsymbol{B}}_{\eta,c}(t_{k+i})\bar{\boldsymbol{u}}_{k+i}.
\end{aligned}$$

Note that this approximation neglects the effect of the variation of the arm coordinates $\boldsymbol{q}_r$, $\boldsymbol{q}_l$ with respect to the auxiliary trajectory, decoupling the pitch dynamics from the rest of the system.

Combining the discretized linear dynamics of (4.4) with (4.7), we can define the prediction model for the reduced dynamics:

$$\boldsymbol{\xi}_{i+1} = \boldsymbol{A}_{\boldsymbol{\xi},k+i}\boldsymbol{\xi}_i + \boldsymbol{B}_{\boldsymbol{\xi},k+i}\boldsymbol{u}_i + \boldsymbol{f}_{\boldsymbol{\xi},k+i}, \tag{4.8}$$

where

$$\boldsymbol{A}_{\boldsymbol{\xi},k+i} = \begin{pmatrix} \boldsymbol{A}_{\text{base}} & & \\ & \boldsymbol{A}_{\text{arms}} & \\ & & \bar{\boldsymbol{A}}_{\boldsymbol{\eta},k+i} \end{pmatrix}, \; \boldsymbol{B}_{\boldsymbol{\xi},k+i} = \begin{pmatrix} \boldsymbol{B}_{\text{base}} \\ \boldsymbol{B}_{\text{arms}} \\ \bar{\boldsymbol{B}}_{\boldsymbol{\eta},k+i} \end{pmatrix}, \; \boldsymbol{f}_{\boldsymbol{\xi},k+i} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \bar{\boldsymbol{f}}_{\boldsymbol{\eta},k+i} \end{pmatrix},$$

$$\boldsymbol{A}_{\text{base}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \delta_t \\ 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{B}_{\text{base}} = \begin{pmatrix} \delta_t & 0 & \boldsymbol{0}_{n_a}^T \\ 0 & \frac{\delta_t^2}{2} & \boldsymbol{0}_{n_a}^T \\ 0 & \delta_t & \boldsymbol{0}_{n_a}^T \end{pmatrix},$$

$$\boldsymbol{A}_{\text{arms}} = \begin{pmatrix} \boldsymbol{I}_{n_r} & \delta_t \boldsymbol{I}_{n_r} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{n_r} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_{n_l} & \delta_t \boldsymbol{I}_{n_l} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_{n_l} \end{pmatrix}, \quad \boldsymbol{B}_{\text{arms}} = \begin{pmatrix} \boldsymbol{0}_{n_r \times 2} & \frac{\delta_t^2}{2}\boldsymbol{I}_{n_r} & \boldsymbol{0} \\ \boldsymbol{0}_{n_r \times 2} & \delta_t \boldsymbol{I}_{n_r} & \boldsymbol{0} \\ \boldsymbol{0}_{n_l \times 2} & \boldsymbol{0} & \frac{\delta_t^2}{2}\boldsymbol{I}_{n_l} \\ \boldsymbol{0}_{n_l \times 2} & \boldsymbol{0} & \delta_t \boldsymbol{I}_{n_l} \end{pmatrix}.$$

Stability constraint

As already mentioned, the pitch dynamics in a WIP robot is unstable. This is obviously true also for its linear approximation (4.7). To cope with this instability,

we include in the MPC formulation a constraint in order to guarantee that the evolution of ϕ does not diverge. Note that this is conceptually different from enforcing a box constraint on ϕ, which would unnecessarily restrict the range of admissible motions for the system. To formulate such a constraint, we will retain the time-varying prediction model (4.7) for $t < t_{k+C}$, and use a time-invariant approximation for $t \geq t_{k+C}$.

In particular, let $\boldsymbol{x}^* = (\boldsymbol{q}^*, \mathbf{0}_{3+n_a})$ and

$$\boldsymbol{q}^* = (\bar{x}_{k+C}, \bar{y}_{k+C}, \bar{\theta}_{k+C}, \phi^*, \bar{\boldsymbol{q}}_{r,k+C}, \bar{\boldsymbol{q}}_{l,k+C}). \tag{4.9}$$

where the configuration at the last sample of the auxiliary trajectory has been used, with the exception of $\bar{\phi}_{k+C}$ which has been replaced by ϕ^*, the value of the pitch angle that puts the WIP in static equilibrium when the arms are at $\bar{\boldsymbol{q}}_{r,k+C}$, $\bar{\boldsymbol{q}}_{l,k+C}$. The time-invariant approximation that we will use for $t \geq t_{k+C}$ is defined as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{A}^*_{\boldsymbol{\eta}} \boldsymbol{\eta} + \boldsymbol{f}^*_{\boldsymbol{\eta}} \tag{4.10}$$

where the dynamic matrix and the constant drift, respectively

$$\boldsymbol{A}^*_{\boldsymbol{\eta}} = \left. \frac{\partial \boldsymbol{f}_{\boldsymbol{\eta}}}{\partial \boldsymbol{\eta}} \right|_{\boldsymbol{x}^*, \mathbf{0}}, \qquad \boldsymbol{f}^*_{\boldsymbol{\eta}} = \boldsymbol{f}_{\boldsymbol{\eta}}(\boldsymbol{x}^*, \mathbf{0}) - \boldsymbol{A}^*_{\boldsymbol{\eta}} \boldsymbol{\eta}^*,$$

are computed at $\boldsymbol{x} = \boldsymbol{x}^*$ and $\boldsymbol{u} = \mathbf{0}$, and we have set $\boldsymbol{\eta}^* = (\phi^*, 0)$.

The particular choice of the state and inputs at which the linear approximation is frozen can be justified by noting that the state $\boldsymbol{x}^*$ is a static equilibrium for system (4.4) which requires no input ($\boldsymbol{u} = \mathbf{0}$). In the terminology of [50], this corresponds to using a *truncated tail* to derive the stability constraint. In practice, this is the most viable option due to the fact that the value of the input at t_{k+C} is neither a decision variable in the current QP nor available from the solution of the previous QP, and, in the context of our problem with a generic task, there is no particular input trajectory that could be assumed to be a suitable candidate to anticipate the behavior of the system to use an *anticipative tail*.

At this point, let us perform a coordinate transformation

$$\begin{pmatrix} \eta_{\mathrm{s}} \\ \eta_{\mathrm{u}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{T}^*_{\mathrm{s}} \\ \boldsymbol{T}^*_{\mathrm{u}} \end{pmatrix} \boldsymbol{\eta} = \boldsymbol{T}^* \boldsymbol{\eta},$$

with $\boldsymbol{T}^*$ chosen in such a way that the unstable component η_{u} evolves according to

$$\dot{\eta}_{\mathrm{u}} = \lambda^*_{\mathrm{u}} \eta_{\mathrm{u}} + f^*_{\mathrm{u}}, \tag{4.11}$$

where $\lambda^*_{\mathrm{u}} > 0$ is the positive eigenvalue of $\boldsymbol{A}^*_{\boldsymbol{\eta}}$ in the time-invariant linearized pitch dynamics (4.10) and f^*_{u} is the corresponding constant drift on η_{u}.

As discussed in Sect. 3.2, in spite of the instability of (4.11) the trajectory of $\boldsymbol{\eta}(t)$ in (4.10) is guaranteed to be bounded for $t > t_{k+C}$ provided that the following stability condition is satisfied

$$\eta_{\mathrm{u}}(t_{k+C}) = -\int_{t_{k+C}}^{\infty} e^{-\lambda^*_{\mathrm{u}}(\tau - t_{k+C})} f^*_{\mathrm{u}} \, d\tau.$$

In this case, due to the presence of the constant drift with a truncated tail, the integral can be computed in closed form, yielding the stability condition:

$$\boldsymbol{T}_{\mathrm{u}}^{*}\boldsymbol{\eta}_{C} = -\frac{f_{\mathrm{u}}^{*}}{\lambda_{\mathrm{u}}^{*}}. \tag{4.12}$$

Note that this condition is updated at each control cycle to reflect the changes in the local linearization (4.10), the coordinate transformation matrix $\boldsymbol{T}^{*}$ and the resulting unstable dynamics (4.11).

Torque constraints

The input constraints account for the physical limitations of the robot actuators, which take the form

$$-\boldsymbol{\tau}_{M} \leq \boldsymbol{\tau}_{k+i} \leq \boldsymbol{\tau}_{M} \qquad i = 0, \ldots, C-1, \tag{4.13}$$

where $\boldsymbol{\tau}_{M}$ is the vector of symmetric torque limits. In view of the nonlinear mapping (4.3) between torques $\boldsymbol{\tau}$ and inputs $\boldsymbol{u}$, to transform (4.13) into a linear constraint on $\boldsymbol{u}$ we simply evaluate the mapping along the auxiliary trajectory, obtaining

$$\left(\bar{\boldsymbol{\Psi}}_{k+i}\right)^{-1}\bar{\boldsymbol{\alpha}}_{k+i} - \boldsymbol{\tau}_{M} \leq \left(\bar{\boldsymbol{\Psi}}_{k+i}\right)^{-1}\boldsymbol{u}_{i} \leq \left(\bar{\boldsymbol{\Psi}}_{k+i}\right)^{-1}\bar{\boldsymbol{\alpha}}_{k+i} + \boldsymbol{\tau}_{M} \tag{4.14}$$

where $\bar{\boldsymbol{\Psi}}_{k+i} = \boldsymbol{\Psi}(\bar{\boldsymbol{x}}_{k+i})$ and $\bar{\boldsymbol{\alpha}}_{k+i} = \boldsymbol{\alpha}(\bar{\boldsymbol{x}}_{k+i})$.

Since in the auxiliary trajectory we have set $\bar{\boldsymbol{x}}_{k} = \boldsymbol{x}_{k}$, the linear constraint (4.14) on the first input $\boldsymbol{u}_{k}$ — which is the only one actually applied in a MPC algorithm — is exactly equivalent to the original constraint (4.13) on $\boldsymbol{\tau}_{k}$.

Cost function

The cost function of our QP is built by adding three different terms aimed at solving the tracking problem.

The first term is designed in such a way to implement a kinematic control law for the task variables $\boldsymbol{r}(t)$. In particular, we let

$$\dot{\boldsymbol{r}}_{r}(t) = \dot{\boldsymbol{r}}_{d}(t) + \boldsymbol{K}\left(\boldsymbol{r}_{d}(t) - \boldsymbol{r}(t)\right), \tag{4.15}$$

where $\boldsymbol{K}$ is a positive definite diagonal matrix, and ask IS-MPC to generate inputs that force $\dot{\boldsymbol{r}}$ to track[4] $\dot{\boldsymbol{r}}_{r}$. Based on this, we let

$$\begin{aligned} L_{r} &= \sum_{i=1}^{C} \|\dot{\boldsymbol{r}}_{i} - \dot{\boldsymbol{r}}_{r}(t_{k+i})\|_{\boldsymbol{Q}_{r}}^{2} \\ &= \sum_{i=1}^{C} \left\|\boldsymbol{J}(\bar{\boldsymbol{q}}_{k+i})\,\boldsymbol{\nu}_{i} - \left(\dot{\boldsymbol{r}}_{d}(t_{k+i}) + \boldsymbol{K}\left(\boldsymbol{r}_{d}(t_{k+i}) - \boldsymbol{h}(\bar{\boldsymbol{q}}_{k+i})\right)\right)\right\|_{\boldsymbol{Q}_{r}}^{2}. \end{aligned} \tag{4.16}$$

Note that:

[4] In fact, one may easily verify that if $\dot{\boldsymbol{r}}$ converges to $\dot{\boldsymbol{r}}_{r}$ as given by (4.15), then $\boldsymbol{r}$ will converge to the desired trajectory $\boldsymbol{r}_{d}(t)$.

- the nonlinearity of $\boldsymbol{J}(\cdot)$ and $\boldsymbol{h}(\cdot)$ is dealt with by evaluating these terms over the auxiliary trajectory $\bar{\boldsymbol{q}}_{k+i}$;
- thanks to the partial feedback linearization and to the linearized of the pitch dynamics (4.7), all components of the velocity vector $\boldsymbol{\nu}_i$ depend linearly on the transformed input samples (i.e., the decision variables at t_k).

To regularize the optimization problem and to decrease the control effort, the cost function includes a second term

$$L_u = \sum_{i=0}^{C-1} \|\boldsymbol{u}_i\|^2_{\boldsymbol{Q}_u}. \tag{4.17}$$

Finally, a third term introduces preferred positions and velocities for the arm joint coordinates

$$L_a = \sum_{i=0}^{C-1} \|\boldsymbol{q}_{\{r,l\},i}\|^2_{\boldsymbol{Q}_p} + \|\dot{\boldsymbol{q}}_{\{r,l\},i}\|^2_{\boldsymbol{Q}_v}. \tag{4.18}$$

This last term will typically have a small weight as it is only introduced to resolve arm redundancy.

MPC formulation

The IS-MPC algorithm solves at each iteration the following OCP, obtained combining the cost function (4.16–4.18), the prediction model (4.8), the input constraints (4.14), and the stability constraint (4.12):

$$\begin{cases} \underset{\boldsymbol{u}}{\text{minimize}} & L_r + L_u + L_a \\ \text{subject to} & \boldsymbol{\xi}_0 = \hat{\boldsymbol{\xi}}_k \\ & \boldsymbol{\xi}_{i+1} = \boldsymbol{A}_{\boldsymbol{\xi},k+i}\boldsymbol{\xi}_i + \boldsymbol{B}_{\boldsymbol{\xi},k+i}\boldsymbol{u}_i + \boldsymbol{f}_{\boldsymbol{\xi},k+i} & \forall i \in \mathbb{I}_0^{C-1} \\ & \bar{\boldsymbol{\Psi}}^{-1}_{k+i}\bar{\boldsymbol{\alpha}}_{k+i} - \boldsymbol{\tau}_M \leq \bar{\boldsymbol{\Psi}}^{-1}_{k+i}\boldsymbol{u}_i \leq \bar{\boldsymbol{\Psi}}^{-1}_{k+i}\bar{\boldsymbol{\alpha}}_{k+i} + \boldsymbol{\tau}_M & \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{T}^*_{\mathrm{u}}\boldsymbol{\eta}_C = -\dfrac{f^*_{\mathrm{u}}}{\lambda^*_{\mathrm{u}}} \end{cases}$$

Being the cost function quadratic and the constraints linear, the MPC can easily be transcribed into a QP. To extend this formulation, one can easily add joint constraints by rewriting them as linear constraints on the decision variables by means of the prediction model (4.8).

As usual with MPC, only the first input of the solution is applied to the robot. To this end, the first sample $\boldsymbol{u}^*_0$ of the optimal input is used to compute the joint torque command

$$\boldsymbol{\tau}_k = \boldsymbol{\Psi}^{-1}(\boldsymbol{x}_k)(\boldsymbol{u}^*_0 - \boldsymbol{\alpha}(\boldsymbol{x}_k)). \tag{4.20}$$

The remaining input samples in the solution are used to predict the auxiliary trajectory for the next iteration.

4.5 Results

The proposed method was validated by MATLAB simulations on the ALTER-EGO WIP robot shown in Fig. 4.1. For each arm, we have considered only two degrees of freedom, namely the shoulder pitch angle and the elbow angle. The robot has an overall mass $m = 21.32$ kg, with the arm links contributing respectively for 1.8 kg and 1.0 kg. The wheel radius is $R = 0.13$ m and the semi-distance between the wheels is $a = 0.248$ m. The torque limits are set to ± 10 Nm for the wheels, ± 6 Nm for the shoulders and ± 1.1 Nm for the elbows.

The sampling time is chosen as $\delta_t = 0.02$ s while the MPC control horizon is $T_c = 0.5$ s. Using a dense formulation, each QP contains 150 decision variables and is solved using the `quadprog` function in MATLAB. The entire control loop runs in real-time (i.e., each iteration requires less than 0.02 s) on a standard laptop computer, so one can confidently expect an optimized C++ implementation to provide full real-time performance on an experimental platform at a high control frequency.

The accompanying video[5] contains clips of all the simulations.

4.5.1 Navigation task

In the first simulation, the proposed method is applied to a navigation task. The desired ground trajectory is a sine wave in x, y, with an amplitude of 0.2 m and a a wavelenght of 1 m. The desired velocity along the x direction is 0.3 m/s. As explained in Sect. 4.3.3, for navigation tasks it is necessary to use as task function the position of a point displaced from the base; in particular, we have set $d = 0.2$ m. The weight matrices used in the QP cost function are $\boldsymbol{Q}_r = 50\,\boldsymbol{I}$, $\boldsymbol{Q}_u = \mathrm{diag}\{0.05, 0.05, 1, 1, 1, 1\}$ and $\boldsymbol{Q}_p = \boldsymbol{Q}_v = \boldsymbol{I}$, while the gain in Eq. (4.15) is set to $\boldsymbol{K} = \boldsymbol{I}$.

Figure 4.4 is a stroboscopic view of the generated motion, while Fig. 4.5 shows the norm of the tracking error e_{xy} and the input torques. At the start of the simulation, there is an initial error due to the fact that the robot state is not matched with the desired trajectory. This error increases at first, a behavior which represents the typical undershoot characterizing the response of non-minimum phase systems. In fact, in order to achieve a stable forward acceleration of the base, the robot has to first move backwards so as to tilt the pitch angle in the forward direction. After this transient, the desired trajectory is followed with good accuracy.

The torque plots show that the arm actuators are actively involved in the constrained minimization of the cost function, confirming the whole-body nature of our approach.

Figures 4.6 and 4.7 report additional results for the case of a circular ground trajectory. Note how, in this case, the system is able to achieve zero tracking error after an initial transient. An animation of the generated motion is included in the accompanying video, where it is also shown that removing the stability constraint leads to the robot immediately losing balance. The video also contains simulation results for the additional scenario of reaching a navigation set-point.

[5] https://youtu.be/PyyoZNOeklE

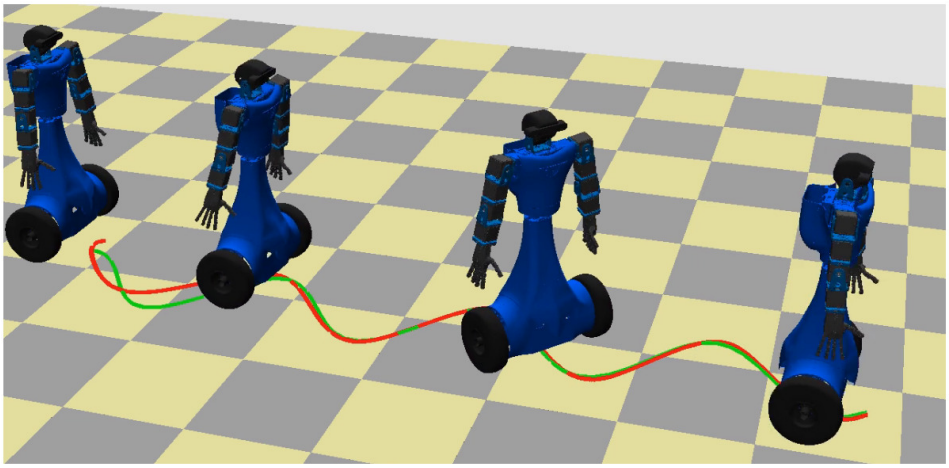

Figure 4.4. Navigation task with ALTER-EGO following a sinusoidal trajectory: Stroboscopic view of the generated motion with actual (red) and desired (green) task trajectories.

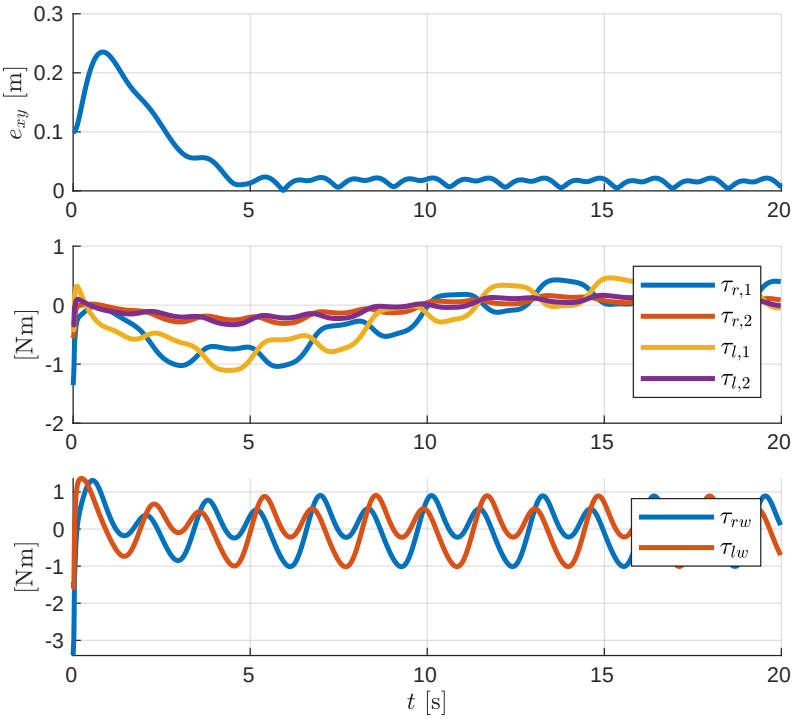

Figure 4.5. Navigation task with ALTER-EGO following a sinusoidal trajectory: tracking error norm (top), arm torques (center), wheel torques (bottom). Torque limits are not shown since all actuators are far from saturation.

4.5.2 Loco-manipulation task

The second simulation deals with a loco-manipulation task. In particular, we assign as desired trajectory for the right hand a helix with a radius $r = 0.2$ m, placed at an average height of 0.9 m from the ground.

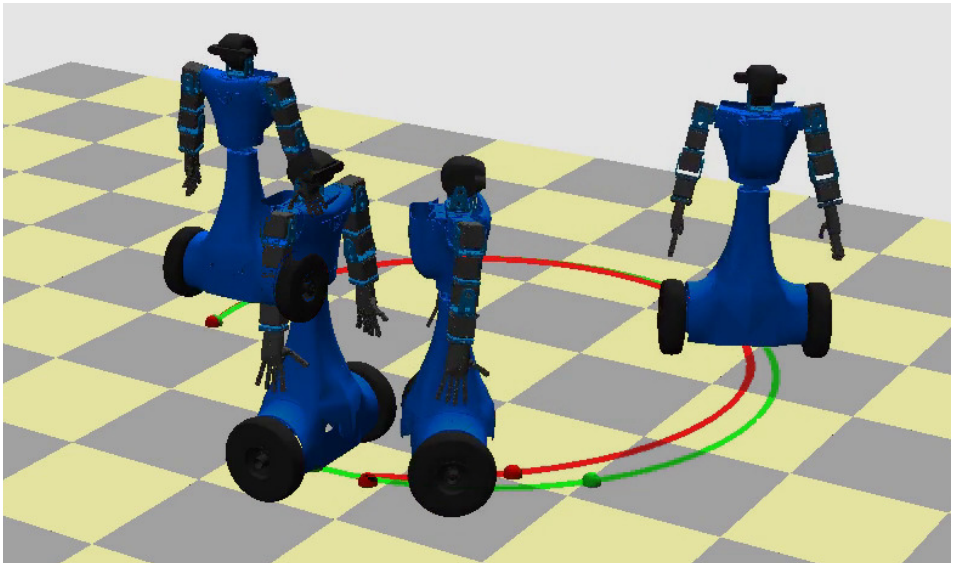

Figure 4.6. Navigation task with ALTER-EGO following a circular trajectory: Stroboscopic view of the generated motion with actual (red) and desired (green) task trajectories.

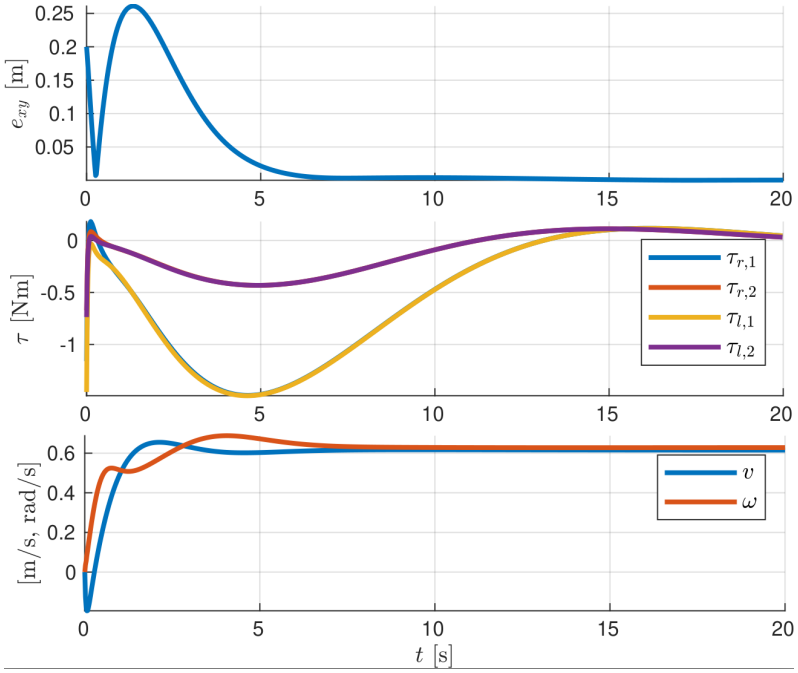

Figure 4.7. Navigation task with ALTER-EGO following a sinusoidal circular: tracking error norm (top), arm torques (center), wheel velocities (bottom). Torque limits are not shown since all actuators are far from saturation.

The weight matrices for the QP cost function are $\boldsymbol{Q}_p = \text{diag}\{10^{-4}, 0.8, 1, 1\}$, $\boldsymbol{Q}_v = \text{diag}\{0.01, 0.1, 1, 1\}$, $\boldsymbol{Q}_u = \text{diag}\{0.1, 0.025, 0.05, 0.01, 1, 1\}$, and $\boldsymbol{Q}_r = 20\,\boldsymbol{I}$, while the gain in eq. (4.15) is again set to $\boldsymbol{K} = \boldsymbol{I}$.

A stroboscopic view of the generated motion is shown in Fig. 4.8, with the

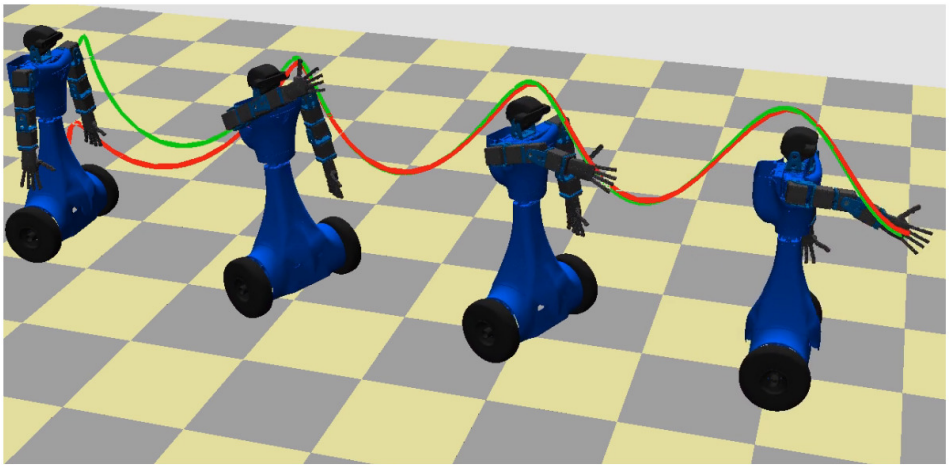

Figure 4.8. Loco-manipulation task with ALTER-EGO: Stroboscopic view of the generated motion with actual (red) and desired (green) task trajectories.

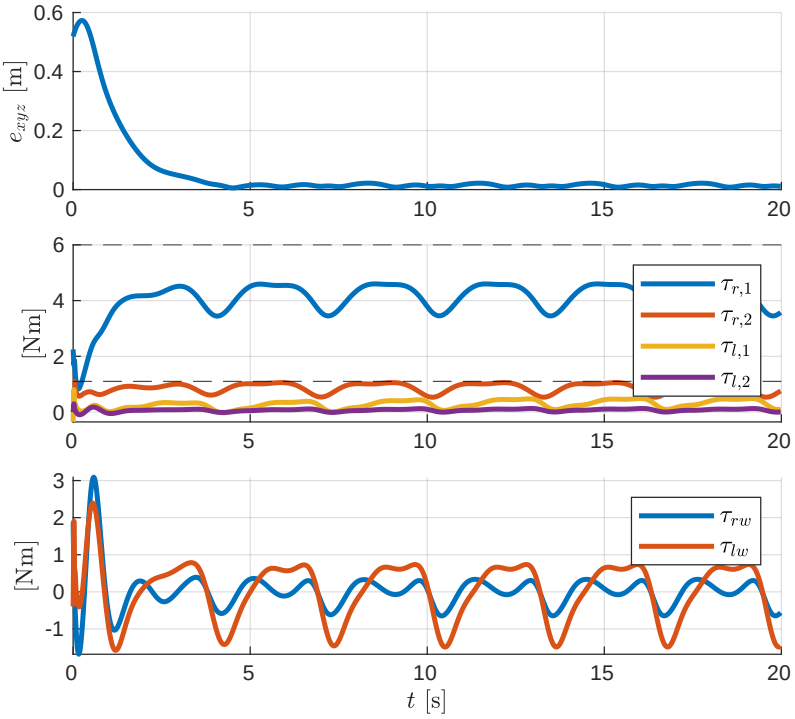

Figure 4.9. Loco-manipulation task with ALTER-EGO: tracking error norm (top), arm torques (center), wheel torques (bottom). Arm torque limits are shown by dashed lines, while wheel torque limits are outside the plot range.

tracking error norm e_{xyz} and the input torques reported in Fig. 4.9. Again, the tracking error exhibits an initial transient after which is reduced essentially to zero, with some negligible fluctuations. In the right arm, the shoulder torque is doing most of work, by hovering around 4 Nm to compensate gravity, while the elbow torque saturates to the upper limit, fully confirming the validity of the input constraint transformation procedure discussed in Sect. 4.4.2. Left arm torques are much

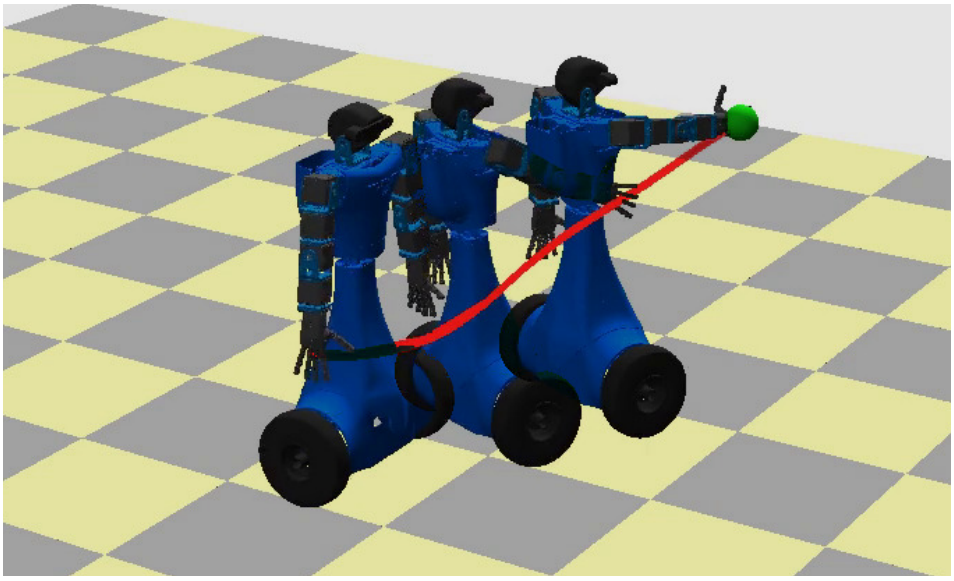

Figure 4.10. Loco-manipulation task with ALTER-EGO: Stroboscopic view of the generated motion with actual (red) task trajectory and desired end-effector position (green ball).

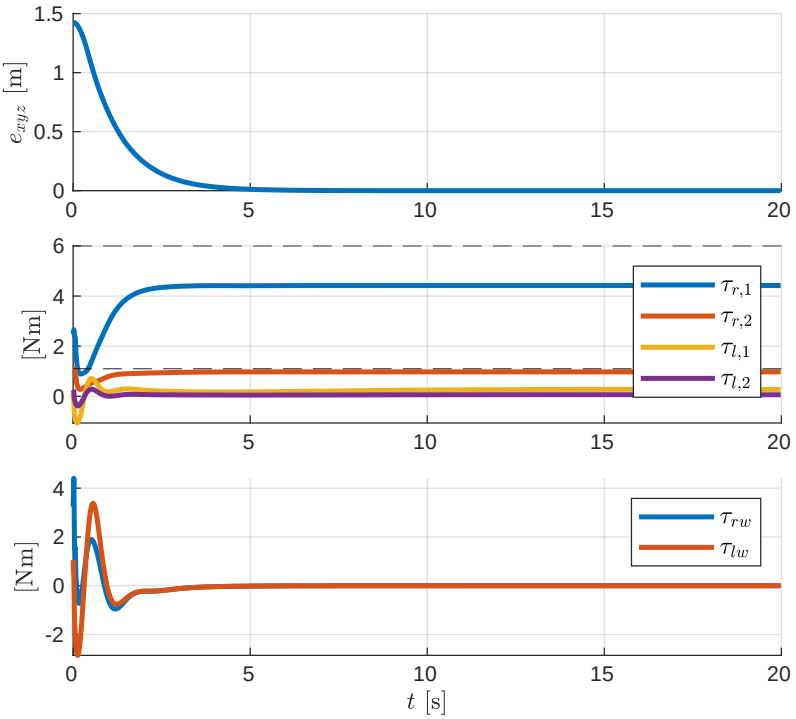

Figure 4.11. Loco-manipulation task with ALTER-EGO: tracking error norm (top), arm torques (center), wheel torques (bottom). Arm torque limits are shown by dashed lines, while wheel torque limits are outside the plot range.

smaller, but still actively involved in motion generation.

We also report results for a setpoint regulation task, in which the right hand

has to reach a fixed position target (see Fig. 4.10 and 4.11). In this case, the end-effector converges to the target in around 5 s, with the arm motion starting from the beginning of the movement thanks to the whole-body nature of the controller. Also, the right arm elbow torque reaches its upper limit as the arm is fully stretched. Animations of these motions are also included in the accompanying video, along with one additional loco-manipulation scenario of tracking a mid-air circular trajectory with the right hand.

Chapter 5

Anti-Jackknifing Control of Tractor-Trailer Vehicles

Trailers are added to ground vehicles to increase their payload capacity without sacrificing too much of their maneuverability. Still, it is quite obvious that driving and controlling a tractor-trailer system is more challenging than a single-body vehicle [85]. One phenomenon that may arise during the motion of vehicles with trailers is the so-called *jackknifing*. This term denotes a situation in which the hitch angle between the tractor and trailer grows until the vehicle folds on itself, losing controllability and possibly causing collisions (see Fig. 5.1).

Jackknifing is a serious issue during backup maneuvers, as any truck driver knows by experience. It may also arise in forward motion, for example when a truck brakes or turns abruptly. The nature of the phenomenon is however different in the two cases: jackknifing in backward motion is essentially a kinematic issue which occurs at any speed, while in forward motion it is a high-speed inertial effect related to slippage.

5.1 Related works

In the literature, the control problem for tractor-trailer vehicles has been addressed following two categories of approaches: in the first, the reference motion is a path or trajectory in Cartesian space, while in the second it is given in configuration space. From a control viewpoint, the two approaches correspond respectively to *output* and *state* tracking.

Works belonging to the first category include [86], where a feedback control scheme is proposed to drive a *general* (i.e., nonzero-hooked) one-trailer system along backward trajectories by transforming the control inputs of a virtual vehicle moving forward along the same trajectory; [87], which introduces a two-level trajectory tracking controller for a zero-hooked one-trailer system; and [88], where the trajectory tracking problem is considered for a general n-trailer vehicle whose last trailer must track a linear/circular trajectory. More recent work includes [89], which addresses the problem by defining the last trailer as a virtual tractor, and [90], which proposes a curvature-based method for both forward and backward path following in the presence of sideslip.

Figure 5.1. Tractor-Trailer vehicle having crashed after jackknifing.

Coming to the second category, a first subgroup consists of works where the reference state path/trajectory corresponds to specific (linear/circular) Cartesian motions. This includes [91], where a zero-hooked one-trailer system is controlled via input/state linearization and time scale transformation; [92, 93], which tackle the general one-trailer system using linear and Lyapunov-based design, respectively; and [94, 95], that consider path tracking for the general two trailer-system.

The second subgroup of state tracking approaches deals with generic reference paths/trajectories in configuration space. Among these, we mention [96], where a low-level hitch angle controller is designed to simplify the general one-trailer system model and the associated control problem, and [97], that uses a Linear Quadratic (LQ) controller for driving a general two-trailer system along a backward path generated by composing motion primitives. This subgroup also includes works using Model Predictive Control, such as [98, 99], where only forward motions are considered, and [100], which focuses on the case of the general two-trailer system.

5.2 Contribution

The control method presented in this chapter belongs to the first category, i.e., tracking in Cartesian (output) space. We favor this approach because it is closer to the typical problems arising in applications. Indeed, using a method of the second category for Cartesian motion control would require a preliminary conversion of the reference motion to a stable path/trajectory in configuration (state) space, and this is a non-trivial problem that can only be solved in closed form in very special cases, such as linear/circular motions.

In particular, we focus on the kinematic jackknifing that occurs along backward motions. The starting observation is that such phenomenon is a manifestation of the instability of the residual internal dynamics associated to output trajectory tracking; or, in other words, of the non-minimum phase nature of the system in backward motion. Therefore, we propose to build a feedback control law as the combination of two actions: a Cartesian trajectory tracking term, computed using input-output linearization, and a corrective term, aimed at avoiding the internal state divergence

and generated via MPC. For the latter, we apply the IS-MPC method described in Chapter 3.

Since the stability condition in IS-MPC applies to linear systems, we compute the linear approximation of the tractor-trailer system around a suitable state trajectory and use it as a prediction model for IS-MPC. The latter includes an explicit stability constraint whose role is to counteract the divergence of internal dynamics. The resulting method has been verified in simulation and experimentally validated on a purposely built prototype.

With respect to the above mentioned literature, the proposed control method has the following beneficial features:

- since we directly address the tracking problem in Cartesian space, the reference trajectory can be completely generic and no preliminary conversion is required to a stable trajectory in configuration space;
- the method applies with any number of trailers, with any combination of zero- and nonzero-hooking;
- in contrast to most previous works, our method requires almost no tuning, as the only control parameters are the gains of the Cartesian tracking controller;
- use of MPC allows enforcing state and input constraints to take into account vehicle kinematic limitations (joint limits), avoid workspace obstacles during the motion, or comply with the presence of actuator saturations;
- the jackknifing problem is directly tackled by avoiding the onset of the divergence of the zero dynamics via an explicit stability constraint.

With reference to the last point, it should be emphasized that methods belonging to the second category (state tracking) still require the introduction of artificial thresholds on the tracking error to avoid jackknifing, e.g., see [100]. Identifying these thresholds requires an initial campaign of simulations or experiments, and in any case their use may prove to be exceedingly conservative, in the sense that it prevents the vehicle from executing motions that are actually feasible.

Another interesting aspect of the proposed approach is that the resulting controller works for both forward and backward trajectory tracking, thereby eliminating the necessity of using specialized controllers for the two cases. In fact, in forward motion the stability constraint automatically disappears (the dimension of the unstable zero dynamics goes to zero), and the QP problem at the core of our MPC (see Sect. 5.4.4) will produce a corrective term whose only role is to guarantee kinematic and actuation feasibility.

The chapter is organized as follows. In Section 5.3 we introduce the considered control problem, describing the vehicle kinematic model and offering an interpretation of the jackknife phenomenon associated to tracking control. In Section 5.4, we describe in general terms the proposed control approach, and then in detail the generation of the auxiliary trajectory, the approximate linearization procedure and the IS-MPC algorithm for computing the corrective control term. Simulations and experiments are presented in Sect. 5.5.1 and 5.5.3, respectively. The extension to a two-trailer vehicles is outlined in Sect. 5.6.

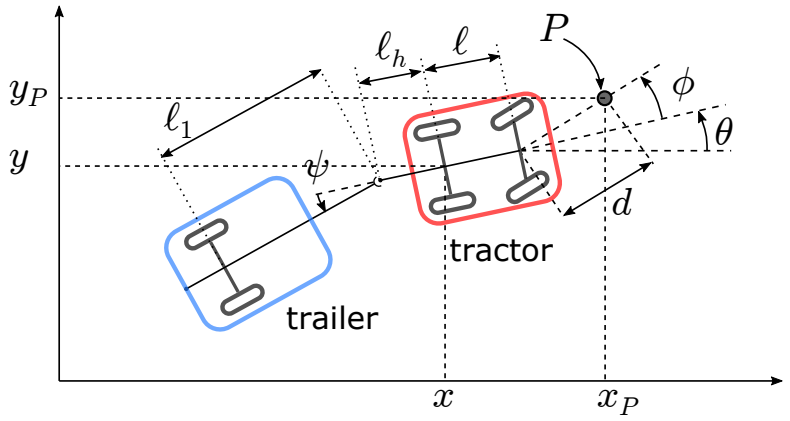

Figure 5.2. The considered tractor-trailer vehicle. Note the nonzero hooking.

5.3 The control problem

In this section we introduce the considered vehicle, state the control problem and provide an interpretation of the jackknife phenomenon in this context.

5.3.1 Modeling

Consider the vehicle shown in Fig. 5.2 consisting of a car-like tractor towing a single[1] trailer. Denote by x, y the coordinates of the tractor rear axle midpoint, and by θ, ϕ and ψ respectively the tractor heading, the steering angle and the hitch angle (i.e., the relative orientation of the trailer with respect to the tractor). We denote with $\boldsymbol{q}_r = (x, y, \theta, \psi, \phi)$ the configuration vector of the system. Also, let ℓ and ℓ_1 be the length of the tractor and the trailer, and ℓ_h the distance between the tractor rear axle midpoint and the hitch joint axis. We will consider the general case in which $\ell_h \neq 0$ (*nonzero hooking*).

If no wheel slip occurs (an assumption that is consistent with the low speed typically associated to backup maneuvers), the kinematic model [81] of the vehicle is derived by imposing constraints on the velocity of the wheels along the wheel axis directions. These can be written as three independent Pfaffian constraints in matrix form as:

$$\boldsymbol{A}^T(\boldsymbol{q}_r)\dot{\boldsymbol{q}}_r = \begin{pmatrix} \sin\theta & -\cos\theta & 0 & 0 & 0 \\ \sin(\theta+\phi) & -\cos(\theta+\phi) & -\ell\cos\phi & 0 & 0 \\ \sin(\theta+\psi) & -\cos(\theta+\psi) & \ell_h\cos\psi + \ell_1 & \ell_1 & 0 \end{pmatrix} \dot{\boldsymbol{q}}_r = \boldsymbol{0}. \quad (5.1)$$

The generalized coordinates are constrained to belong to the 2-dimensional null space of matrix $\boldsymbol{A}^T(\boldsymbol{q}_r)$. The equations of motion of the system can then be obtained by finding a basis $(\boldsymbol{g}_1(\boldsymbol{q}_r), \boldsymbol{g}_2(\boldsymbol{q}_r))$ of $\mathcal{N}\left(\boldsymbol{A}^T(\boldsymbol{q}_r)\right)$, resulting in a kinematic model of the form

$$\dot{\boldsymbol{q}}_r = \boldsymbol{g}_1(\boldsymbol{q}_r)v + \boldsymbol{g}_2(\boldsymbol{q}_r)\omega,$$

[1] The control design to be presented does not exploit in any way the fact that a single trailer is present. Therefore, our method is applicable to vehicles with more than one trailer, including the so-called *general n-trailer system*. See Section 5.6 for the application to the two-trailer system.

where v and ω are respectively the driving and steering velocities, taken as control inputs and collected in the vector $\boldsymbol{\nu} = (v, \omega)$. Then, it is straightforward to verify that the following model satisfies the constraints (5.1):

$$\begin{aligned}
\dot{x} &= v\cos\theta \\
\dot{y} &= v\sin\theta \\
\dot{\theta} &= \frac{v\tan\phi}{\ell} \\
\dot{\psi} &= -\frac{v\tan\phi}{\ell}\left(1+\frac{\ell_h}{\ell_1}\cos\psi\right) - \frac{v\sin\psi}{\ell_1} \\
\dot{\phi} &= \omega.
\end{aligned} \tag{5.2}$$

5.3.2 Internal instability under tracking control

Assume that a Cartesian reference trajectory $(x_{\mathrm{ref}}(t), y_{\mathrm{ref}}(t))$ is assigned to be tracked by the vehicle. From a control viewpoint, this is simply an *output* trajectory — an associated state trajectory is not given.

The most direct way to design a tracking controller is to use input-output linearization via feedback. Ideally, one would like to track the reference trajectory with the vehicle representative point (x, y). However, this cannot be achieved by static feedback because the decoupling matrix turns out to be singular. A possible workaround is to choose as output a different point P with coordinates (x_P, y_P), as shown in Fig. 5.2. One easily finds

$$\begin{pmatrix} \dot{x}_P \\ \dot{y}_P \end{pmatrix} = \boldsymbol{D}(\theta,\phi)\begin{pmatrix} v \\ \omega \end{pmatrix}$$

with

$$\boldsymbol{D}(\theta,\phi) = \begin{pmatrix} c_\theta - \frac{t_\phi}{\ell}(\ell s_\theta + d\, s_{\theta+\phi}) & -d\, s_{\theta+\phi} \\ s_\theta + \frac{t_\phi}{\ell}(\ell c_\theta + d\, c_{\theta+\phi}) & d\, c_{\theta+\phi} \end{pmatrix}$$

setting for compactness $s_\alpha = \sin\alpha$, $c_\alpha = \cos\alpha$, $t_\alpha = \tan\alpha$ from now on. Since $\det \boldsymbol{D} = d/c_\phi$, matrix $\boldsymbol{D}$ is invertible if d is nonzero. Under this assumption, one can achieve input-output linearization by using the feedback transformation

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \boldsymbol{D}^{-1}(\theta,\phi)\boldsymbol{u} \tag{5.3}$$

where $\boldsymbol{u} = (u_1, u_2)$ is the new control vector. The input-output linearized dynamics is expressed in normal form (cfr. Sect. 3.1) as

$$\begin{aligned}
\dot{x}_P &= u_1 \\
\dot{y}_P &= u_2 \\
\dot{\theta} &= \frac{s_\phi}{\ell}(c_{\theta+\phi}\,u_1 + s_{\theta+\phi}\,u_2) \\
\dot{\psi} &= -\frac{1}{\ell\,\ell_1}(\ell_h s_\phi\, c_\psi + \ell_1 s_\phi + \ell c_\phi\, s_\psi)(c_{\theta+\phi}\,u_1 + s_{\theta+\phi}\,u_2) \\
\dot{\phi} &= -\left(\frac{c_{\theta+\phi}\,s_\phi}{\ell} + \frac{s_{\theta+\phi}}{d}\right)u_1 - \left(\frac{s_{\theta+\phi}\,s_\phi}{\ell} - \frac{c_{\theta+\phi}}{d}\right)u_2.
\end{aligned} \tag{5.4}$$

By setting

$$\boldsymbol{u} = \boldsymbol{u}_{\text{track}} = \begin{pmatrix} \dot{x}_{\text{ref}} + k_x(x_{\text{ref}} - x_P) \\ \dot{y}_{\text{ref}} + k_y(y_{\text{ref}} - y_P) \end{pmatrix} \tag{5.5}$$

with $k_x, k_y > 0$, the tracking error will converge exponentially to zero for any initial condition.

However, the evolution of variables θ, ψ and ϕ is not controlled in this scheme. Additional insight can be gained by looking at the *zero dynamics* of (5.4–5.5), i.e., the closed-loop dynamics of θ, ψ and ϕ along the reference trajectory:

$$\begin{aligned} \dot{\theta} &= \frac{s_\phi}{\ell}\left(c_{\theta+\phi}\,\dot{x}_{\text{ref}} + s_{\theta+\phi}\,\dot{y}_{\text{ref}}\right) \\ \dot{\psi} &= -\frac{\ell_h s_\phi\, c_\psi + \ell_1 s_\phi + \ell c_\phi\, s_\psi}{\ell \ell_1}\left(c_{\theta+\phi}\,\dot{x}_{\text{ref}} + s_{\theta+\phi}\,\dot{y}_{\text{ref}}\right) \\ \dot{\phi} &= -\left(\frac{c_{\theta+\phi}\, s_\phi}{\ell} + \frac{s_{\theta+\phi}}{d}\right)\dot{x}_{\text{ref}} - \left(\frac{s_{\theta+\phi}\, s_\phi}{\ell} - \frac{c_{\theta+\phi}}{d}\right)\dot{y}_{\text{ref}}. \end{aligned} \tag{5.6}$$

Consider for example the case in which the vehicle must track the linear trajectory

$$x_{\text{ref}} = v_{\text{ref}}\, t, \qquad y_{\text{ref}} = 0, \tag{5.7}$$

with $v_{\text{ref}} < 0$, starting from $x_P = y_P = 0$ [m], and $\theta = \psi = \phi = 0$ [rad]; i.e., point P is on the desired trajectory but the vehicle points in the opposite direction (*tracking in backward motion*). The behavior of the vehicle can be predicted by setting $\dot{x}_{\text{ref}} = v_{\text{ref}}$ and $\dot{y}_{\text{ref}} = 0$ in the zero dynamics (5.6), and deriving[2] its linear approximation around the origin, whose eigenvalues are positive and, in particular, easily found to be $\{-v_{\text{ref}}/\ell, -v_{\text{ref}}/\ell_1, -v_{\text{ref}}/d\}$. From this, we can deduce several interesting facts:

- The presence of three positive eigenvalues indicates that the origin is unstable for the zero dynamics — one also says that system (5.4–5.5) is *non-minimum phase* in this case. This entails that the evolution of the internal variables is not bounded when tracking in backward motion. In fact, a simple simulation reveals that in practice θ, ψ and ϕ will diverge from zero as the vehicle moves. This will severely affect the vehicle's maneuvrability, because the hitch angle will engage its mechanical joint limit or a self-collision will occur between the tractor and the trailer.

- Not surprisingly, the zero dynamics instability is emphasized at higher speeds (large $|v_{\text{ref}}|$), for shorter vehicles (small ℓ, ℓ_1) or when P is closer to the wheels (small d).

- If $v_{\text{ref}} > 0$ in eq. (5.7), the reference trajectory moves in the direction of the positive x axis, with the vehicle initially pointing in the same direction (*tracking in forward motion*). Based on the eigenvalue analysis, the origin of the zero dynamics is now asymptotically stable, so that θ, ψ and ϕ are bounded — in fact, they converge to zero.

[2]For this computation, one may use the general expression of the linear approximation of system (5.4) given in Sect. 5.4.3, setting $\tilde{\boldsymbol{q}} = (v_{\text{ref}}\, t, 0, 0, 0, 0)$, $\boldsymbol{u}_{\text{corr}} = \boldsymbol{0}$, and focusing on the last three equations.

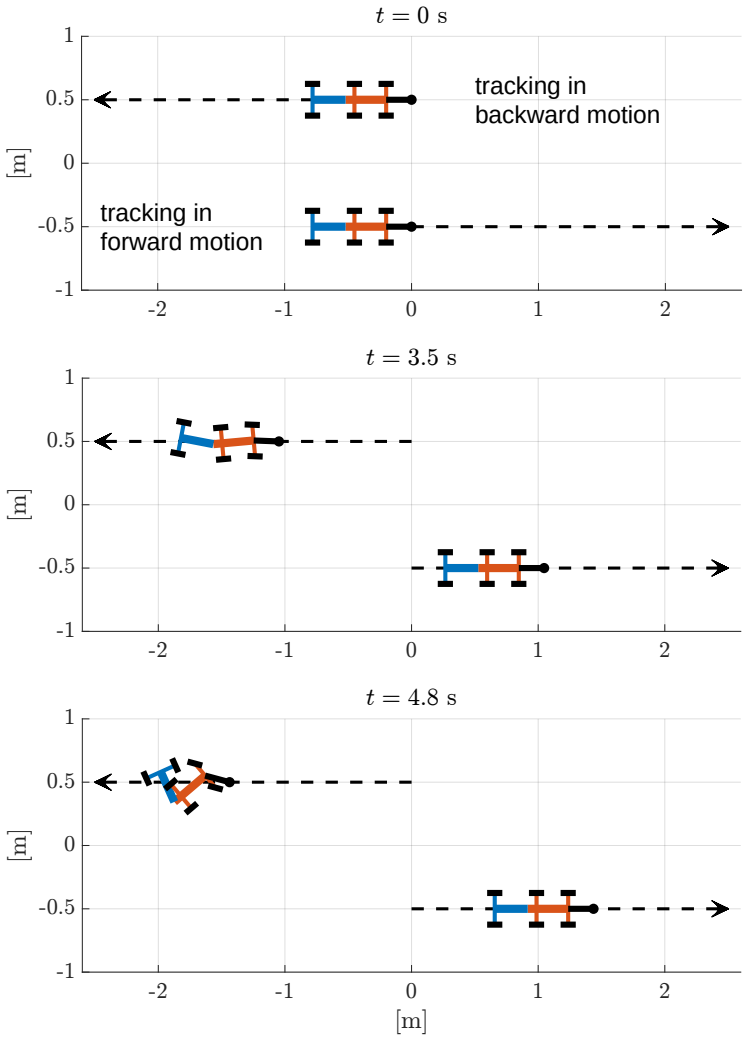

Figure 5.3. For system (5.4–5.5), tracking in backward motion is unstable and leads to jackknifing, whereas tracking in forward motion is internally stable (tractor in red, trailer in blue).

The simulations in Figure 5.3 summarize the above discussion (see the accompanying video[3] for an animation).

In practice, the divergence of the angular variables, and in particular of ψ, corresponds to the occurrence of the jackknife phenomenon for the vehicle. Its control interpretation is therefore straightforward: jackknifing is a manifestation of the instability of the zero dynamics of system (5.4–5.5) when tracking in backward motion.

5.4 The proposed approach

In this section we discuss the proposed method for avoiding the jackknife phenomenon in backward motion. We will first provide a general overview of the solu-

[3] https://youtu.be/ImG1ZV1EYBs

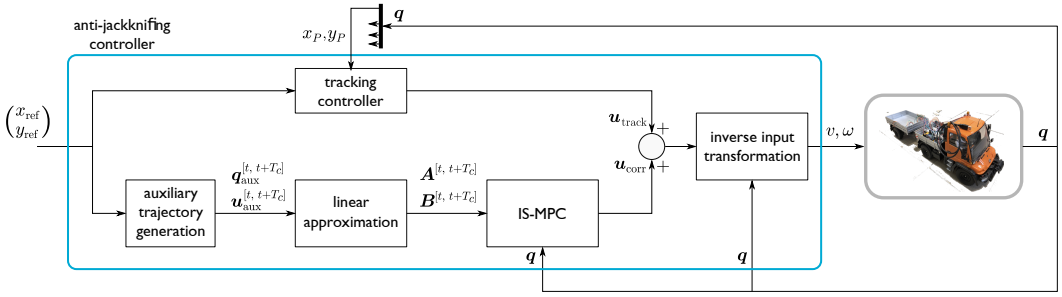

Figure 5.4. A block scheme of the proposed anti-jackknifing control approach. Note that at each instant t we must compute $\boldsymbol{q}_{\text{aux}}^{[t,t+T_c]}$, $\boldsymbol{u}_{\text{aux}}^{[t,t+T_c]}$, the portion of the auxiliary trajectory and associated input that are contained in the IS-MPC control horizon, and $\boldsymbol{A}^{[t,t+T_c]}$, $\boldsymbol{B}^{[t,t+T_c]}$, the time-varying matrices of the approximate linearization in the same interval.

tion approach and then proceed to a detailed discussion of its main components.

5.4.1 Overview

Let us start with the tractor-trailer system in the input-output linearized form (5.4) thanks to the feedback transformation (5.3). The basic idea is to add to the pure tracking control

$$\boldsymbol{u}_{\text{track}} = \begin{pmatrix} \dot{x}_{\text{ref}} + k_x(x_{\text{ref}} - x_P) \\ \dot{y}_{\text{ref}} + k_y(y_{\text{ref}} - y_P) \end{pmatrix} \tag{5.8}$$

a corrective action aimed at avoiding the onset of instability:

$$\boldsymbol{u} = \boldsymbol{u}_{\text{track}} + \boldsymbol{u}_{\text{corr}}. \tag{5.9}$$

In our approach, $\boldsymbol{u}_{\text{corr}}$ is generated using the IS-MPC method. Since model (5.4) is nonlinear, we compute in real time its linear approximation around an *auxiliary* trajectory $\boldsymbol{q}_{\text{aux}}$, whose construction is illustrated in Sect. 5.4.2, and we feed it to the IS-MPC block. Since the latter operates over a control horizon T_c, both the auxiliary trajectory and the linear approximation — which is obviously time-varying — must be made available over the same time interval. Figure 5.4 shows a block scheme of the proposed approach.

Note that the idea of adding a corrective term to avoid jackknifing may also be of interest in a *mixed-initiative* control context, in which the human driver would essentially provide the basic tracking control action and the proposed algorithm could be used to design an anti-jackknifing ADAS (Advanced Driving Assistance System, e.g., see [101]).

5.4.2 Generation of the auxiliary trajectory

The reference output trajectory does not entail a trajectory for the state variables; this is related to the fact that the Cartesian coordinates x, y (or x_P, y_P) do not represent a *flat* output for the tractor-trailer system. Therefore, we must identify an auxiliary state trajectory $\boldsymbol{q}_{\text{aux}}(t)$ around which to compute an accurate linear

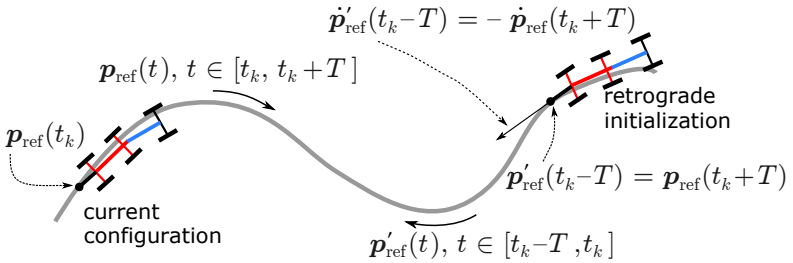

Figure 5.5. Definition of the retrograde initialization (tractor in red, trailer in blue). For compactness we have let $\boldsymbol{p}_{\rm ref} = (x_{\rm ref}, y_{\rm ref})$. In this figure, the current configuration of the vehicle is such that point P matches its reference value; however, this will not be true in general.

approximation of the nonlinear system (5.4). To this end, we generate a *stable* state trajectory (i.e., a trajectory along which θ, ψ and ϕ do not diverge) compatible with the reference output trajectory using the following procedure.

1. Call t_k the current time instant at which the computation is performed. Given a $T > 0$, let $t_k - T$ be the (past) initial time instant, and initialize the auxiliary trajectory at

$$\boldsymbol{q}_{\rm aux}(t_k - T) = \begin{pmatrix} x_{\rm ref}(t_k + T) \\ y_{\rm ref}(t_k + T) \\ \text{ATAN2}(-\dot{y}_{\rm ref}(t_k + T), -\dot{x}_{\rm ref}(t_k + T)) \\ 0 \\ 0 \end{pmatrix}$$

   i.e., point P matched to the (future) value of the reference trajectory at t_k+T, tractor oriented as the backward tangent to the reference trajectory, trailer aligned with the tractor, and zero steering angle (see Fig. 5.5).

2. Generate the state trajectory $\boldsymbol{q}_{\rm aux}(t)$ for $t \in [t_k-T,\, t_k]$ by integrating model (5.4) from the retrograde initialization $\boldsymbol{q}_{\rm aux}(t_k-T)$ up to t_k under the pure tracking (no correction) control law (5.5). In doing this, the reference trajectory must be *reverted* by replacing $x_{\rm ref}(t)$ and $y_{\rm ref}(t)$ with $x'_{\rm ref}(t) = x_{\rm ref}(2t_k - t)$ and $y'_{\rm ref}(t) = y_{\rm ref}(2t_k - t)$, respectively. The resulting control law will define the input $\boldsymbol{u}_{\rm aux} = \boldsymbol{u}_{\rm track}(\boldsymbol{q}_{\rm aux})$ associated to the auxiliary trajectory[4]. Since this tracking is in forward motion, trajectory $\boldsymbol{q}_{\rm aux}(t)$ for $t \in [t_k - T, t_k]$ is stable.

3. The auxiliary trajectory $\boldsymbol{q}_{\rm aux}(t)$ for $t \in [t_k, t_k + T]$ is finally obtained by reverting the stable trajectory $\boldsymbol{q}_{\rm aux}(t)$ for $t \in [t_k - T, t_k]$. This is simply obtained by setting $\boldsymbol{q}_{\rm aux}(t) = \boldsymbol{q}_{\rm aux}(2t_k - t)$.

Figure 5.6 describes the above procedure in a nutshell.

Note that the specific choice of θ, ψ and ϕ in the retrograde initialization $\boldsymbol{q}_{\rm aux}(t_k - T)$ is not important as long as it leads to tracking the reference trajectory in *forward* motion; the above choice is an example. As for T, it should be

[4]Note that $\boldsymbol{u}_{\rm aux}$ will only consist of the feedforward component as the retrograde initialization is matched to the reference trajectory.

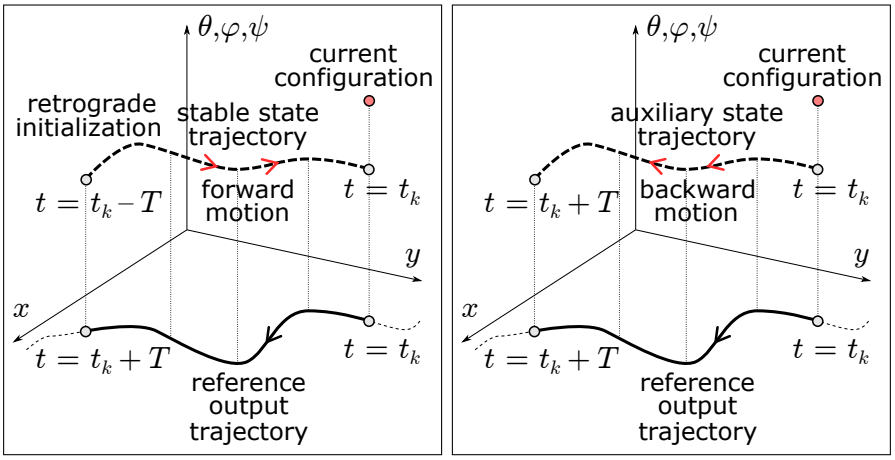

Figure 5.6. Starting from the retrograde initialization of Fig. 5.5, a stable state trajectory is generated by pure tracking of the reversed output trajectory in forward motion. Then, this stable trajectory is itself reversed to produce the auxiliary state trajectory.

sufficiently large for the transient to be practically over in t_k; moreover, it should be larger than the control horizon of the MPC (see Sect. 5.4.4).

5.4.3 Linearization around the auxiliary trajectory

At this point, it is possible to compute the linear approximation of model (5.4), subject to the control law (5.9), around the auxiliary trajectory $\boldsymbol{q}_{\text{aux}}$, for $t \in [t_k, t_k + T]$. Letting $\boldsymbol{\varepsilon} = \boldsymbol{q} - \boldsymbol{q}_{\text{aux}}$, one obtains a linear system which is time-varying due to the dependence on the auxiliary trajectory:

$$\dot{\boldsymbol{\varepsilon}} = \boldsymbol{A}(t)\boldsymbol{\varepsilon} + \boldsymbol{B}(t)\boldsymbol{u}_{\text{corr}}. \tag{5.10}$$

The *nonzero* elements A_{ij} of $\boldsymbol{A}(t)$ and B_{ij} of $\boldsymbol{B}(t)$ are given in the following expressions, where $\boldsymbol{A}(t)$, $\boldsymbol{B}(t)$ are deduced by setting $\tilde{\boldsymbol{q}}(t) = \boldsymbol{q}_{\text{aux}}(t)$ and $\dot{\tilde{x}}_P = \dot{x}_{\text{ref}}$, $\dot{\tilde{y}}_P(t) = \dot{y}_{\text{ref}}(t)$.

$$
\begin{aligned}
A_{11} &= -k_x \quad A_{22} = -k_y \quad A_{31} = -\frac{k_x}{\ell} s_{\tilde{\phi}} c_{\tilde{\theta}+\tilde{\phi}} \\
A_{32} &= -\frac{k_y}{\ell} s_{\tilde{\phi}} s_{\tilde{\theta}+\tilde{\phi}} \quad A_{33} = \frac{s_{\tilde{\phi}}}{\ell}\left(-s_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{x}}_P + c_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{y}}_P\right) \\
A_{35} &= \frac{1}{\ell}\left(c_{\tilde{\theta}+2\tilde{\phi}}\dot{\tilde{x}}_P + s_{\tilde{\theta}+2\tilde{\phi}}\dot{\tilde{y}}_P\right) \\
A_{41} &= \frac{k_x c_{\tilde{\theta}+\tilde{\phi}}}{\ell\ell_1}\left(\ell_h s_{\tilde{\phi}} c_{\tilde{\psi}} + \ell_1 s_{\tilde{\phi}} + \ell c_{\tilde{\phi}} s_{\tilde{\psi}}\right) \\
A_{42} &= \frac{k_y s_{\tilde{\theta}+\tilde{\phi}}}{\ell\ell_1}\left(\ell_h s_{\tilde{\phi}} c_{\tilde{\psi}} + \ell_1 s_{\tilde{\phi}} + \ell c_{\tilde{\phi}} s_{\tilde{\psi}}\right) \\
A_{43} &= \frac{1}{\ell\ell_1}\left(\ell_h s_{\tilde{\phi}} c_{\tilde{\psi}} + \ell_1 s_{\tilde{\phi}} + \ell c_{\tilde{\phi}} s_{\tilde{\psi}}\right)\left(s_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{x}}_P - c_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{y}}_P\right) \\
A_{44} &= -\frac{1}{\ell\ell_1}\left(-\ell_h s_{\tilde{\phi}} s_{\tilde{\psi}} + \ell c_{\tilde{\phi}} c_{\tilde{\psi}}\right)\left(c_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{x}}_P + s_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{y}}_P\right) \\
A_{45} &= -\frac{1}{\ell\ell_1}\Big(\ell_h c_{\tilde{\phi}} c_{\tilde{\psi}} + \ell_1 c_{\tilde{\phi}} - \ell s_{\tilde{\phi}} s_{\tilde{\psi}}\Big)\Big(c_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{x}}_P + s_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{y}}_P\Big) \\
&\quad + \frac{1}{\ell\ell_1}\Big(\ell_h s_{\tilde{\phi}} c_{\tilde{\psi}} + \ell_1 s_{\tilde{\phi}} + \ell c_{\tilde{\phi}} s_{\tilde{\psi}}\Big)\Big(s_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{x}}_P - c_{\tilde{\theta}+\tilde{\phi}}\dot{\tilde{y}}_P\Big) \\
A_{51} &= k_x\left(\frac{c_{\tilde{\theta}+\tilde{\phi}} s_{\tilde{\phi}}}{\ell} + \frac{s_{\tilde{\theta}+\tilde{\phi}}}{d}\right) \quad A_{52} = k_y\left(\frac{s_{\tilde{\theta}+\tilde{\phi}} s_{\tilde{\phi}}}{\ell} - \frac{c_{\tilde{\theta}+\tilde{\phi}}}{d}\right) \\
A_{53} &= \left(\frac{s_{\tilde{\theta}+\tilde{\phi}} s_{\tilde{\phi}}}{\ell} - \frac{c_{\tilde{\theta}+\tilde{\phi}}}{d}\right)\dot{\tilde{x}}_P - \left(\frac{c_{\tilde{\theta}+\tilde{\phi}} s_{\tilde{\phi}}}{\ell} + \frac{s_{\tilde{\theta}+\tilde{\phi}}}{d}\right)\dot{\tilde{y}}_P \\
A_{55} &= -\left(\frac{c_{\tilde{\theta}+2\tilde{\phi}}}{\ell} + \frac{c_{\tilde{\theta}+\tilde{\phi}}}{d}\right)\dot{\tilde{x}}_P - \left(\frac{s_{\tilde{\theta}+2\tilde{\phi}}}{\ell} + \frac{s_{\tilde{\theta}+\tilde{\phi}}}{d}\right)\dot{\tilde{y}}_P
\end{aligned}
$$

and

$$
\begin{aligned}
B_{11} &= 1 \quad B_{22} = 1 \quad B_{31} = \frac{1}{\ell} s_{\tilde{\phi}} c_{\tilde{\theta}+\tilde{\phi}} \quad B_{32} = \frac{1}{\ell} s_{\tilde{\phi}} s_{\tilde{\theta}+\tilde{\phi}} \\
B_{41} &= -\frac{1}{\ell\ell_1}\left(\ell_h s_{\tilde{\phi}} c_{\tilde{\phi}} + \ell_1 s_{\tilde{\phi}} + \ell c_{\tilde{\phi}} s_{\tilde{\psi}}\right) c_{\tilde{\theta}+\tilde{\phi}} \\
B_{42} &= -\frac{1}{\ell\ell_1}\left(\ell_h s_{\tilde{\phi}} c_{\tilde{\phi}} + \ell_1 s_{\tilde{\phi}} + \ell c_{\tilde{\phi}} s_{\tilde{\psi}}\right) s_{\tilde{\theta}+\tilde{\phi}} \\
B_{51} &= -\frac{c_{\tilde{\theta}+\tilde{\phi}} s_{\tilde{\phi}}}{\ell} - \frac{s_{\tilde{\theta}+\tilde{\phi}}}{d} \quad B_{52} = -\frac{s_{\tilde{\theta}+\tilde{\phi}} s_{\tilde{\phi}}}{\ell} + \frac{c_{\tilde{\theta}+\tilde{\phi}}}{d}.
\end{aligned}
$$

Since the proposed framework includes an MPC module (see Fig. 5.4), our algorithm works in discrete-time, producing control inputs $\boldsymbol{u}$ that are piecewise-constant over sampling intervals of duration δ_t.

We then approximate (5.10) with the following piecewise-time-invariant system

$$
\dot{\boldsymbol{\varepsilon}} = \boldsymbol{A}(t_k)\boldsymbol{\varepsilon} + \boldsymbol{B}(t_k)\boldsymbol{u}_{\mathrm{corr},k} \qquad t \in [t_k, t_{k+1}].
$$

This 5-dimensional system is partitioned in a 2- and a 3-dimensional system:

$$
\begin{pmatrix} \dot{\boldsymbol{\varepsilon}}_{\mathrm{s}} \\ \dot{\boldsymbol{\varepsilon}}_{\mathrm{u}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{\mathrm{ss}} & \boldsymbol{0}_{2\times 3} \\ \boldsymbol{A}_{\mathrm{us}}(t_k) & \boldsymbol{A}_{\mathrm{uu}}(t_k) \end{pmatrix} \begin{pmatrix} \boldsymbol{\varepsilon}_{\mathrm{s}} \\ \boldsymbol{\varepsilon}_{\mathrm{u}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{I}_2 \\ \boldsymbol{B}_{\mathrm{u}}(t_k) \end{pmatrix} \boldsymbol{u}_{\mathrm{corr},k}. \tag{5.11}
$$

While it is $\boldsymbol{A}_{\text{ss}} = \text{diag}\{-k_x, -k_y\}$, the three eigenvalues of $\boldsymbol{A}_{\text{uu}}(t_k)$ have positive real part when tracking in backward motion, and thus the dynamics of θ, ψ and ϕ are unstable. This was proven analytically for linear trajectories in Sect. 5.3.2. We do not give a general proof of this claim, which can however be verified numerically.

5.4.4 MPC-based control correction

In the proposed control scheme (Fig. 5.4), the role of IS-MPC is to compute the control correction term $\boldsymbol{u}_{\text{corr}}$ so as to avoid the onset of instability — and hence jackknifing.

As a prediction model, we use the following:

$$\dot{\boldsymbol{\varepsilon}} = \boldsymbol{A}(t_{k+i})\boldsymbol{\varepsilon} + \boldsymbol{B}(t_{k+i})\,\boldsymbol{u}_{\text{corr}} \qquad i = 0, \dots, C-1 \tag{5.12}$$

$$\dot{\boldsymbol{\varepsilon}} = \boldsymbol{A}(t_{k+C})\boldsymbol{\varepsilon} + \boldsymbol{B}(t_{k+C})\,\boldsymbol{u}_{\text{corr}} \qquad i \geq C, \tag{5.13}$$

i.e., the piecewise-time-invariant system (5.11) within the control horizon and the same system frozen at t_{k+C} after that.

In order to transcribe the MPC problem into a QP, we discretize (5.12) to obtain the prediction model valid over the control horizon T_c:

$$\boldsymbol{\varepsilon}_{i+1} = \boldsymbol{A}_{k+i}\boldsymbol{\varepsilon}_i + \boldsymbol{B}_{k+i}\boldsymbol{u}_{\text{corr},i}, \quad i = 0, \dots, C-1, \tag{5.14}$$

with $\boldsymbol{A}_{k+i}$ and $\boldsymbol{B}_{k+i}$ being obtained from the discretization of $\boldsymbol{A}(t_k)$ and $\boldsymbol{B}(t_k)$, respectively, using (2.12).

Stability constraint

We now introduce a stability condition which guarantees that the internal dynamics does not diverge (it is the condition under which the free evolution exactly cancels the divergent component of the forced evolution). To this end, we need a preliminary transformation of the model equations.

For the prediction model after time t_{k+C} (5.13), which is time-invariant, one can use a change of coordinates $\boldsymbol{\eta} = \boldsymbol{T}\boldsymbol{\varepsilon}$, with

$$\boldsymbol{T} = \left(\begin{array}{cc} \boldsymbol{I}_2 & \boldsymbol{0}_{2\times 3} \\ \hline \multicolumn{2}{c}{\boldsymbol{T}_{\text{u}}} \end{array}\right),$$

such that the system becomes block-diagonal:

$$\begin{pmatrix} \dot{\boldsymbol{\eta}}_{\text{s}} \\ \dot{\boldsymbol{\eta}}_{\text{u}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{\text{ss}} & \boldsymbol{0}_{2\times 3} \\ \boldsymbol{0}_{3\times 2} & \boldsymbol{\Lambda}_{\text{u},k+C} \end{pmatrix} \begin{pmatrix} \boldsymbol{\eta}_{\text{s}} \\ \boldsymbol{\eta}_{\text{u}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{I}_2 \\ \boldsymbol{G}_{\text{u},k+C} \end{pmatrix} \boldsymbol{u}_{\text{corr}},$$

with $\boldsymbol{\Lambda}_{\text{u},k+C}$ collecting the unstable eigenvalues of $\boldsymbol{A}_{\text{uu}}(t_{k+C})$.

Following the boundedness condition in Sect. 3.2, we have that if the unstable component $\boldsymbol{\eta}_{\text{u}}$ satisfies

$$\boldsymbol{\eta}_{\text{u}}(t_{k+C}) = -\int_{t_{k+C}}^{\infty} e^{-\boldsymbol{\Lambda}_{\text{u},k+C}(\tau - t_{k+C})} \boldsymbol{G}_{\text{u},k+C} \boldsymbol{u}_{\text{corr}}(\tau) d\tau$$

Then, the evolution of $\boldsymbol{\varepsilon}$ will remain bounded if the inputs $\boldsymbol{u}_{\text{corr}}$ are bounded.

The *stability condition* can be rewritten as the constraint on the transformed final state of the MPC prediction:

$$\boldsymbol{T}_{\mathrm{u}}\boldsymbol{\varepsilon}_C = \boldsymbol{\eta}_{\mathrm{u}}^* = -\int_{t_{k+C}}^{\infty} e^{-\boldsymbol{\Lambda}_{\mathrm{u},k+C}(\tau - t_{k+C})}\boldsymbol{G}_{\mathrm{u},k+C}\boldsymbol{u}_{\mathrm{corr}}(\tau)d\tau. \tag{5.15}$$

Since the left-hand side depends the decision variables of the MPC problem, i.e., $\boldsymbol{u}_{\mathrm{corr},i}$ for $i = 0, \ldots, C-1$, eq. (5.15) can be regarded as a *stability constraint*. Note that the right-side depends on the corrective actions $\boldsymbol{u}_{\mathrm{corr},i}$ for $i \geq C$, i.e., after the control horizon. The latter, collectively referred to as the *tail*, are obviously unknown, and they must be conjectured in order to obtain a *causal* constraint that can be computed at t_k. Possible tails in IS-MPC include the *truncated* tail, which corresponds to setting $\boldsymbol{u}_{\mathrm{corr},i} = \boldsymbol{0}$ for $i \geq k$, and the *periodic* tail, obtained by replication of the corrective actions within the control horizon as $\boldsymbol{u}_{\mathrm{corr},i} = \boldsymbol{u}_{\mathrm{corr},i-C}$ for $i \geq C$. This replication may be infinite or finite; in the second case, the remaining part of the tail is truncated.

Wrapping up, eq. (5.15) is a 3-dimensional linear constraint in the MPC decision variables. At each control cycle the coordinate transformation $\boldsymbol{T}_{\mathrm{u}}$ and the integral $\boldsymbol{\eta}_{\mathrm{u}}^*$ are recomputed using the auxiliary trajectory.

Other constraints

In addition to the stability constraint, the MPC framework allows to introduce practically relevant constraints on the hitch angle ψ and the steering angle ϕ:

$$|\psi_i| \leq \psi_{\max} \qquad |\phi_i| \leq \phi_{\max} \qquad i = 1, \ldots, C, \tag{5.16}$$

where $\psi_{\max}$ and $\phi_{\max}$ are the mechanical limits on the corresponding joints. These constraints are still linear when expressed in the transformed state coordinates $\boldsymbol{\varepsilon}_i = \boldsymbol{q}_i - \tilde{\boldsymbol{q}}_i$.

One may also have box constraints on the velocity inputs:

$$|v_i| \leq v_{\max} \qquad |\omega_i| \leq \omega_{\max} \qquad i = 0, \ldots, C-1.$$

In view of (5.3) and (5.9), these constraints can be written at the generic sampling instant $t_{k+i} \in [t_k, t_{k+C-1}]$ as

$$-\begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} \leq \boldsymbol{D}^{-1}(\theta_i, \phi_i)(\boldsymbol{u}_{\mathrm{track},i} + \boldsymbol{u}_{\mathrm{corr},i}) \leq \begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix}.$$

To transform this into a linear constraint on the MPC decision variable $\boldsymbol{u}_{\mathrm{corr},i}$, we replace $\boldsymbol{D}^{-1}(\theta_i, \phi_i)$ and $\boldsymbol{u}_{\mathrm{track},j}$ with two constant quantities $\bar{\boldsymbol{D}}_i^{-1}$ and $\bar{\boldsymbol{u}}_{\mathrm{track},i}$, obtained by substituting the predicted state at t_{k+i} with its value according to the MPC solution at t_{k-1}, which is known. Therefore, the velocity input constraints finally become

$$-\begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} - \bar{\boldsymbol{D}}_i^{-1}\bar{\boldsymbol{u}}_{\mathrm{track},i} \leq \bar{\boldsymbol{D}}_i^{-1}\boldsymbol{u}_{\mathrm{corr},i} \leq \begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} - \bar{\boldsymbol{D}}_i^{-1}\bar{\boldsymbol{u}}_{\mathrm{track},i}. \tag{5.17}$$

IS-MPC formulation

The IS-MPC algorithm solves at each iteration the following OCP, obtained combining the prediction model (5.14), the stability constraint (5.15), the state constraints (5.16) and the input constraint (5.17):

$$
\begin{cases}
\underset{\boldsymbol{u}_{\text{corr}}}{\text{minimize}} & \displaystyle\sum_{i=0}^{C-1} \|\boldsymbol{u}_{\text{corr},i}\|^2 & \\
\text{subject to} & \boldsymbol{\varepsilon}_0 = \hat{\boldsymbol{\varepsilon}}_k & \\
& \boldsymbol{\varepsilon}_{i+1} = \boldsymbol{A}_{k+i}\boldsymbol{\varepsilon}_i + \boldsymbol{B}_{k+i}\boldsymbol{u}_{\text{corr},i} & \forall i \in \mathbb{I}_0^{C-1} \\
& \bar{\boldsymbol{D}}_i^{-1}\boldsymbol{u}_{\text{corr},i} \leq \begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} - \bar{\boldsymbol{D}}_i^{-1}\bar{\boldsymbol{u}}_{\text{track},i} & \forall i \in \mathbb{I}_0^{C-1} \\
& \bar{\boldsymbol{D}}_i^{-1}\boldsymbol{u}_{\text{corr},i} \geq -\begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} - \bar{\boldsymbol{D}}_i^{-1}\bar{\boldsymbol{u}}_{\text{track},i} & \forall i \in \mathbb{I}_0^{C-1} \\
& |\psi_i| \leq \psi_{\max} & \forall i \in \mathbb{I}_1^{C} \\
& |\phi_i| \leq \phi_{\max} & \forall i \in \mathbb{I}_1^{C} \\
& \boldsymbol{T}_{\text{u}}\boldsymbol{\varepsilon}_C = \boldsymbol{\eta}_{\text{u}}^* &
\end{cases}
$$

The choice of the cost function reflects the fact that the corrective action $\boldsymbol{u}_{\text{corr}}$ in (5.9) will perturb exact tracking, and therefore it should be limited to the minimum necessary. Being the cost function quadratic and the constraints linear, the MPC can easily be transcribed into a QP.

The actual expression of the stability constraint (5.15) will depend on the chosen tail (truncated, infinite-periodic, finite-periodic). As customary with MPC, only the first corrective action $\boldsymbol{u}_{\text{corr},0}$ is actually used as real-time control, and a new QP problem is set up and solved at the next sampling instant.

5.5 Results

5.5.1 Simulations

As a preliminary validation of the proposed method, we have performed numerical simulations in MATLAB for a tractor-trailer vehicle having the same kinematic parameters of our physical prototype, which will be described in detail in the next section. Accordingly, we have set $\ell = 0.255$ m, $\ell_h = 0.065$ m, $\ell_1 = 0.263$ m, $\psi_{\max} = \pi/4$ rad, $\phi_{\max} = \pi/12$ rad, $v_{\max} = 0.5$ m/s, $\omega_{\max} = 1.5$ rad/s. As for the control scheme, we have chosen $d = 0.1$ m and $k_x = k_y = 1$. The sampling interval is $\delta_t = 0.1$ s, while the MPC control horizon is $T_c = 5$ s. A finite-periodic tail with 2 replications was used in the stability constraint (5.15). The accompanying video[5] contains clips of all the simulations.

In the first simulation, the vehicle starts from $\boldsymbol{q}_0 = (6, 0.01, 0, 0, 0)$ [m, m, rad, rad, rad] and must track in backward motion the linear reference trajectory

$$x_{\text{ref}} = 6 + v_{\text{ref}}\, t, \qquad y_{\text{ref}} = 0,$$

[5] https://youtu.be/ImG1ZV1EYBs

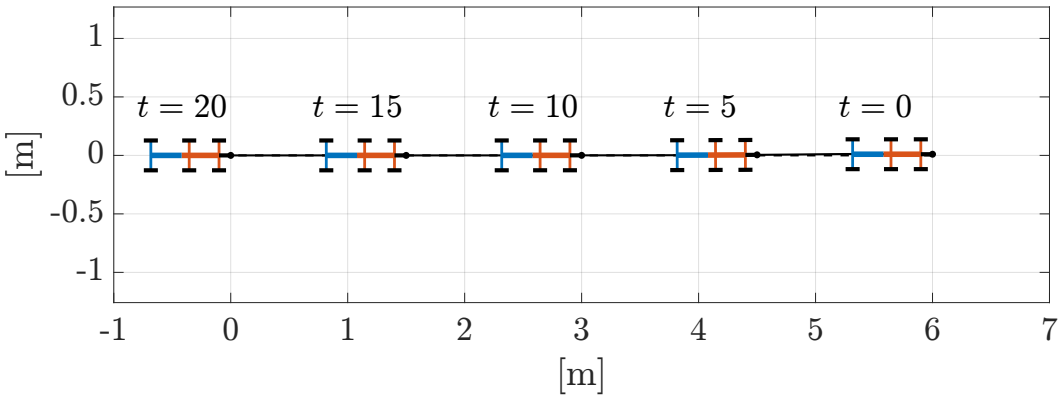

Figure 5.7. Simulation 1: Stable backward tracking of a linear trajectory by the proposed method (tractor in red, trailer in blue).

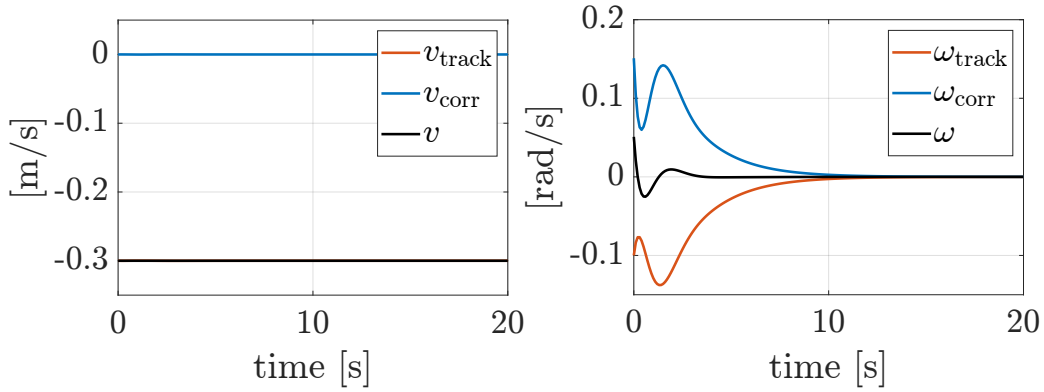

Figure 5.8. Simulation 1: Driving and steering velocities.

with $v_{\text{ref}} = -0.3$ m/s[6]. In the absence of a corrective action ($\boldsymbol{u} = \boldsymbol{u}_{\text{track}}$), jackknifing will occur, similarly to what happens in Fig. 5.3. The inclusion of the corrective action generated by IS-MPC in the control law ($\boldsymbol{u} = \boldsymbol{u}_{\text{track}} + \boldsymbol{u}_{\text{corr}}$) successfully prevents the phenomenon, producing stable tracking of the reference trajectory, see Fig. 5.7. The tracking error reaches a transient peak value of 0.012 m in norm and then converges to zero. The driving and steering velocities plots of Fig. 5.8 indicate that in this case only the steering velocity ω is affected by the corrective action, which tends to counteract the feedback component of the tracking action. For completeness, we have also run the same simulation for the case $v_{\text{ref}} = -0.8$ m/s. The obtained results, which are only shown in the accompanying video, confirm that the proposed method is effective independently of the reference speed.

In the second simulation, a circular trajectory with a radius of 5 m and a tangential velocity of 0.25 m/s must be tracked in backward motion starting from $\boldsymbol{q}_0 = (0, 0.01, -0.07, 0.03, 0.05)$ [m, m, rad, rad, rad]. The proposed strategy is again effective, as shown by the results in Figs. 5.9–5.10; note how in this case also the driving velocity is involved in the corrective action. In this case, the transient

[6]Since the total length of our vehicle is around 60 cm, the speed-over-length ratio is 0.5. For a typical tractor-trailer truck, whose average length in the US is 22 m, this would correspond to a speed of 40 km/h circa.

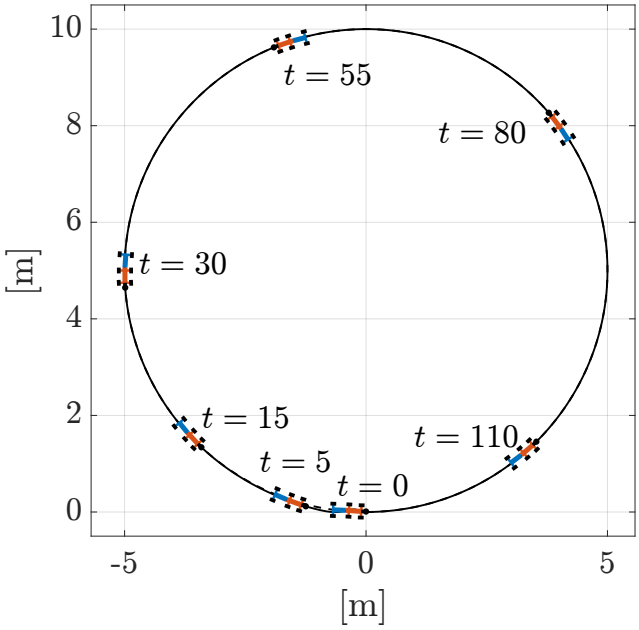

Figure 5.9. Simulation 2: Stable backward tracking of a circular trajectory by the proposed method.

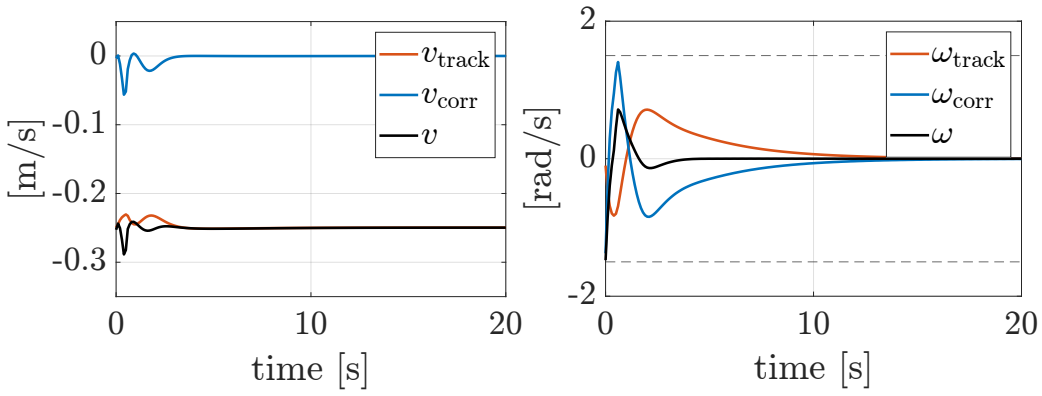

Figure 5.10. Simulation 2: Driving and steering velocities.

peak of the tracking error is 0.052 m. Under pure tracking control, jackknifing would instead occur at the very start of the motion (see the accompanying video).

Finally, in the third simulation the vehicle must track in backward motion an eight-shaped reference trajectory starting from $\boldsymbol{q}_0 = (0, 0.05, 3.92, -0.09, 0)$ [m, m, rad, rad, rad]. The results, shown in Figs. 5.11–5.13, confirm that the proposed method can avoid the jackknifing phenomenon over generic trajectories, including those with variable curvature. The peak value of the Cartesian error during the transient is 0.1 m. Figure 5.12, right, also proves that state constraints are correctly taken into account by the MPC algorithm; the same is true for the input constraints. As before, jackknifing immediately occurs without corrective action, see the accompanying video.

Overall, the above simulation confirm that the inclusion of the stability con-

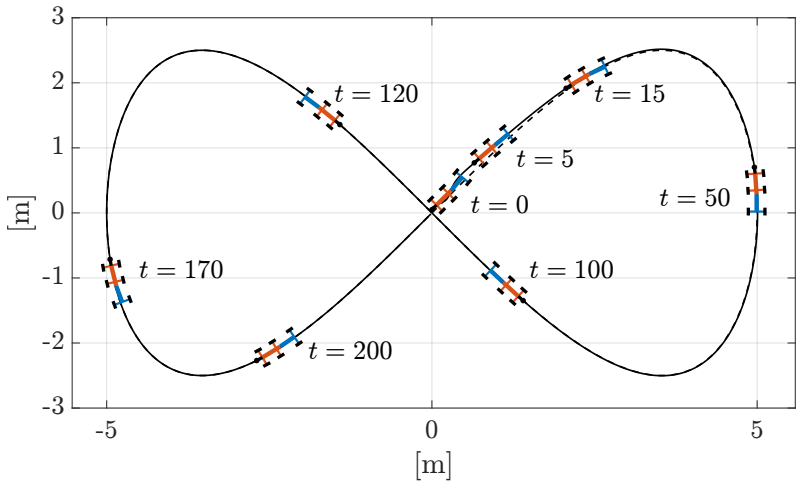

Figure 5.11. Simulation 3: Stable backward tracking of an eight-shaped trajectory by the proposed method.

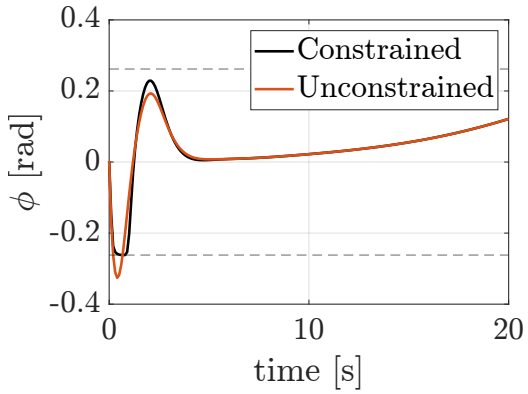

Figure 5.12. Simulation 3: Evolution of the steering angle ϕ; note the saturation in the early part of the motion. Also shown is the steering angle (red) generated by the proposed method if the state and input constraints are removed.

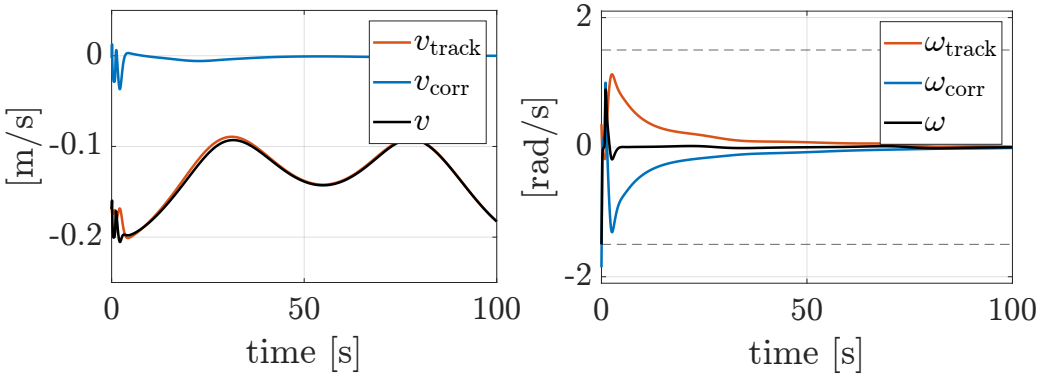

Figure 5.13. Simulation 3: Driving and steering velocities.

straint in our MPC formulation guarantees that the whole zero dynamics (i.e., the steering angle, the vehicle orientation and the hitch angle) will be stable during the

motion. In particular, this means that also the trailer orientation (which is the sum of the vehicle orientation θ and the hitch angle ψ) is guaranteed to be bounded.

5.5.2 Comparison with an alternative method

To highlight the peculiarities of the proposed method, we have implemented the anti-jackknifing MPC controller originally presented by Ljungqvist *et al.* in [100] and performed a comparison of the simulation results.

The alternative method is substantially different to the proposed one, since the addressed control problem is to track in backward motion a path in *state* space (i.e., the configuration space), as opposed to tracking in *output* space. This is much more convenient in most applications, where the objective is simply to track a Cartesian trajectory, while the motion of the vehicle in terms of the angular state variables (θ, ψ, ϕ) is not assigned a priori. In this case, the method in [100] would require to plan in advance a feasible, stable state trajectory associated to the desired output trajectory; however, while this is relatively easy for special trajectories (e.g., linear, circular) and special systems (e.g., differentially flat), no technique is available for solving this problem in general.

To achieve local stabilization, a terminal state cost is set as the value function solution of the discrete-time algebraic Riccati equation (DARE) of the associated infinite-horizon Linear Quadratic Regulator [29].

In this case, we do not perform any input-output feedback linearization, but we linearize model (5.2) after having applied the change of coordinates $(x, y) \rightarrow (x_P, y_P)$, obtaining a model of the form

$$\dot{\boldsymbol{q}} = \boldsymbol{F}(t)\boldsymbol{q} + \boldsymbol{G}(t)\boldsymbol{\nu}. \tag{5.18}$$

To perform the tracking task, a desired *state* trajectory $\boldsymbol{q}_d(t)$ is selected, with a feedforward input $\boldsymbol{\nu}_d(t)$ used to generate such trajectory. Then, the MPC cost function is designed as

$$L(\boldsymbol{\nu}) = \|\boldsymbol{q}_C - \boldsymbol{q}_d(t_{k+C})\|_{\boldsymbol{V}^*} + \sum_{i=0}^{C-1} \|\boldsymbol{q}_i - \boldsymbol{q}_d(t_{k+i})\|_{\boldsymbol{Q}} + \|\boldsymbol{\nu}_i - \boldsymbol{\nu}_d(t_{k+i})\|_{\boldsymbol{R}}, \tag{5.19}$$

Where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive-definite diagonal weighting matrices, $\boldsymbol{V}^*$ is the solution to the DARE of the LQR problem $(\bar{\boldsymbol{F}}, \bar{\boldsymbol{G}}, \boldsymbol{Q}, \boldsymbol{R})$, with $\bar{\boldsymbol{F}}, \bar{\boldsymbol{G}}$ being the discretized version of dynamics (5.18) around a straight path. The DARE is solved offline using the `idare` function in MATLAB.

To avoid jackknifing, an artificial upper bound ψ^* on the hitch angle is enforced as a constraint in the MPC problem; such upper bound is empirically determined by considering many different initial conditions and verifying by simulation which of them trigger the jackknifing phenomenon.

Finally, the MPC problem for the alternative controller is:

$$\begin{cases} \underset{\boldsymbol{\nu}}{\text{minimize}} & L(\boldsymbol{\nu}) \text{ from (5.19)} & \\ \text{subject to} & \boldsymbol{q}_0 = \hat{\boldsymbol{q}}_k & \\ & \boldsymbol{q}_{i+1} = \boldsymbol{F}_{k+i}\boldsymbol{q}_i + \boldsymbol{G}_{k+i}\boldsymbol{\nu}_i & \forall i \in \mathbb{I}_0^{C-1} \\ & -\begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} \leq \boldsymbol{\nu}_i \leq \begin{pmatrix} v_{\max} \\ \omega_{\max} \end{pmatrix} & \forall i \in \mathbb{I}_0^{C-1} \\ & |\psi_i| \leq \psi^* & \forall i \in \mathbb{I}_1^{C} \\ & |\phi_i| \leq \phi_{\max} & \forall i \in \mathbb{I}_1^{C} \end{cases}$$

For the following simulations, we have set $\boldsymbol{Q} = \text{diag}(5, 50, 5, 1, 1)$ and $\boldsymbol{R} = \text{diag}(1, 0.2)$, while the rest of the parameters are set equal to IS-MPC.

In the first simulation, the vehicle must track in backward motion a rectilinear Cartesian trajectory. For this case, it is trivial to find a compatible state trajectory which is stable (all angular variables should be zero). As already explained, our method does not introduce any artificial bound on the hitch angle other than the one derived from mechanical limits, as the onset of jackknifing is avoided thanks to the stability constraint. The results, shown in Fig. 5.14, demonstrate how our IS-MPC method achieves faster convergence and more accurate tracking of the reference trajectory; in particular, the hitch angle with IS-MPC safely exceeds the artificial upper bound (shown as a dashed line in the ψ plot) without any consequence in terms of jackknifing.

In the second simulation the vehicle must track in backward motion a circular trajectory. Also in this case it is easy to compute a compatible state trajectory which is stable. The results, shown in Fig. 5.15, show that the method in [100] cannot track this trajectory because the artificial upper bound on the hitch angle does not allow it. Conversely, IS-MPC achieves perfect tracking. As before, ψ

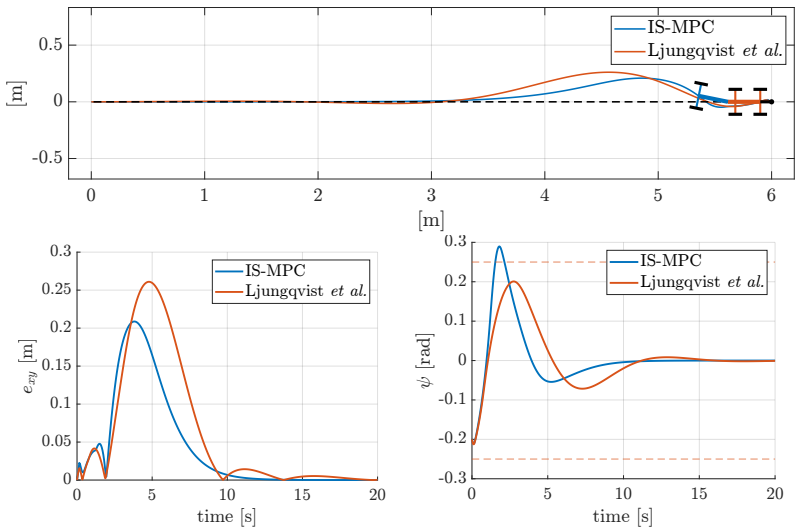

Figure 5.14. Tracking a linear trajectory in backward motion: comparison between [100] and the proposed IS-MPC method.

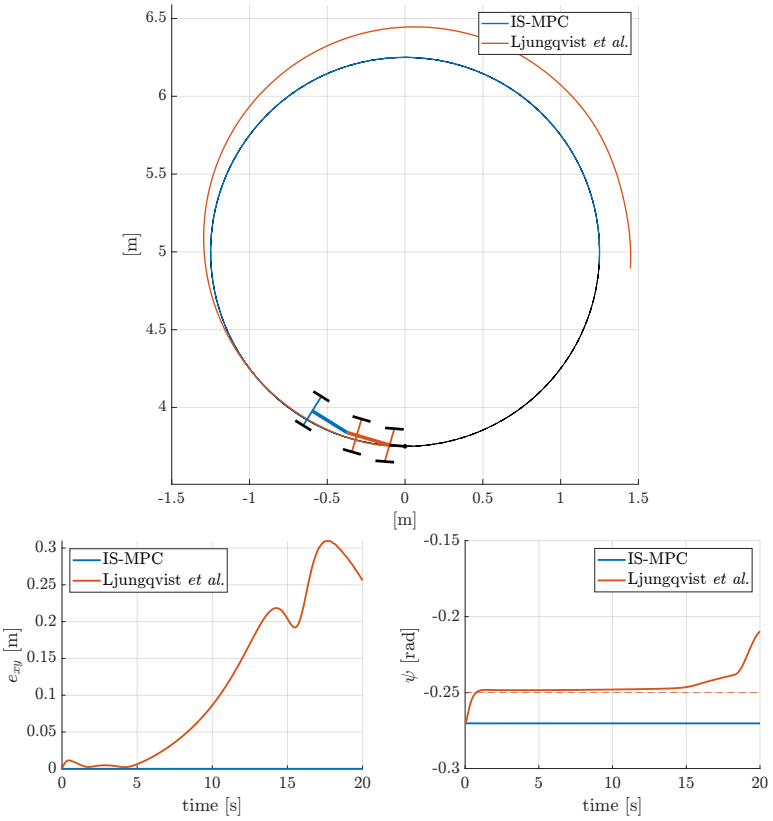

Figure 5.15. Tracking a circular trajectory in backward motion: comparison between [100] and the proposed IS-MPC method.

safely exceeds the artificial bound of [100] during the motion but no jackknifing phenomenon is experienced.

5.5.3 Experiments

To perform an experimental validation of the proposed method, we have built a prototype tractor-trailer system starting from a commercial radio-controlled model, the Carson Unimog U300, a 1:12 replica of the Mercedes Unimog U300 truck. The kinematic parameters of the vehicle have been given at the beginning of the previous section. The model was extensively modified and instrumented as needed in order to implement the antijackknifing controller. In particular:

- the original electronics were replaced with two Arduino Uno microcontroller boards in a master-slave configuration, connected via the I^2C protocol;
- an H-bridge was added for driving the two DC motors;
- encoders were mounted on the trailer wheels as well as on the hitch joint;
- a USB/Bluetooth module was added for communication between the master board and the external computer where the antijackknifing controller runs.

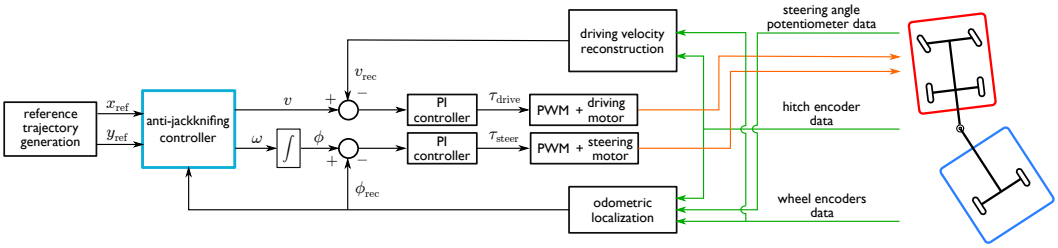

Figure 5.16. The overall control architecture of the vehicle, with input signals and output signals shown in green and orange, respectively. The anti-jackknifing controller runs on an external computer that communicates with the vehicle via Bluetooth.

The external computer is a standard laptop that receives from the master board the current odometric estimate of the robot configuration, obtained from encoder measurements via 2nd-order Runge-Kutta integration of model (5.2). It then executes in real time the IS-MPC algorithm in MATLAB, using `quadprog` as QP solver, and sends the resulting velocity commands v and ω back to the master board. This high-level control cycle runs at a rate of 10 Hz.

The master board sends the velocity commands to the slave board, which runs the low-level PI controllers of the motors. In particular, the driving velocity error is computed by comparing v with v_{rec}, the current driving velocity of the tractor as reconstructed from trailer wheel encoder data. As for the steering motor, ω is integrated to provide a reference for the steering angle ϕ, which is then compared with ϕ_{rec}, the current steering angle as reconstructed via odometric localization. The PI loops for the driving speed and the steering angle run respectively at 20 and 100Hz.

Figure 5.16 shows how the anti-jackknifing block of Fig. 5.4 has been merged in the overall control architecture of the vehicle.

The control parameters d and k_x, k_y are the same of the simulations, but the MPC control horizon T_c has been reduced to 1 s to decrease control delay and ensure that the controller runs comfortably in real-time in spite of the additional communication overhead.

Figures 5.17 and 5.18 show the results obtained on linear and circular trajectories, respectively. It is clear that the proposed method achieves stable backward tracking in both cases, fully corroborating the positive outcome of the simulations and ultimately proving its robustness to the inevitable uncertainties and perturbations that affect such a low-cost experimental setup. See the accompanying video[7] for clips and data plots of these experiments.

[7] https://youtu.be/ImG1ZV1EYBs

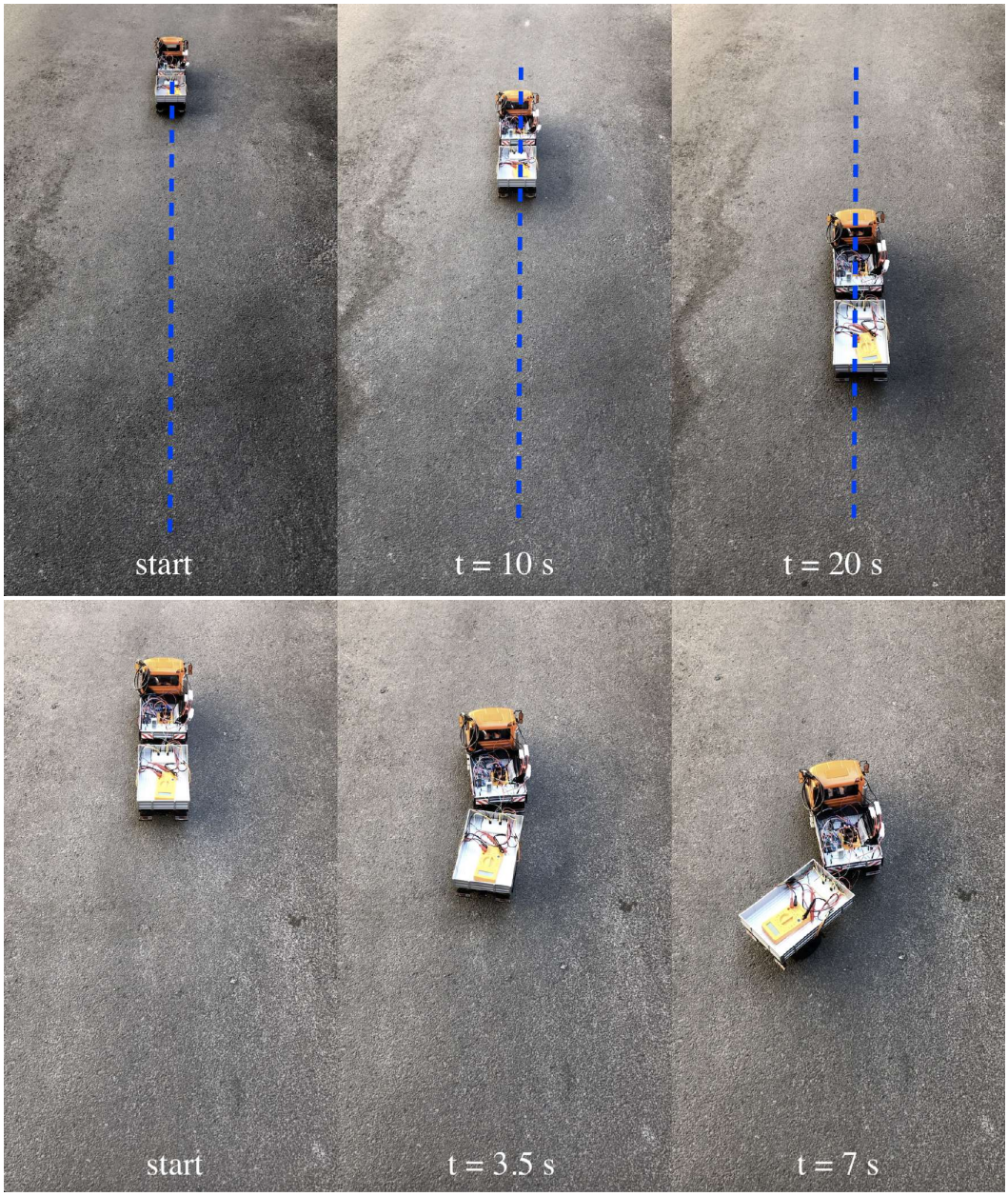

Figure 5.17. Experiment 1: Stable backward tracking of a linear trajectory by the proposed method (top, reference trajectory in blue); jackknife occurs if no corrective action is added (bottom, zoom on the initial part of the motion).

5.6 Extension to the two-trailer system

The proposed method can be extended in principle to a vehicle towing any number of trailers. For illustration, we present below some results for the case of two trailers.

Consider the vehicle in Fig. 5.19, in which a second trailer of length ℓ_2 has been hooked at a distance ℓ_{h_1} from the wheel axle midpoint of the first trailer. Correspondingly, the state vector $\boldsymbol{q}$ is augmented by including the ψ_2 angular coordinate.

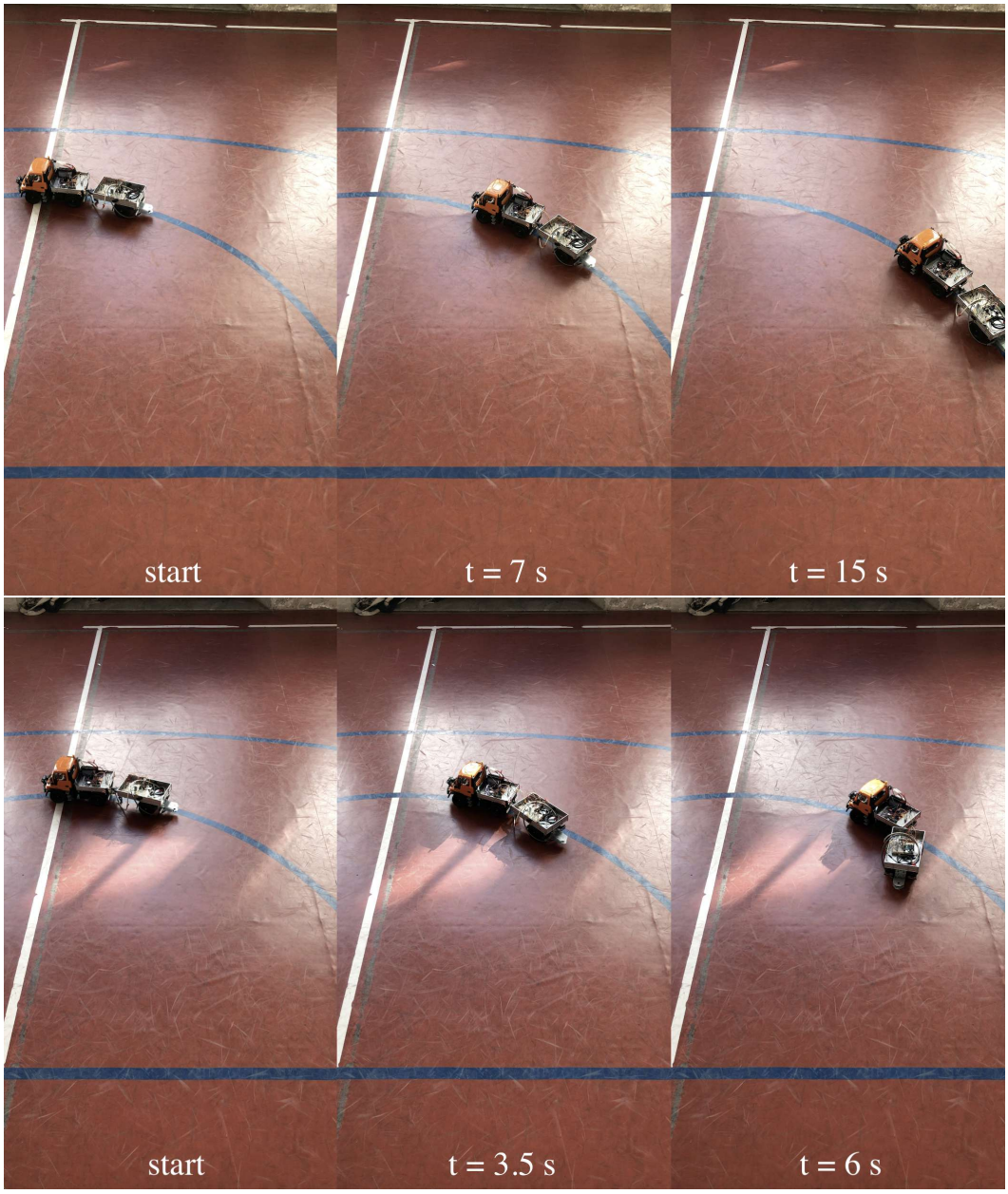

Figure 5.18. Experiment 2: Stable backward tracking of a circular trajectory by the proposed method (top, reference trajectory in blue); jackknife occurs if no corrective action is added (bottom, zoom on the initial part of the motion).

The kinematic model can be written as

$$
\begin{aligned}
\dot{x} &= v\, c_\theta \\
\dot{y} &= v\, s_\theta \\
\dot{\theta} &= \frac{v\, t_\phi}{\ell} \\
\dot{\psi}_1 &= -\frac{v\, t_\phi}{\ell}\left(1 + \frac{\ell_h}{\ell_1}\, c_{\psi_1}\right) - \frac{v\, s_{\psi_1}}{\ell_1} \\
\dot{\psi}_2 &= \frac{v\, \ell_{h_1} t_\phi}{\ell\, \ell_2}\left(\frac{(\ell_1 + \ell_{h_1}) c_{\psi_1} c_{\psi_2}}{\ell_1} + \frac{\ell_2 c_{\psi_1}}{\ell_1} - c_{\psi_1+\psi_2}\right) \\
&\quad + \frac{v\, s_{\psi_1}}{\ell_1}\left(1 + \frac{(\ell_1 + \ell_{h_1}) c_{\psi_2}}{\ell_2}\right) - \frac{v\, s_{\psi_1+\psi_2}}{\ell_2} \\
\dot{\phi} &= \omega.
\end{aligned}
\tag{5.20}
$$

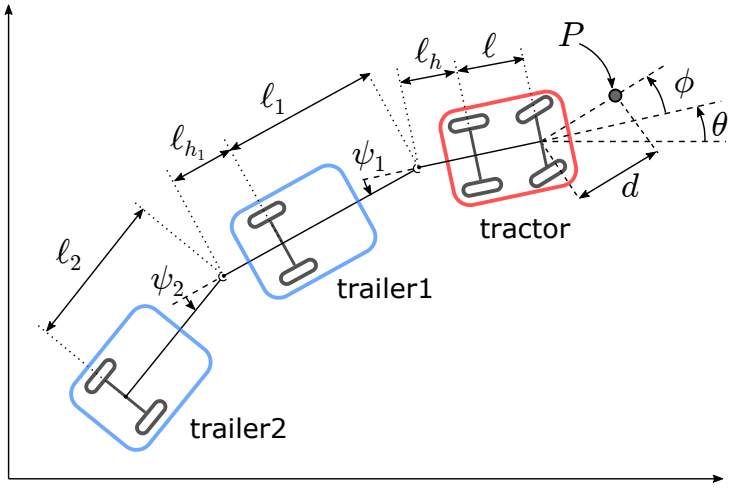

Figure 5.19. A vehicle composed by a tractor and two trailers.

The developments for this model (input-output linearization, generation of the auxiliary trajectory, linearization around it, and MPC-based control correction) are essentially the same of Sect. 5.4, the only notable difference being that the dimension of the unstable subsystem in (5.11), corresponding to the number of eigenvalues with positive real part, is now increased to 4. For an n-trailer system, this number would be $n + 2$.

To validate experimentally the above extension, a second trailer identical to the first was added to our vehicle, with an encoder measuring the corresponding hitch angle ψ_2. The proposed method was successfully tested over linear, circular and spline trajectories (Figs. 5.20–5.22), producing stable backward tracking in all cases. See the accompanying video[8] for clips of these experiments.

[8] https://youtu.be/ImG1ZV1EYBs

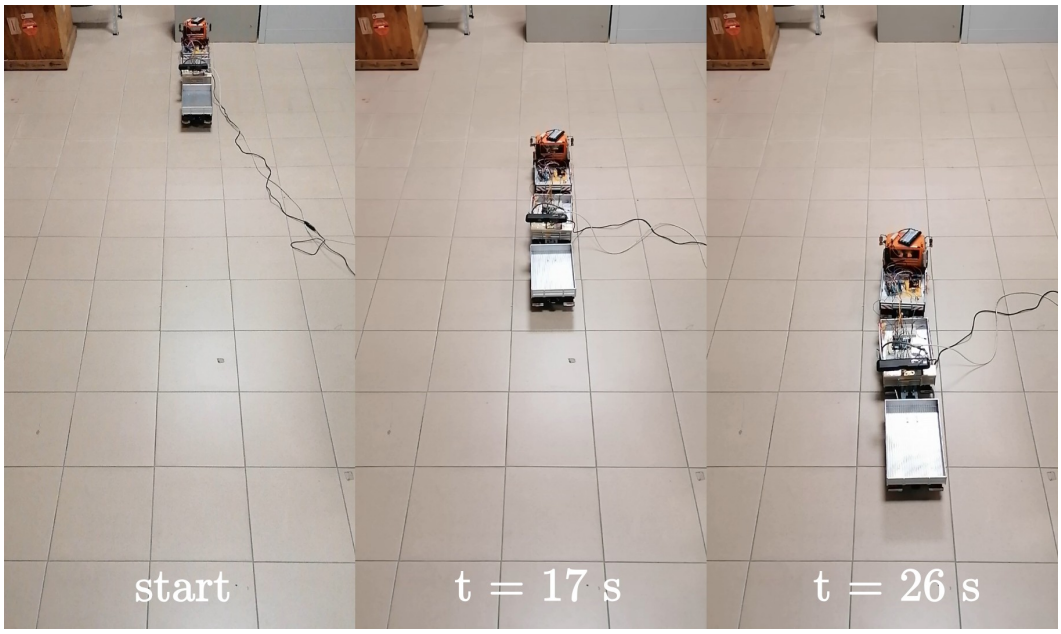

Figure 5.20. Experiment 4: Stable backward tracking of a linear trajectory for a two-trailer vehicle.

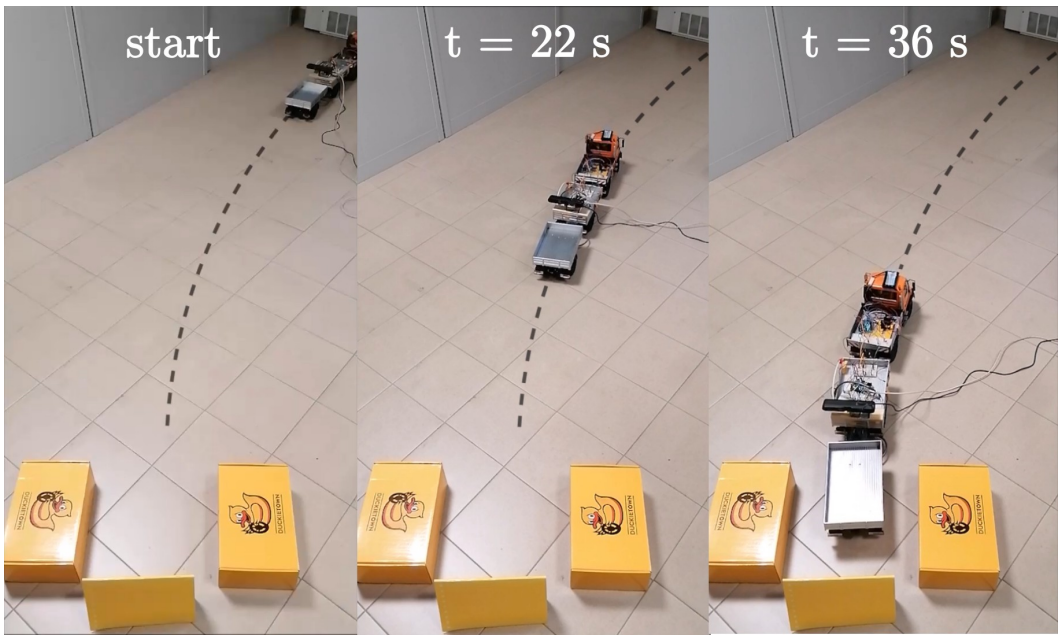

Figure 5.21. Experiment 5: Stable backward tracking of a circular trajectory for a two-trailer vehicle (reference trajectory in black).

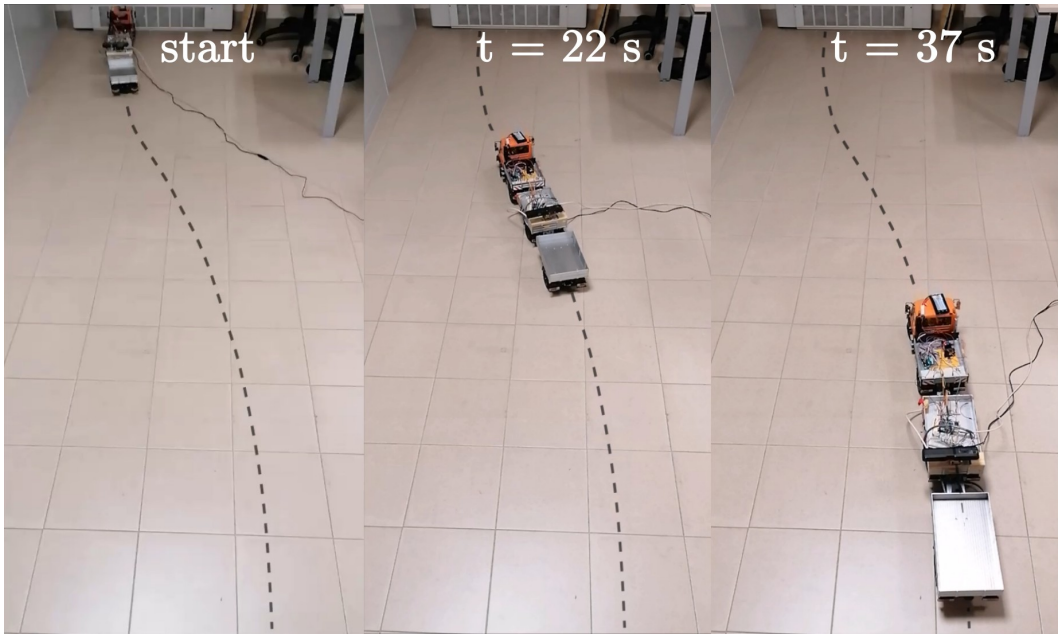

Figure 5.22. Experiment 6: Stable backward tracking of a spline trajectory for a two-trailer vehicle (reference trajectory in black).

Part II

Robust Motion Generation using Sensitivity-Based Tubes

Chapter 6

Closed-Loop Sensitivity

Aerial robotics has experienced a growing interest propelled by advancements in research, resulting in numerous practical uses. A primary concern with aerial robots is to improve system autonomy and ensure safety during motion tasks. Nevertheless, attaining high levels of autonomy for flying robots remains an enduring challenge due to the need to operate in unpredictable and uncertain real-world conditions. Despite efforts to create precise models, real-world complexities introduce several uncertainties among which are inaccuracies in the model parameters, which can potentially disrupt the system behavior during task execution. The pursuit of greater autonomy, precision, and safety in aerial robotics requires bridging the gap between theoretical models and practical real-world conditions. In many cases of interest, a model of the robot/environment can be considered available with the main source of uncertainty lumped in the inaccurate knowledge of some model parameters. In these cases, the uncertainty is not *generic* but it has a very specific structure which, if exploited, can lead to better predictions of its effects on the robot motion. Adaptive [102–104] or robust control [105, 106] are typical methods to deal with parametric uncertainty, either through online parameter estimation or by introducing trade-offs between performance and robustness vs. parametric uncertainty. When MPC is employed, its effectiveness depends on the model, and thus parameters, accuracy [2, 107]. Uncertainties in these parameters can significantly impact both the controller performance and robot behavior. Instead of simply reacting to disturbances to counteract their effect, another possibility is to plan feedforward trajectories with minimal *state sensitivity*, as in [108–110] to minimize the effect of uncertainties over the state trajectory. However, these approaches operate in an open-loop fashion and thus do not consider the presence and strengths/weaknesses of the motion controller that is eventually implemented on the robot.

In light of these considerations, some recent works [111–116] have introduced and exploited the notion of *closed-loop state sensitivity matrix*: this quantity locally captures how deviations in the model parameters (w.r.t. their nominal values) affect the evolution of the robot/environment states in *closed-loop*, i.e., by also taking into account the strengths/weaknesses of the particular control action chosen for executing the task. A norm of the state sensitivity can, for instance, be minimized for generating reference trajectories that result by construction minimally sensitive to parametric uncertainties, thus increasing the intrinsic robustness of their tracking

in closed-loop. This is particularly relevant whenever the task requirements allow for some redundancy in the reference trajectory that can be optimized w.r.t. the sensitivity metrics, e.g., in case of collision-free navigation in cluttered environments or regulation tasks. The notion of state sensitivity has been later extended to also consider the effects of uncertainties on the control inputs [112], by defining and minimizing the so-called *input sensitivity*: this allows, for instance, to better cope with actuation limits that might otherwise be violated when the robot deviates from its nominal trajectory [115]. Finally, it is also possible to leverage the sensitivity matrix to obtain time-varying bounds (tubes) on the state and/or input evolution assuming a (known) range of variation for the parameters. This makes it possible to plan robust trajectories that, at least locally, ensure the feasibility of the resulting motion (against state/input constraints) also in the presence of parametric uncertainties.

While these methods have shown promise in simulated case studies, only in [116] the notion of *closed-loop state/input sensitivity* was applied to real-world experiments with a robotic manipulator. Notably, there have been no real implementations using Quadrotors executing a motion task yet.

In this chapter, we will first review the basics on the closed-loop sensitivity metric and on the computation of tubes of perturbed trajectories, based on such metric, to then present an experimental validation of its use in the planning of robust reference trajectories for a Quadrotor. Although the discussion revolves around the *offline* optimization of the reference trajectory, to be tracked by a traditional controller, the main results are still relevant to the *online* MPC case, which will be discussed in the next chapter.

6.1 Parametric sensitivity of closed-loop systems

Consider a discrete-time dynamical system described by

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}) \tag{6.1}$$

under the action of a controller having the generic form

$$\begin{aligned} \boldsymbol{\xi}_{k+1} &= \boldsymbol{\zeta}(\boldsymbol{x}_k, \boldsymbol{\xi}_k, \boldsymbol{p}_c) \\ \boldsymbol{u}_k &= \boldsymbol{\mu}(\boldsymbol{x}_k, \boldsymbol{\xi}_k, \boldsymbol{p}_c), \end{aligned} \tag{6.2}$$

where $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$ and $\boldsymbol{u}_k \in \mathbb{R}^{n_u}$ denote the state and input vectors, $\boldsymbol{\xi}_k \in \mathbb{R}^{n_\xi}$ the internal states of the possibly dynamic controller, $\boldsymbol{p} \in \mathbb{R}^{n_p}$ the (uncertain) model parameters, and $\boldsymbol{p}_c \in \mathbb{R}^{n_p}$ the *nominal* parameters used by the controller. The combination of (6.1) and (6.2) determines the closed-loop system

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}) \\ \boldsymbol{\xi}_{k+1} = \boldsymbol{\zeta}(\boldsymbol{x}_k, \boldsymbol{\xi}_k, \boldsymbol{p}_c) \\ \boldsymbol{u}_k = \boldsymbol{\mu}(\boldsymbol{x}_k, \boldsymbol{\xi}_k, \boldsymbol{p}_c) \end{cases} \tag{6.3}$$

and the map $\boldsymbol{\phi}_k(\boldsymbol{x}_0, \boldsymbol{\xi}_0) \coloneqq \boldsymbol{\phi}(\boldsymbol{x}_0, \boldsymbol{\xi}_0, t_k) = \boldsymbol{x}_k$ denotes the state solution of (6.3) at time t_k starting from an initial condition $(\boldsymbol{x}_0, \boldsymbol{\xi}_0)$ at t_0. Given a known initial

condition $(\boldsymbol{x}_0, \boldsymbol{\xi}_0)$, the closed-loop trajectory of the system will depend on the actual value of the parameters $\boldsymbol{p}$. We denote with $\bar{\boldsymbol{x}}$ the closed-loop trajectory obtained in the *nominal* case, i.e., when $\boldsymbol{p} = \boldsymbol{p}_c$, and with $\bar{\boldsymbol{\xi}}$, $\bar{\boldsymbol{u}}$ the associated nominal trajectories. In general, however, $\boldsymbol{p} \neq \boldsymbol{p}_c$ because of parametric uncertainty and, therefore, the actual state/input trajectories $(\boldsymbol{x}, \boldsymbol{u})$ will deviate from the nominal ones $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$.

Following [111, 112], we use the notion of closed-loop sensitivity to describe how changes in the parameters affect the state trajectory under the action of the controller. Define the *state sensitivity* $\boldsymbol{\Pi} \in \mathbb{R}^{n_x \times n_p}$ as

$$\boldsymbol{\Pi}_k = \left. \frac{d\boldsymbol{\phi}_k(\boldsymbol{x}_0, \boldsymbol{\xi}_0)}{d\boldsymbol{p}} \right|_{\boldsymbol{p}=\boldsymbol{p}_c}. \tag{6.4}$$

A closed-form expression for (6.4) is, in general, not available but, as shown in [111], it is possible to obtain the following update rule

$$\begin{aligned} \boldsymbol{\Pi}_{k+1} &= \frac{d\boldsymbol{\phi}_{k+1}(\boldsymbol{x}_0, \boldsymbol{\xi}_0)}{d\boldsymbol{p}} = \frac{d\boldsymbol{f}(\boldsymbol{\phi}_k(\boldsymbol{x}_0, \boldsymbol{\xi}_0), \boldsymbol{u}_k, \boldsymbol{p})}{d\boldsymbol{p}} \\ &= \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{\phi}_k(\boldsymbol{x}_0, \boldsymbol{\xi}_0)}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \frac{d\boldsymbol{u}_k}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} \\ &= \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \boldsymbol{\Pi}_k + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \frac{d\boldsymbol{u}_k}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}}, \end{aligned} \tag{6.5}$$

i.e., a discrete-time prediction model for the state sensitivity $\boldsymbol{\Pi}_k$ with initialization $\boldsymbol{\Pi}_0 = \boldsymbol{0}$ since the initial state is assumed known and by definition $\boldsymbol{\phi}_0(\boldsymbol{x}_0, \boldsymbol{\xi}_0) = \boldsymbol{x}_0$.

From (6.5) we also define matrix $\boldsymbol{\Theta} \in \mathbb{R}^{n_u \times n_p}$

$$\boldsymbol{\Theta}_k = \left. \frac{d\boldsymbol{u}_k}{d\boldsymbol{p}} \right|_{\boldsymbol{p}=\boldsymbol{p}_c} = \frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{x}} \boldsymbol{\Pi}_k + \frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\xi}} \frac{d\boldsymbol{\xi}_k}{d\boldsymbol{p}}$$

and $\boldsymbol{\Pi}_{\boldsymbol{\xi}} \in \mathbb{R}^{n_\xi \times n_p}$

$$\boldsymbol{\Pi}_{\boldsymbol{\xi}} = \left. \frac{d\boldsymbol{\xi}_k}{d\boldsymbol{p}} \right|_{\boldsymbol{p}=\boldsymbol{p}_c}$$

which, applying the same reasoning of (6.5), has the prediction model

$$\boldsymbol{\Pi}_{\boldsymbol{\xi},k+1} = \frac{\partial \boldsymbol{\zeta}}{\partial \boldsymbol{x}} \boldsymbol{\Pi}_k + \frac{\partial \boldsymbol{\zeta}}{\partial \boldsymbol{\xi}} \boldsymbol{\Pi}_{\boldsymbol{\xi},k},$$

with initial condition $\boldsymbol{\Pi}_{\boldsymbol{\xi},0} = \boldsymbol{0}$. Matrix $\boldsymbol{\Theta}_k$, denoted as *input sensitivity*, represents the indirect effect of a parameter change on the control action by means of feedback, which can be seen as a measure of how the input in closed-loop will deviate from its nominal trajectory $\bar{\boldsymbol{u}}$. In case the controller has internal states, the input sensitivity depends on matrix $\boldsymbol{\Pi}_{\boldsymbol{\xi},k}$, which represents the sensitivity of the controller states.

Finally, let

$$\begin{aligned} \boldsymbol{A}_k &= \left. \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \boldsymbol{p}_c} & \boldsymbol{B}_k &= \left. \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \boldsymbol{p}_c} & \boldsymbol{M}_k &= \left. \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \boldsymbol{p}_c} \\ \boldsymbol{F}_k &= \left. \frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{x}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{\xi}}_k, \boldsymbol{p}_c} & \boldsymbol{G}_k &= \left. \frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\xi}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{\xi}}_k, \boldsymbol{p}_c} \\ \boldsymbol{Z}_k &= \left. \frac{\partial \boldsymbol{\zeta}}{\partial \boldsymbol{x}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{\xi}}_k, \boldsymbol{p}_c} & \boldsymbol{Y}_k &= \left. \frac{\partial \boldsymbol{\zeta}}{\partial \boldsymbol{\xi}} \right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{\xi}}_k, \boldsymbol{p}_c} \end{aligned}$$

such that the closed-loop sensitivity prediction model can be compactly rewritten as

$$\boldsymbol{\Pi}_{k+1} = \boldsymbol{A}_k\boldsymbol{\Pi}_k + \boldsymbol{B}_k\left(\boldsymbol{F}_k\boldsymbol{\Pi}_k + \boldsymbol{G}_k\boldsymbol{\Pi}_{\boldsymbol{\xi},k}\right) + \boldsymbol{M}_k, \tag{6.6}$$

$$\boldsymbol{\Pi}_{\boldsymbol{\xi},k+1} = \boldsymbol{Z}_k\boldsymbol{\Pi}_k + \boldsymbol{Y}_k\boldsymbol{\Pi}_{\boldsymbol{\xi},k}, \tag{6.7}$$

$$\boldsymbol{\Theta}_k = \boldsymbol{F}_k\boldsymbol{\Pi}_k + \boldsymbol{G}_k\boldsymbol{\Pi}_{\boldsymbol{\xi},k}. \tag{6.8}$$

In the case in which the controller is static, the sensitivity model is significantly simplified. In fact, for a static control law $\boldsymbol{u}_k = \boldsymbol{\mu}(\boldsymbol{x}_k, \boldsymbol{p}_c)$ no internal state $\boldsymbol{\xi}$ is present, and the corresponding sensitivity $\boldsymbol{\Pi}_{\boldsymbol{\xi},k}$ disappears from the previous equations:

$$\boldsymbol{\Pi}_{k+1} = \left(\boldsymbol{A}_k + \boldsymbol{B}_k\boldsymbol{F}_k\right)\boldsymbol{\Pi}_k + \boldsymbol{M}_k, \tag{6.9}$$

$$\boldsymbol{\Theta}_k = \boldsymbol{F}_k\boldsymbol{\Pi}_k. \tag{6.10}$$

An interesting interpretation can be given to matrix $\boldsymbol{F}_k$. Suppose that the system deviates from the nominal state $\bar{\boldsymbol{x}}_k$ to a perturbed state $\boldsymbol{x}_{\epsilon,k}$. Then, $\boldsymbol{F}_k$ represents the state feedback gains of the first-order Taylor approximation of the control law (6.2) since, for a small enough perturbation, the control action $\boldsymbol{u}_{\epsilon,k} = \boldsymbol{\mu}(\boldsymbol{x}_{\epsilon,k}, \boldsymbol{\xi}_k, \boldsymbol{p}_c)$ can be expanded as

$$\boldsymbol{u}_{\epsilon,k} \simeq \bar{\boldsymbol{u}}_k + \boldsymbol{F}_k(\boldsymbol{x}_{\epsilon,k} - \bar{\boldsymbol{x}}_k).$$

With this in mind, we will refer to $\boldsymbol{F}_k$ as the *(feedback) gains* of the controller.

Additionally, it is worth mentioning that, from (6.6) and (6.8), one can also define the sensitivity of any function of the state and inputs to characterize its local behavior to changes in the parameters. Letting $\boldsymbol{\gamma}(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^{n_\gamma}$ be a generic function, its closed-loop sensitivity to parameters is simply:

$$\boldsymbol{\Gamma}_k = \left.\frac{d\boldsymbol{\gamma}}{d\boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{p}_c} = \frac{\partial\boldsymbol{\gamma}}{\partial\boldsymbol{x}}\boldsymbol{\Pi}_k + \frac{\partial\boldsymbol{\gamma}}{\partial\boldsymbol{u}}\boldsymbol{\Theta}_k. \tag{6.11}$$

6.2 Tubes of perturbed trajectories

One of the possible uses of the closed-loop sensitivities just introduced is to map a set of parameter deviations to a set of perturbed trajectories of the closed-loop system. In particular, we are interested in obtaining an ellipsoidal approximation of the bundle of perturbed trajectories for the states and the inputs — or for any function of these quantities.

Assume a (known) maximum parameter deviation $\Delta\boldsymbol{p}_{\max}$ for each component of the vector $\boldsymbol{p}$, that is, the parameters are supposed to belong to the set

$$\mathcal{P} = \{\boldsymbol{p} \in \mathbb{R}^{n_p} : -\Delta\boldsymbol{p}_{\max} \leq \boldsymbol{p} - \boldsymbol{p}_c \leq \Delta\boldsymbol{p}_{\max}\} \tag{6.12}$$

centered at the nominal value $\boldsymbol{p}_c$, and let $\boldsymbol{W} = \operatorname{diag}(\Delta p_{\max,i}^2)$. To estimate the effect of a parameter deviation on the closed-loop trajectory, we define a mapping from the parameter space to the state space through ellipsoids. Let the parameter ellipsoid with matrix $\boldsymbol{W} \in \mathbb{R}^{n_p \times n_p}$ centered at $\boldsymbol{p}_c$ be

$$\mathcal{E}_{\boldsymbol{p}} = \left\{\boldsymbol{p} \in \mathcal{P} \,:\, \Delta\boldsymbol{p}^T\boldsymbol{W}^{-1}\Delta\boldsymbol{p} \leq 1\right\} \tag{6.13}$$

with $\Delta\boldsymbol{p} = \boldsymbol{p} - \boldsymbol{p}_c$. From a first-order approximation around the nominal trajectory $\bar{\boldsymbol{x}}$, one has

$$\Delta\boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \simeq \boldsymbol{\Pi}_k \Delta\boldsymbol{p}, \tag{6.14}$$

which, assuming that $\boldsymbol{p} \in \mathcal{E}_{\boldsymbol{p}} \subset \mathcal{P}$, can be exploited to obtain the corresponding uncertainty ellipsoid in the state space at a given time t_k. To do this, we pseudo-invert (6.14) to map *feasible* state deviations to parameter deviations:

$$\Delta\boldsymbol{p} = \boldsymbol{\Pi}_k^{\dagger} \Delta\boldsymbol{x}_k. \tag{6.15}$$

We use the term feasible to highlight the fact that in case $n_x > n_p$ — thus system (6.14) is overdetermined — only state deviations which are in the range space of $\boldsymbol{\Pi}_k$ will provide an exact solution. This is however always the case in our treatment since we assume that all state deviations satisfy (6.14).

Applying (6.15) to (6.13), and noting that $\boldsymbol{\Pi}_k^{\dagger T} \boldsymbol{W}^{-1} \boldsymbol{\Pi}_k^{\dagger} = \left(\boldsymbol{\Pi}_k \boldsymbol{W} \boldsymbol{\Pi}_k^T\right)^{\dagger}$, we obtain the uncertainty ellipsoid in state space:

$$\Delta\boldsymbol{x}_k^T \left(\boldsymbol{\Pi}_k \boldsymbol{W} \boldsymbol{\Pi}_k^T\right)^{\dagger} \Delta\boldsymbol{x}_k \leq 1. \tag{6.16}$$

Moreover, the same reasoning applies to the input deviation $\Delta\boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k$ resulting in the input space ellipsoid

$$\Delta\boldsymbol{u}_k^T \left(\boldsymbol{\Theta}_k \boldsymbol{W} \boldsymbol{\Theta}_k^T\right)^{\dagger} \Delta\boldsymbol{u}_k \leq 1. \tag{6.17}$$

Note that both the state and the input ellipsoids are time-varying, as the respective sensitivities follow the evolution given by (6.6) and (6.8). The evolution of these ellipsoids over time is what we will refer to as *state* and *input tubes* in the following. Under assumptions (6.13)–(6.14), the closed-loop trajectories will evolve inside the tubes centered around the nominal trajectory (see Fig. 6.1 for an illustrative example), that is, the state/input at time t_k will belong to their corresponding ellipsoid at t_k. One can then make use of these tubes for several purposes, e.g., for shaping the closed-loop trajectory so that the state/input ellipsoids do not violate any state or input constraint.

In many cases of interest, one needs to determine the maximum deviation along a *particular direction of interest* $\boldsymbol{n}$ in the state/input space (for instance, for evaluating the deviation of a particular component of the state/input). Given an ellipsoid with matrix $\boldsymbol{K}$, the *ellipsoid radius* $\rho \in \mathbb{R}$ along a direction $\boldsymbol{n}$ can be obtained as

$$\rho_{\boldsymbol{n}} = \sqrt{\boldsymbol{n}^T \boldsymbol{K} \boldsymbol{n}}. \tag{6.18}$$

More in detail, consider the ellipsoid $\mathcal{E}_{\boldsymbol{q}}$ centered at $\bar{\boldsymbol{q}} \in \mathbb{R}^{n_q}$ defined as

$$\mathcal{E}_{\boldsymbol{q}} = \left\{ \Delta\boldsymbol{q} \in \mathbb{R}^{n_q} \,:\, \Delta\boldsymbol{q}^T \boldsymbol{K}^{\dagger} \Delta\boldsymbol{q} \leq 1 \right\}, \tag{6.19}$$

where $\Delta\boldsymbol{q} = \boldsymbol{q} - \bar{\boldsymbol{q}}$ and $\boldsymbol{K} \in \mathbb{R}^{n_q \times n_q}$ is a symmetric positive semi-definite matrix.

Given a direction $\boldsymbol{n} \in \mathbb{R}^{n_q}$, we are interested in finding the maximum deviation from the center of the ellipsoid along $\boldsymbol{n}$. More formally, we want to find the smallest scalar $\alpha > 0$ such that

$$\forall \Delta\boldsymbol{q} \in \mathcal{E}_{\boldsymbol{q}} \quad \boldsymbol{n}^T \Delta\boldsymbol{q} \leq \alpha.$$

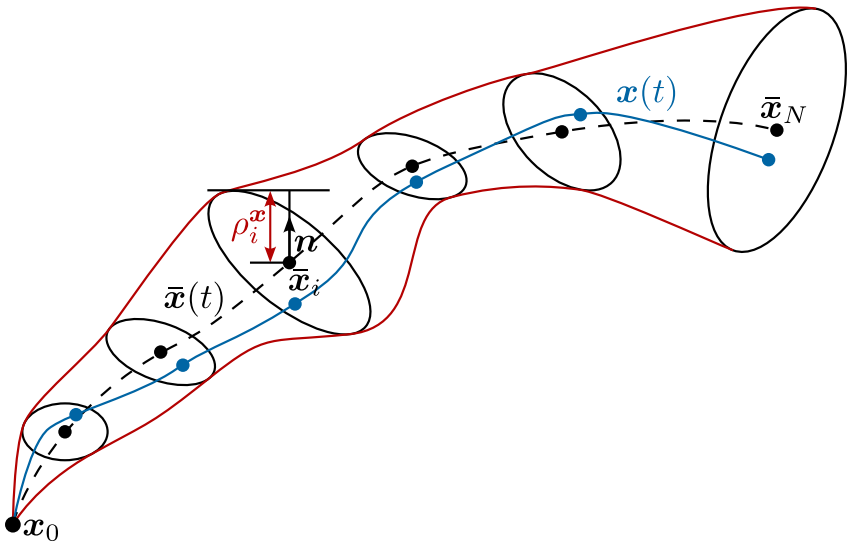

Figure 6.1. Qualitative visualization of a perturbed state trajectory $\boldsymbol{x}(t)$ (blue) and the ellipsoidal tubes (red) centered around the nominal trajectory $\bar{\boldsymbol{x}}(t)$. Note how, in general, the tube radius does not necessarily monotonically increase thanks to the effect of the feedback action, and despite the cumulative effect of uncertainties over time.

Note that, being $\mathcal{E}_{\boldsymbol{q}}$ in (6.19) a convex set, we can focus our search on its boundary region $\partial\mathcal{E}_{\boldsymbol{q}}$.

Finding α can be rephrased as solving the problem:

$$\begin{cases} \underset{\alpha,\Delta\boldsymbol{q}}{\text{minimize}} & \alpha \\ \text{subject to} & \boldsymbol{n}^T\Delta\boldsymbol{q} = \alpha \\ & \Delta\boldsymbol{q} \in \partial\mathcal{E}_{\boldsymbol{q}}, \end{cases} \tag{6.20}$$

that admits the solution

$$\alpha^* = \sqrt{\boldsymbol{n}^T\boldsymbol{K}\boldsymbol{n}},$$
$$\Delta\boldsymbol{q}^* = \frac{\boldsymbol{K}\boldsymbol{n}}{\sqrt{\boldsymbol{n}^T\boldsymbol{K}\boldsymbol{n}}}.$$

Proof. One can solve (6.20) by finding a saddle point of its Lagrangian

$$\mathcal{L}(\alpha, \Delta\boldsymbol{q}, \boldsymbol{\lambda}) = \alpha + \lambda_1(\boldsymbol{n}^T\Delta\boldsymbol{q} - \alpha) + \lambda_2(\Delta\boldsymbol{q}^T\boldsymbol{K}^\dagger\Delta\boldsymbol{q} - 1),$$

which entails solving the system of equations $\nabla\mathcal{L} = \boldsymbol{0}$:

$$1 - \lambda_1^* = 0, \tag{6.21a}$$
$$\lambda_1^*\boldsymbol{n}^T + 2\lambda_2^*(\Delta\boldsymbol{q}^*)^T\boldsymbol{K}^\dagger = \boldsymbol{0}, \tag{6.21b}$$
$$\boldsymbol{n}^T\Delta\boldsymbol{q}^* - \alpha^* = 0, \tag{6.21c}$$
$$(\Delta\boldsymbol{q}^*)^T\boldsymbol{K}^\dagger\Delta\boldsymbol{q}^* - 1 = 0. \tag{6.21d}$$

From (6.21a), it is trivial to find $\lambda_1^* = 1$. By substituting (6.21b) into (6.21c) one obtains

$$\alpha^* = \boldsymbol{n}^T\Delta\boldsymbol{q}^* = -2\lambda_2^*(\Delta\boldsymbol{q}^*)^T\boldsymbol{K}^\dagger\Delta\boldsymbol{q}^*.$$

Noting that $(\Delta \boldsymbol{q}^*)^T \boldsymbol{K}^\dagger \Delta \boldsymbol{q}^* = 1$, we obtain

$$\lambda_2^* = -\frac{\alpha^*}{2}. \tag{6.22}$$

We can then substitute (6.22) back into (6.21b) and right-multiply by $\boldsymbol{K}$ to obtain

$$\Delta \boldsymbol{q}^* = \frac{\boldsymbol{K}\boldsymbol{n}}{\alpha^*},$$

which, plugged into (6.21d), finally yields

$$\alpha^* = \sqrt{\boldsymbol{n}^T \boldsymbol{K} \boldsymbol{n}},$$

corresponding to the radius of the ellipsoid in the desired direction. ∎

Considering a set of directions of interest, one can then compute the radius along each direction for obtaining a vector of radii $\boldsymbol{\rho}$ which can be used as a measure of the worst-case deviation of the system along these directions. For illustration let us consider w.l.o.g. the case of input tubes. Once the evolution of the input sensitivity $\boldsymbol{\Theta}$ has been obtained, one can compute the input ellipsoid matrix $\boldsymbol{K}_k^{\boldsymbol{u}} = \boldsymbol{\Theta}_k \boldsymbol{W} \boldsymbol{\Theta}_k^T$. Then, using (6.18) with $\boldsymbol{n}$ spanning the canonical basis of the input space $\mathbb{R}^{n_u}$, it is possible to obtain

$$\rho_{j,k}^{\boldsymbol{u}} = \sqrt{K_{jj,k}^{\boldsymbol{u}}}, \quad \forall j \in \mathbb{I}_1^{n_u}, \quad \forall k \in \mathbb{I}_0^{C-1}. \tag{6.23}$$

The quantities $\rho_{j,k}^{\boldsymbol{u}}$, collected in vector $\boldsymbol{\rho}_k^{\boldsymbol{u}}$, are the radii of the input ellipsoid along each axis j and represent, therefore, the worst-case deviations of each input component j w.r.t. its nominal value due to parametric uncertainty. Note that at the start of the trajectory $\boldsymbol{\rho}_0^{\boldsymbol{u}} = \boldsymbol{0}$ by construction, since $\boldsymbol{\Pi}_0 = \boldsymbol{0}$ and, therefore, $\boldsymbol{\Theta}_0 = \boldsymbol{0}$.

The same reasoning can clearly be applied to the states to find the associated radii from the state sensitivity (6.6) and, using (6.11), to any function of the state and inputs to find its worst-case deviation along the nominal trajectory.

6.3 Robust trajectory planning for a Quadrotor

This section presents an experimental validation of the concepts of *closed-loop state and input sensitivity* in the context of robust flight control for a Quadrotor (UAV) equipped with the popular PX4[1] autopilot. The objective is to experimentally assess how the optimization of the reference trajectory with respect to these sensitivity metrics can improve the closed-loop system performance against model uncertainties commonly affecting the Quadrotor systems. To accomplish this, we present a series of experiments designed to validate our optimization approach on two distinct trajectories, with the primary aim of assessing its precision in guiding the Quadrotor through the center of a window at relatively high speeds. This approach provides some interesting insights for increasing the closed-loop robustness of the robot state and inputs against physical parametric uncertainties that may degrade the system's performance. See Fig. 6.2 for an example of the results.

[1] `https://px4.io/`

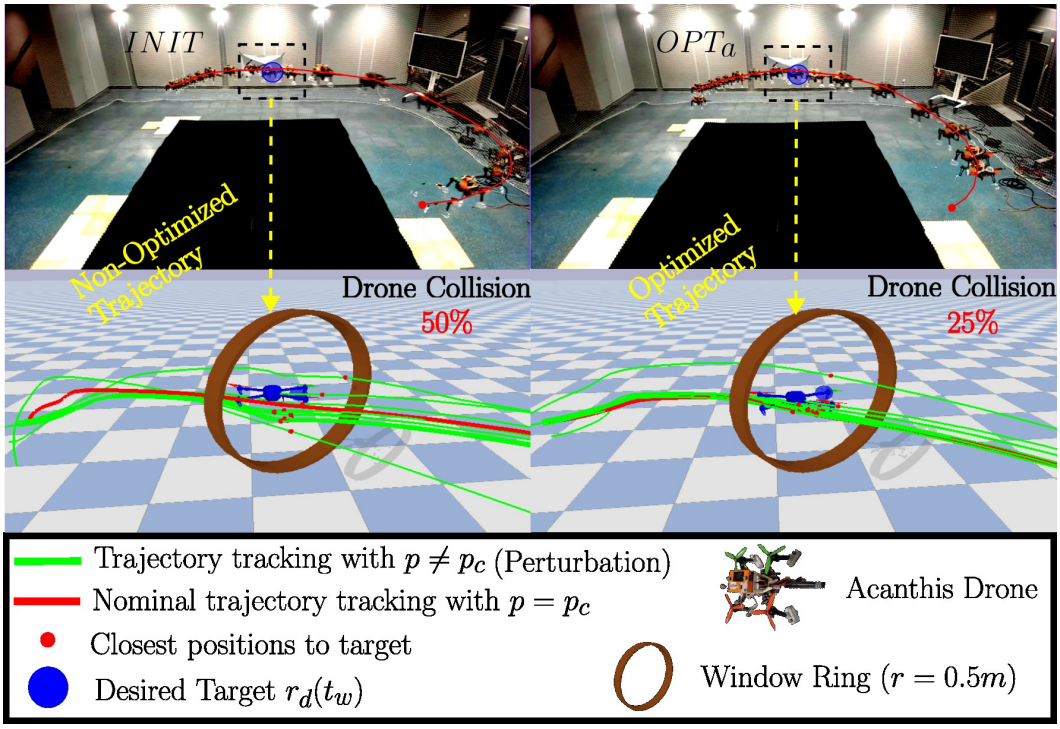

Figure 6.2. Drone trajectory tracking under parametric uncertainties (green) to pass through a target $s_d(t_w)$ (blue sphere). Non-optimized ($INIT$, left) and optimized (OPT_a, right) trajectories are compared. The closest positions reached to the desired target after each experiment (small red sphere) is visualized in PyBullet where the drone passes through a circular window at relatively high speed. A video of the experiments is available at https://youtu.be/QFnrQ_O2BiU.

The employed Quadrotor is controlled by the widely used open-source autopilot PX4. While PX4 is used by numerous research groups, companies, and enthusiasts for its valuable features, there are several instances where its control performance can fall short of meeting user requirements. Users often attempt to enhance control by customizing the autopilot, sometimes opting for a model-based approach. However, such customization is not straightforward and demands a strong grasp of firmware architecture and proficient programming skills. Having this in mind, our objective is to employ the *closed-loop state/input sensitivity* metric (and derived quantities) on a drone controlled by PX4 default controller to assess how proper reference trajectory planning, designed to be robust against parametric uncertainties, can enhance the performance of a standard drone.

In the following, we first review the dynamic model of a Quadrotor, we then illustrate the equations of the PX4 controller, and present the proposed optimization problem. Finally, we report simulation and experimental results.

6.3.1 Quadrotor model

To account for the typical perturbations that are present in real-world systems, we consider a 3D Quadrotor model with displaced center of mass [117]. Let $\mathcal{F}_{\mathcal{W}} =$

Table 6.1. Quadrotor model parameters

Symbol	Value
m_r	1.535 kg
$\boldsymbol{g}_C$	$(0, 0, 0)$ m
(I_{xx}, I_{yy}, I_{zz})	$(29.125, 29.125, 55.225) \cdot 10^{-3}$ kg·m^2
k_f	$5.86 \cdot 10^{-6}$ N/(rad/s)^2
k_m	0.06
l	0.28 m
β	$\pi/3$ rad

$\{\boldsymbol{O}_\mathcal{W}, \boldsymbol{x}_\mathcal{W}, \boldsymbol{y}_\mathcal{W}, \boldsymbol{z}_\mathcal{W}\}$ denote the world inertial frame, and $\boldsymbol{\mathcal{F}}_\mathcal{B} = \{\boldsymbol{O}_\mathcal{B}, \boldsymbol{x}_\mathcal{B}, \boldsymbol{y}_\mathcal{B}, \boldsymbol{z}_\mathcal{B}\}$ represent the body frame attached to the Quadrotor geometric center. We define: $\boldsymbol{r} = (x, y, z) \in \mathbb{R}^3$ as the drone position in $\boldsymbol{\mathcal{F}}_\mathcal{W}$, $\boldsymbol{v} = (v_x, v_y, v_z) \in \mathbb{R}^3$ as its linear velocity in $\boldsymbol{\mathcal{F}}_\mathcal{W}$, $\boldsymbol{q} = (q_w, q_x, q_y, q_z) \in \mathbb{S}^3$ as the unit-norm quaternion representing the orientation of $\boldsymbol{\mathcal{F}}_\mathcal{B}$ relative to $\boldsymbol{\mathcal{F}}_\mathcal{W}$, and $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ as the angular velocity of $\boldsymbol{\mathcal{F}}_\mathcal{B}$ with respect to $\boldsymbol{\mathcal{F}}_\mathcal{W}$, expressed in $\boldsymbol{\mathcal{F}}_\mathcal{B}$. The Quadrotor state vector is then $\boldsymbol{x} = (\boldsymbol{r}, \boldsymbol{v}, \boldsymbol{q}, \boldsymbol{\omega}) \in \mathbb{R}^6 \times \mathbb{S}^3 \times \mathbb{R}^3$.

Let w_i^2 be the squared speed of the i-th propeller and define the Quadrotor control input as $\boldsymbol{u} = (w_1^2, \ldots, w_4^2)$. An allocation matrix $\boldsymbol{T}$ is used to relate the inputs $\boldsymbol{u}$ to the thrust/torques $(f, \boldsymbol{\tau})$ in body frame:

$$\begin{pmatrix} f \\ \boldsymbol{\tau} \end{pmatrix} = \boldsymbol{T}\boldsymbol{u} = k_f \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & l\sin\beta & 0 & -l\sin\beta \\ -l\cos\beta & 0 & l\cos\beta & 0 \\ k_m & -k_m & k_m & -k_m \end{pmatrix} \boldsymbol{u}, \tag{6.24}$$

where l is the length of the Quadrotor arms, k_f and k_m are the thrust and the drag aerodynamic coefficients of the propellers [117, 118], and β is the angle determining the equivalent arm length. We assume that a payload of mass m_p might be attached to the Quadrotor with mass m_r at a position $\boldsymbol{d}_p = (0, 0, d_z)$ in $\boldsymbol{\mathcal{F}}_\mathcal{B}$, with the total mass then being $m = m_r + m_p$. We denote with $\mathbf{J} = \mathrm{diag}(I_{xx} + m_p d_z^2, I_{yy} + m_p d_z^2, I_{zz}) \in \mathbb{R}^{3\times3}$ the Quadrotor body frame inertia matrix about the geometric center $\boldsymbol{O}_\mathcal{B}$. We finally consider that the Quadrotor center of mass $\boldsymbol{G}_\mathcal{B}$ is displaced w.r.t. the geometric center $\boldsymbol{O}_\mathcal{B}$ by a displacement denoted as $\boldsymbol{g}_C = (g_x, g_y, g_z)$ in $\boldsymbol{\mathcal{F}}_\mathcal{B}$. From [117], the total force $\boldsymbol{f}_\mathrm{tot}$ acting on the Quadrotor in $\boldsymbol{\mathcal{F}}_\mathcal{B}$ includes the propeller total thrust f, gravitational effects, and an additional fictitious force due to the displaced $\boldsymbol{g}_C$:

$$\boldsymbol{f}_\mathrm{tot} = f\boldsymbol{z}_W - mg\mathbf{R}(\boldsymbol{q})^T\boldsymbol{z}_W - m[\boldsymbol{\omega}]_\times[\boldsymbol{\omega}]_\times\boldsymbol{g}_C.$$

For the total torque $\boldsymbol{\tau}_\mathrm{tot}$ (expressed in $\boldsymbol{\mathcal{F}}_\mathcal{B}$), one has:

$$\boldsymbol{\tau}_\mathrm{tot} = \boldsymbol{\tau} - mg[\mathbf{g}_C]_\times\mathbf{R}(\boldsymbol{q})^T\boldsymbol{z}_W - [\boldsymbol{\omega}]_\times\mathbf{J}\boldsymbol{\omega},$$

where $\boldsymbol{\tau}$ represents the propeller torque from (6.24) and $\mathbf{R}(\boldsymbol{q}) \in SO(3)$ is the rotation matrix associated to the quaternion $\boldsymbol{q}$. By defining the spatial inertia

matrix as

$$\boldsymbol{M} = \begin{pmatrix} m\boldsymbol{I_3} & -m[\mathbf{g}_C]_\times \\ m[\mathbf{g}_C]_\times & \mathbf{J} \end{pmatrix}$$

one can finally obtain the following relation

$$\begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\alpha} \end{pmatrix} = \boldsymbol{M}^{-1} \begin{pmatrix} \boldsymbol{f}_{\text{tot}} \\ \boldsymbol{\tau}_{\text{tot}} \end{pmatrix}$$

from which the Quadrotor dynamic model with displaced center of mass can be obtained

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) = \begin{cases} \dot{\boldsymbol{r}} = \boldsymbol{v} \\ \dot{\boldsymbol{v}} = \mathbf{R}(\boldsymbol{q})\boldsymbol{\nu}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \\ \dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{pmatrix} 0 \\ \boldsymbol{\omega} \end{pmatrix} \\ \dot{\boldsymbol{\omega}} = \boldsymbol{\alpha}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}). \end{cases} \tag{6.25}$$

The continuous-time dynamics (6.25) is then discretized using a fixed-step 4th order Runge-Kutta integration to obtain the discrete-time model. The model used in simulation is based on the 3DR Iris drone[2], whose nominal parameters are reported in Tab. 6.1[3].

6.3.2 PX4 controller

It is well known that the position $\boldsymbol{r}$ and the yaw angle ψ are flat outputs for the Quadrotor model (6.25), as explained in, e.g., [119]. Based on this fact, the PX4 controller is designed to track a desired position $\boldsymbol{r_d}(t)$ and a yaw angle $\psi_d(t)$ trajectories by generating suitable propeller speeds $\boldsymbol{u}$. More in detail, the controller has a cascaded structure consisting of four stages (as illustrated in Fig. 6.3):

1. the *position controller* in which the desired position $\boldsymbol{r_d}$ is first transformed into an equivalent acceleration setpoint $\boldsymbol{a}_{\text{sp}}$,
2. the *attitude generator* responsible for producing both a combined thrust f and an attitude setpoint $\boldsymbol{q}_{\text{sp}}$ derived from the desired yaw angle $\boldsymbol{\psi_d}$ and the previously computed acceleration setpoint,
3. the *attitude controller* determining the desired body torques and, finally,
4. the *control allocation* where thrust and body torques are transformed in propeller speeds via the inverse of the nominal allocation matrix (6.24).

Following the available documentation and open-source code[4], is is possible to derive an equivalent continuous-time controller to be integrated in the sensitivity-aware planning scheme. Note that in the PX4 controller, the inertial and the body frames

[2] `https://www.arducopter.co.uk/iris-quadcopter-uav.html`

[3] These parameters follow the ones reported in the PX4 Gazebo package `https://github.com/PX4/PX4-SITL_gazebo-classic`

[4] `https://github.com/PX4/PX4-Autopilot`

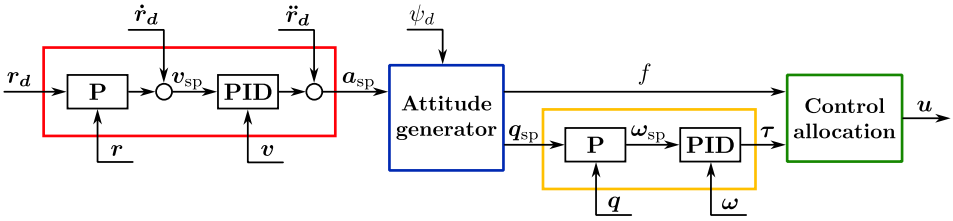

Figure 6.3. Block diagram of the PX4 controller, composed of four stages: the position controller (red), the attitude generator (blue), the attitude controller (yellow), and the control allocation (green).

are expressed using the North-East-Down (NED) convention, so (simple) conversions are needed if one chooses to express the dynamic model in different frames. The position controller in the PX4 consists of a P and a PID action on the position and velocity errors, respectively. The latter introduces two 3-dimensional internal states $\boldsymbol{\xi}_v$ and $\boldsymbol{\xi}_a$ to realize the integral and the low-pass-filtered derivative actions. Denoting with Ω_p the low-pass filter cutoff pulsation and with $\boldsymbol{P}_r$, $\boldsymbol{P}_v$, $\boldsymbol{I}_v$, $\boldsymbol{D}_v$ the diagonal matrix gains, the position controller equations can be expressed as

$$\begin{aligned}
\dot{\boldsymbol{\xi}}_v &= \dot{\boldsymbol{r}}_{\boldsymbol{d}} - \boldsymbol{v} \\
\dot{\boldsymbol{\xi}}_a &= -\Omega_p \boldsymbol{\xi}_a - \Omega_p^2 \boldsymbol{v} \\
\boldsymbol{a}_{\text{sp}} &= \ddot{\boldsymbol{r}}_{\boldsymbol{d}} + \boldsymbol{P}_v(\dot{\boldsymbol{r}}_{\boldsymbol{d}} - \boldsymbol{v} + \boldsymbol{P}_r(\boldsymbol{r}_{\boldsymbol{d}} - \boldsymbol{r})) - \boldsymbol{D}_v(\boldsymbol{\xi}_a + \Omega_p \boldsymbol{v}) + \boldsymbol{I}_v \boldsymbol{\xi}_v.
\end{aligned}$$

The attitude generator receives the acceleration setpoint $\boldsymbol{a}_{\text{sp}} = (\ddot{x}_{\text{sp}}, \ddot{y}_{\text{sp}}, \ddot{z}_{\text{sp}})$ and computes the collective thrust f. Let $\boldsymbol{b}_z = \frac{(-\ddot{x}_{\text{sp}}, -\ddot{y}_{\text{sp}}, g)}{\|(-\ddot{x}_{\text{sp}}, -\ddot{y}_{\text{sp}}, g)\|}$ and $\boldsymbol{t}_{\text{sp}} = \frac{\boldsymbol{b}_z}{b_{z,z}} \left(\ddot{z}_{\text{sp}} \frac{h_t}{g} - h_t \right)$, where $h_t \in (0, 1)$ denotes the nominal normalized hovering thrust so that the computed thrust is $f = -\|\boldsymbol{t}_{\text{sp}}\|$. The attitude setpoint $\boldsymbol{q}_d$ is then computed by aligning the z-axis of the body frame to the desired thrust $\boldsymbol{t}_{\text{sp}}$ and rotating of an angle ψ_d around such axis [120]. To generate the body torques, let the quaternion error vector be $\boldsymbol{q}_{\text{e}} = \boldsymbol{q}^{-1} \otimes \boldsymbol{q}_{\text{sp}}$. The angular rate setpoint is then computed proportionally to the error as $\boldsymbol{\omega}_{\text{sp}} = 2\boldsymbol{P}_q \text{sign}(q_{\text{e},w}) \boldsymbol{q}_{\text{e},\mathbf{v}}$, with $\boldsymbol{P}_q$ denoting the proportional gains. Finally, the equations of the PID controller tracking the attitude rate $\boldsymbol{\omega}_{\text{sp}}$ are

$$\begin{aligned}
\dot{\boldsymbol{\xi}}_\omega &= \boldsymbol{\omega}_{\text{sp}} - \boldsymbol{\omega} \\
\dot{\boldsymbol{\xi}}_\alpha &= -\Omega_a \boldsymbol{\xi}_\alpha - \Omega_a^2 \boldsymbol{\omega} \\
\boldsymbol{\tau} &= \boldsymbol{P}_\omega(\boldsymbol{\omega}_{\text{sp}} - \boldsymbol{\omega}) - \boldsymbol{D}_\omega(\boldsymbol{\xi}_\alpha + \Omega_a \boldsymbol{\omega}) + \boldsymbol{I}_\omega \boldsymbol{\xi}_\omega,
\end{aligned}$$

where $\boldsymbol{\xi}_\omega$ and $\boldsymbol{\xi}_\alpha$ are the 3-dimensional internal states of the controller, Ω_a is the cutoff pulsation of the low-pass filter, and $\boldsymbol{P}_\omega$, $\boldsymbol{I}_\omega$, $\boldsymbol{D}_\omega$ are the diagonal matrix gains. Lastly, the control input $\boldsymbol{u}$ is reconstructed from the allocation matrix as

$$\boldsymbol{u} = \boldsymbol{T}^{-1} \begin{pmatrix} f \\ \boldsymbol{\tau} \end{pmatrix}$$

where, of course, the matrix $\boldsymbol{T}$ from (6.24) is evaluated on the *nominal* values of the parameters $\boldsymbol{p}_c$ (which may differ from the actual $\boldsymbol{p}$ because of inaccuracies in the Quadrotor model).

6.3.3 Problem formulation

Letting $\boldsymbol{s}(\boldsymbol{x}) \in \mathbb{R}^{n_s}$ be a quantity of interest (e.g., the position of the robot center), we consider the task of tracking a desired trajectory $\boldsymbol{s}_d(\boldsymbol{a}, t)$ defined for $t \in [t_0, t_f]$ and function of a finite set of trajectory parameters $\boldsymbol{a} \in \mathbb{R}^{n_a}$. The tracking task is realized by a controller having the generic form (6.2), such as the PX4 controller of the previous section. Note that the controller is implicitly a function of the trajectory parameters $\boldsymbol{a}$ as well, as they define the desired trajectory that the controller is designed to track.

The state and input space ellipsoids can be exploited for defining a "weighted sensitivity norm" by considering the eigenvalues λ_i of the ellipsoid matrix $\boldsymbol{\Pi}_k \boldsymbol{W} \boldsymbol{\Pi}_k^T$ in (6.16). In particular we consider the following matrix norm

$$\|\boldsymbol{\Pi}_k\|_{\boldsymbol{W}} = \max(\lambda_i(\boldsymbol{\Pi}_k \boldsymbol{W} \boldsymbol{\Pi}_k^T)) \tag{6.26}$$

which represents the largest (worst-case) deviation of the state $\boldsymbol{x}$ assuming a parametric uncertainty as in (6.13). Alternative choices for the matrix norm to be used include, for instance, the Frobenious norm [121], that does not require to perform the eigenvalue decomposition and can therefore be more computationally efficient. The effect that a particular choice of norm has on the solution is indeed a topic that requires further study. Nonetheless, norm (6.26) provides a reasonable choice by directly accounting for the worst-case deviation — a useful characterization in the context of robust control. Furthermore, one can also exploit (6.16)–(6.17) and (6.23) for obtaining the tubes of perturbed trajectories for the individual components of the states and the inputs. Considering all the canonical directions in the input space, one can obtain the tube radius $\boldsymbol{\rho}^{\boldsymbol{u}}(t)$ such that

$$-\boldsymbol{\rho}^{\boldsymbol{u}}(t) \leq \boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t) \leq \boldsymbol{\rho}^{\boldsymbol{u}}(t). \tag{6.27}$$

Equation (6.27) bounds from above/below the envelope of perturbed inputs when the parameter uncertainty is bounded as in (6.13), and an analogous upper/lower bound can also be obtained for the generic state component $x_i(t)$.

In this example, we consider the following trajectory optimization problem:

$$\begin{cases} \underset{\boldsymbol{a}}{\text{minimize}} & \|\boldsymbol{\Pi}(t_w)\|_{\boldsymbol{W}} \\ \text{subject to} & \boldsymbol{M}\boldsymbol{a} = \boldsymbol{b} \\ & \boldsymbol{u}_{\min} + \boldsymbol{\rho}^{\boldsymbol{u}}(t) \leq \bar{\boldsymbol{u}}(t) \leq \boldsymbol{u}_{\max} - \boldsymbol{\rho}^{\boldsymbol{u}}(t) \quad \forall t \in [t_0, t_f]. \end{cases} \tag{6.28}$$

We seek the optimal value $\boldsymbol{a}^*$ of the shape parameter $\boldsymbol{a}$ of the reference trajectory $\boldsymbol{s}_d(\boldsymbol{a}, t)$ for minimizing the weighted norm (6.26) at a specific time t_w (e.g., for passing through a desired point like the center of a window with increased accuracy). Minimization of this cost will increase the robustness of the closed-loop system against parameter uncertainties at t_w. The constraints consist of the boundary conditions, such as some given initial/final conditions, for $\boldsymbol{s}_d(\boldsymbol{a}, t)$, represented by the linear constraints $\boldsymbol{M}\boldsymbol{a} = \boldsymbol{b}$, and constraints that bound the envelope of perturbed inputs within actuation limits $[\boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}]$, according to (6.27), ensuring that the tracking of the optimized reference trajectory will be feasible for any value of the uncertain parameters $\boldsymbol{p}$ in (6.13). Other objective functions can be considered such

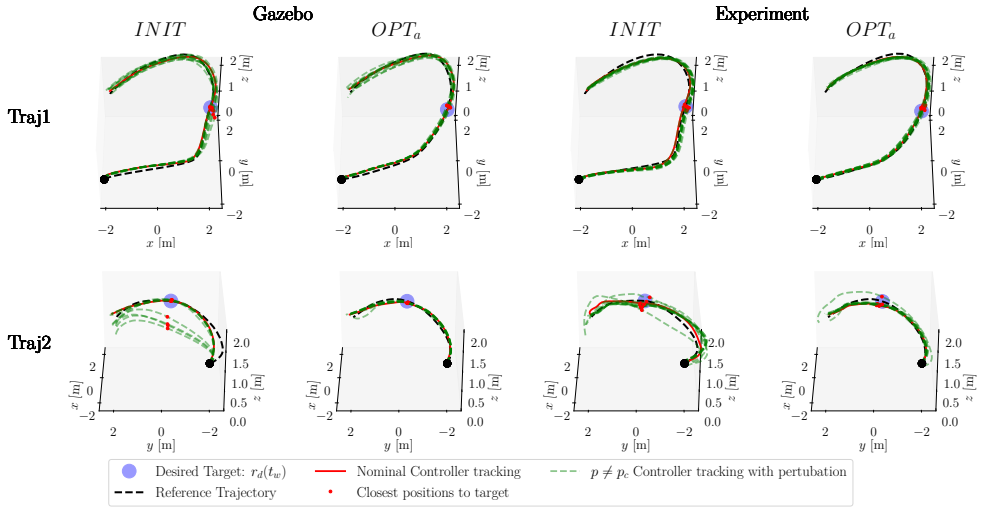

Figure 6.4. Trajectory tracking results by PX4 controller for two trajectory types (1st row: Traj1, 2nd row: Traj2) in simulation (first two columns) and experiments (last two columns). All trajectories start from the black sphere at $(-2, -2, 1)$. Cases include $INIT$ and OPT_a. Perturbed runs consist of N_p green trajectories, while the nominal trajectory tracking (i.e. when $\boldsymbol{p} = \boldsymbol{p}_c$) is denoted in red, all sharing an origin which is the black sphere. Parameters $\boldsymbol{p}$ were randomly drawn from (6.13) in Gazebo simulation where it was implemented by modifying the Iris URDF model. However, the experiments were carried out as in (Fig: 6.8) where actual modifications were done on the drone that resemble the perturbations that will affect parameters $\boldsymbol{p}$. Red spheres at $t = t_w$ mark closest points to desired target $\boldsymbol{r}_d(t_w)$, forming a point cloud around it.

as the integral of the sensitivity norm along the trajectory to enhance tracking accuracy during motion. Furthermore, one can easily include extra constraints like obstacle avoidance by exploiting the tubes on the states, as shown in [114].

6.3.4 Results

A series of experiments and simulations were conducted to assess the effectiveness of the proposed trajectory generation obtained by solving (6.28). The parameters considered uncertain are then the set[5] $\boldsymbol{p} = (k_f, g_x, g_y, g_z, m_r) \in \mathbb{R}^5$. We consider no additional payload attached to the robot, thus $m_p = 0$. Two distinct trajectories $\boldsymbol{s}_d(\boldsymbol{a}, t) = (\boldsymbol{r}_d(\boldsymbol{a}, t), \psi_d(\boldsymbol{a}, t))$, denoted as Traj1 and Traj2 in the following, were used to guide the drone through a circular window at a relatively high speed ($2.0 \leq \|\boldsymbol{v}(t_w)\| \leq 2.5$ m/s) (see Fig. 6.4) while complying with the initial/final state constraints (rest-to-rest motions) and input saturations as in (6.28). Note that the actual window was not introduced during the real experiments for practicality, but it was used in PyBullet[6] with the real flight data for visualization.

We start by generating a first trajectory — that will be referred to as $INIT$ in the following — constructed using piecewise Bezier curves, as discussed in [111,112,

[5] We did not include k_m since uncertainties in this parameter have a negligible impact compared to the other parameters as shown in, e.g., [115].

[6] `https://pybullet.org/`

115, 122], by performing a preliminary trajectory optimization that minimizes the *snap* of $\boldsymbol{r}_d(\boldsymbol{a}, t)$ over the whole time interval with the aim of obtaining a smooth trajectory with minimal curvature changes and satisfying nominal constraints over the state and inputs. Thus, all initial non-optimized trajectories $INIT$ use the minimum snap cost function defined as

$$\begin{cases} \underset{\boldsymbol{a}}{\text{minimize}} & \displaystyle\int_{t_0}^{t_f} \left\| \frac{d^4 \boldsymbol{r}_d(\boldsymbol{a}, \tau)}{dt^4} \right\|^2 d\tau \\ \text{subject to} & \boldsymbol{M}\boldsymbol{a} = \boldsymbol{b} \\ & \boldsymbol{u}_{\min} \leq \bar{\boldsymbol{u}}(t) \leq \boldsymbol{u}_{\max} \quad \forall t \in [t_0, t_f]. \end{cases}$$

Our framework then modifies this trajectory by solving (6.28) with $INIT$ as an initial guess. The resulting trajectory is denoted as OPT_a.

The main goal is to evaluate how close the drone gets to a specific target location, labeled as $\boldsymbol{r}_d(t_w)$, which is right at the center of a predefined window. We want to find out if the drone can safely go through the window even in the presence of uncertainties in its parameters.

The two trajectories are chosen to test the system over different operating conditions: Traj1, having a duration of 7 seconds, requires a relatively low initial acceleration to reach the target; conversely, Traj2, which has a duration of 5 seconds, requires a more significant initial acceleration to reach the target. Both trajectories ensure a minimum speed of 2 m/s when they reach the target.

The framework is implemented in Python and utilizes the COBYLA [123] nonlinear optimizer from the `nlopt` toolbox by employing the symbolic toolbox for symbolic system representation and making use of the Jitcode [124] framework for just-in-time compilation of ordinary differential equations. Within this framework, optimizing trajectories with the PX4 controller typically requires 4-6 minutes per trajectory.

Simulation results

After obtaining the trajectories, dynamic simulations have been carried out in Gazebo thanks to the software in the loop (SITL) feature of PX4, which allows to simulate the autopilot and communicate through a ROS 2 application. This application reads the desired trajectory and sends the series of setpoints (position, velocity, and acceleration) to the drone at a frequency of 50 Hz. Starting from the default model of the Iris Quadrotor, we performed a simulation running with the nominal parameters. Then, we carried out $N_p = 11$ simulations using uncertain parameters $\boldsymbol{p}$ generated by uniformly sampling the inner volume of the ellipsoid (6.13). This is done for Traj1 and Traj2 and both $INIT$ and OPT_a cases. The uncertainty intervals $\Delta p_{\max,i}$ for the matrix $\boldsymbol{W}$ are chosen relative to the nominal parameters $\boldsymbol{p}_c$ as $\Delta k_f = 0.1k_f$, $\Delta g_{x,y} = 0.1l$, $\Delta g_z = 0.1h$, and $\Delta m_r = 0.1m_r$, where h is the distance between the top and bottom plates of the drone chassis. Thus, the optimization problem (6.28) will aim at minimizing the sensitivity of the Quadrotor position $\boldsymbol{r}(t_w)$ at *the instant of passing through the window center* t_w against variations in the parameters k_f, g_x, g_y, g_z and m_r. In our initial assessment, as depicted in Fig. 6.4, we present the results for the N_p "perturbed runs" for both

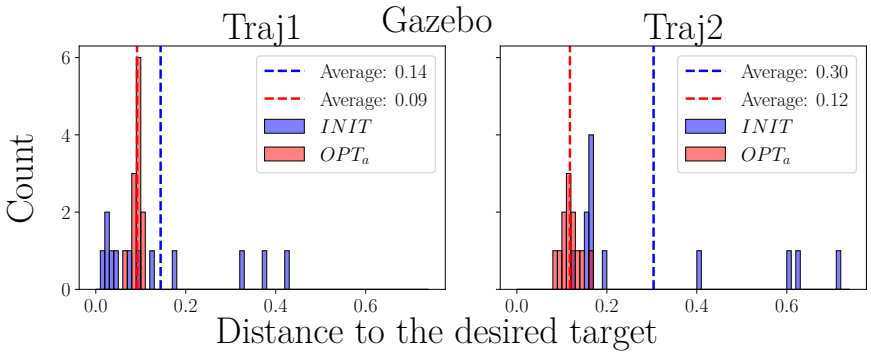

Figure 6.5. Histograms of two trajectory types (left: Traj1, right: Traj2) for both cases ($INIT$ in blue, OPT_a in red) showing the distances to the desired target ($\|\boldsymbol{r}_d(t_w) - \boldsymbol{r}(t_w)\|$) and it's average across N_p runs in Gazebo.

Traj1 and Traj2 (1st and 2nd row, respectively) under the Gazebo label (first two columns). The plots report the different targets that reach $\boldsymbol{r}(t_w)$ (red dots) that are the closest to $\boldsymbol{r}_d(t_w)$ (window center) for each of the perturbed simulations (in green). The result is a point cloud around the desired $\boldsymbol{r}_d(t_w)$. It is worth noticing that, when there is no parameter uncertainty affecting the system (i.e. when $\boldsymbol{p} = \boldsymbol{p}_c$) at nominal tracking (in red), the drone successfully traverses the window with an accuracy below 0.12 m for all the trajectories and in both the optimized and the non-optimized cases. However, the outcomes differ significantly for the perturbed simulations. Specifically, the non-optimized $INIT$ case displays greater deviations from the desired target location $\boldsymbol{r}_d(t_w)$, when compared to the optimized OPT_a case, as expected. This difference is particularly pronounced in the 2nd trajectory, which is more challenging (due to its higher accelerations).

By referring to Fig. 6.5, which displays the distance to the desired target for both trajectories, we observe that the $INIT$ case shows a larger average deviation of 0.14 m for Traj1 and 0.3 m for Traj2, in contrast to the OPT_a cases with averages of 0.09 m for Traj1 and 0.12 m for Traj2. Furthermore, the $INIT$ case exhibits a significantly larger variance, with certain samples deviating 0.42 m in Traj1 and 0.75 m in Traj2. In contrast, OPT_a samples tend to cluster around the average. These findings emphasize the effectiveness of the optimization problem presented in (6.28), which can capture how variations in the parameters affect deviations in the considered states.

Experimental results

After the validation of the optimization problem (6.28) in simulation, we performed real experiments using the Acanthis drone (see Fig. 6.6). Acanthis is an experimental drone platform developed and maintained at Inria Rennes that utilizes the PX4 firmware. The $N_p = 11$ perturbed experiments were performed by physically altering the system as depicted in Fig. 6.8 and were obtained by adding some tools and weights on the extension rod of the Acanthis drone. This allowed us to modify the set of parameters $\boldsymbol{p} = (k_f, g_x, g_y, g_z, m_r)$ at each experiment. In particular, the parameter k_f is affected mainly by the tools that we added under the motors that obstruct the airflow, while the other parameters are affected by the weights

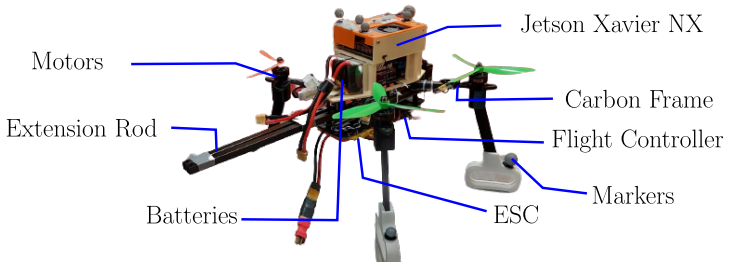

Figure 6.6. Image of the prototype drone Acanthis.

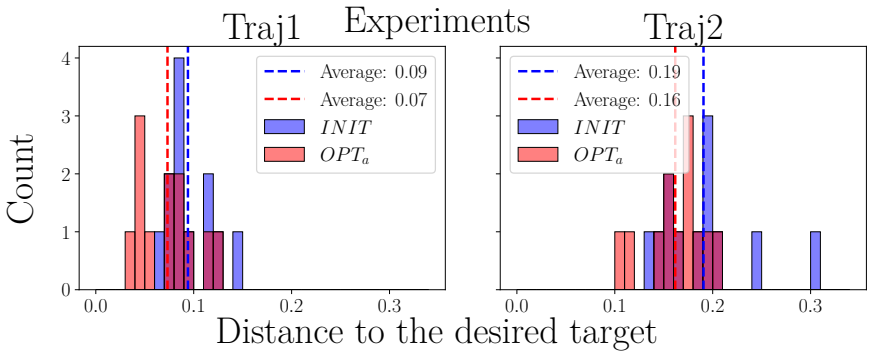

Figure 6.7. Histograms of two trajectory types (left: Traj1, right: Traj2) for both cases ($INIT$ in blue, OPT_a in red) showing the distances to the desired target ($\|\boldsymbol{r}_d(t_w) - \boldsymbol{r}(t_w)\|$) and its average across N_p experiments with Acanthis drone.

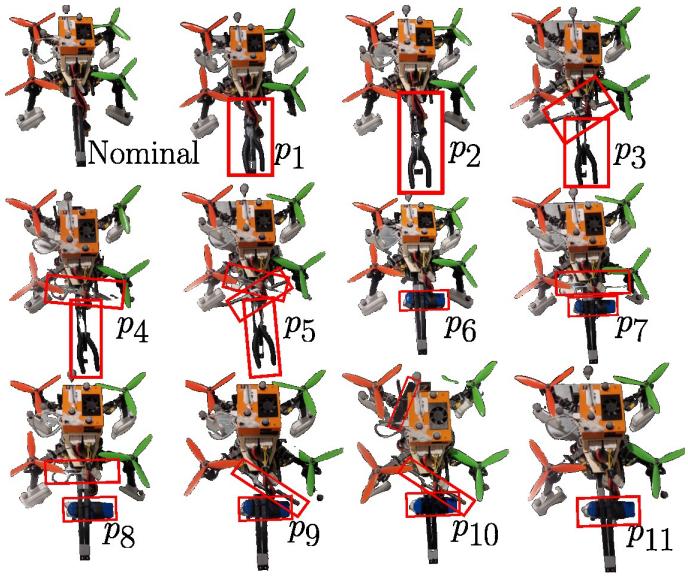

Figure 6.8. Acanthis drone in its nominal state (top left), and subject to 11 distinct physical perturbations labeled as $p_1, ..., p_{11}$.

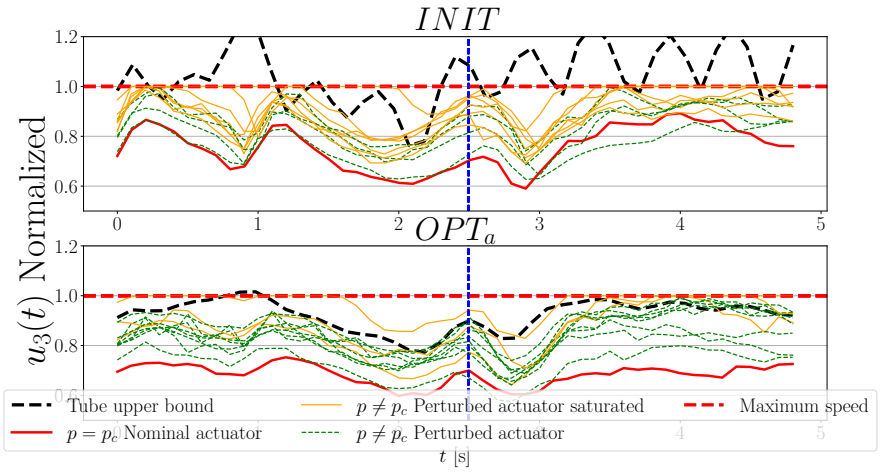

Figure 6.9. Experimental actuator speed comparison for Traj2: $INIT$ (top) vs. OPT_a (bottom). The solid red line denotes nominal actuator speed, while the dashed black line signifies the input tube upper bound (eq. 6.27). The horizontal dashed red line represents the actuator limit. The dashed green and solid orange lines depict N_p perturbation runs as in (Fig. 6.8): the solid orange lines indicate the cases in which the actuation was saturated, and the green dashed lines the cases in which no saturation occurred. Note how many fewer saturation cases are present in the OPT_a case vs. the $INIT$ case thanks to the use of the *input tubes* in the constraints (6.28).

and location of the tools on the extension rod. In the last two columns of Fig. 6.4, we present the results for Traj1 and Traj2 (1st and 2nd row), both for the optimized and non-optimized cases. Notably, it is evident from Traj1 that the deviation is more pronounced in the $INIT$ case. In fact, the point cloud — representing the different target reach locations $\boldsymbol{r}(t_w)$ and denoted with red dots in Fig. 6.4 — is larger when compared to the OPT_a case.

Fig. 6.7 provides insights for Traj1 in the experiments, where we find that the average distance to target for the $INIT$ case is 0.09 m, while it reduces to 0.07 m for the OPT_a case. This reduction signifies an improvement of 2 cm on average. Additionally, the OPT_a case demonstrates less variance, further supporting its effectiveness.

Similar results are confirmed for Traj2. As it can be observed in Fig. 6.4, the deviation from the target is both larger and more spread in the $INIT$ case compared to the OPT_a case. Returning to Fig. 6.7, for Traj2, we find that the average distance to target is about 0.19 m for the $INIT$ case while it is around 0.16 m for the OPT_a case. However, it is important to note that the $INIT$ case exhibits a higher variance with deviations reaching up to 0.31 m, whereas the maximum deviation in the OPT_a case is around 0.21 m. These results highlight the significant improvements in terms of the average distance to the target and the reduced spread achieved with the optimized OPT_a approach. Collisions with the virtual window occurred 50% of the times in the $INIT$ case while they were reduced to 25% for the optimized OPT_a case across the 12 experiments. For more detailed visualization, please refer to the accompanying video[7].

[7] https://youtu.be/QFnrQ_O2BiU

To illustrate the impact of the input sensitivity on the constraints in the optimization problem (6.28), we refer to Fig. 6.9. This figure reports the normalized actuator speed $u_3(t)$ for Traj2 in real experiments, featuring both the $INIT$ case (top) and the OPT_a case (bottom). In the nominal case $\boldsymbol{p} = \boldsymbol{p}_c$ (in red), both $INIT$ and OPT_a maintained the actuator speeds well below their maximum limit, thus ensuring satisfactory reaching of the target. However, the situation changes when the perturbations in Fig. 6.8 act on the drone. In the non-optimized $INIT$ case, a considerable number of experimental perturbed runs (in orange) quickly reached saturation of the maximum speed limit. In contrast, the OPT_a case displayed a quite better behavior, with only one of the N_p perturbations reaching saturation at the beginning (corresponding to p_8 in Fig. 6.8), potentially exceeding the predefined range of 10% deviation as per eq. (6.13). Furthermore, we observed that the OPT_a input evolution generally remained within the upper bound of the input tube (indicated by the dashed black line) obtained from the input sensitivity matrix (6.8). This highlights the significance of including input constraints (with their corresponding tube radii) to ensure robustness against parameter variations of the inputs, particularly for aggressive trajectories such as Traj2.

Chapter 7

Sensitivity-Aware Tube MPC

Even if successful in generating robust trajectories, the methodology presented in the previous chapter and the references therein only considered the case of *offline* planning. However, the ability to re-plan *online* can clearly offer an additional and substantial improvement of the robustness against uncertainties. Therefore, the main goal of this chapter is to study how to exploit the closed-loop state sensitivity (and related/derived quantities) within an *online* planning scheme formulated as a MPC problem. While MPC as a feedback controller is endowed with an inherent robustness [125], practical MPC deployment can face significant challenges related to feasibility and robustness to uncertainties, in particular when hard constraints play an active role in the motion generation. Standard MPC, in fact, generates an *open-loop trajectory* with no information regarding the ability to counteract the effects of possible future disturbances. However, the ability to correctly predict the future behavior of the robot by also accounting for the impact of feedback actions is crucial. Whenever the robot deviates from the predicted trajectory, the feedback controller will act against the disturbance to steer the robot back on the planned motion. Although this positive effect can decrease the impact of the uncertainties, it also requires, in general, an *increased control effort* that needs to be accurately taken into account if input constraints are present. Planning a feasible robust motion must then ensure that the predicted motion possesses enough control authority to satisfy constraints amid potential disturbances.

7.1 Related works

The issue of robustness against uncertainties in MPC has motivated many research efforts over the years to propose a variety of "robust MPC" schemes. The goal of robust MPC is to guarantee the feasibility of the predicted trajectory for all possible disturbance sequences [29, Chapter 3]. When dealing with uncertain systems, the set of all possible trajectories can be seen as a *bundle*. Robust MPC methods try to control the bundle of trajectories to guarantee feasibility. Since MPC acts online as a controller, the time complexity of one iteration of any MPC algorithm must be compatible with the real-time requirements of the controlled robot. This requirement has naturally shaped MPC methods into searching for the best tradeoff between optimality, robustness, and complexity.

A way to directly control the bundle of trajectories, referred to as the scenario-tree approach [126], is to discretize the disturbance set and to control the evolution of all the possible realizations of the system. This approach has, however, limited practical applicability because of its computational complexity. A more practical approach is to find an outer approximation of the bundle of perturbed trajectories in the form of time-varying *tubes*. The so-called Tube MPC, first developed for linear systems [127] and then extended to nonlinear ones [128, 129], aims at finding a tractable way to compute such tubes, possibly *online*. The tube computation typically involves a propagation of the disturbances over the closed-loop dynamics. In most cases, the computation of the tubes' cross-section relies on an *ancillary* control law providing disturbance rejection through feedback. Such controller can be fixed a-priori [130–132] or it can be parameterized and determined online as part of the algorithm in order to, for instance, minimize the tubes' cross-section [129]. Typically, an ellipsoidal approximation of the tubes' cross-section around the nominal trajectory is used. In [129], the tubes' cross-section is obtained, along with the parameterized feedback gains, by solving a Semi-Definite Program point-wise for each predicted sampling instant. Similarly, [130] defines a point-wise optimization problem to find the direction and length of the ellipsoid axes for each state dimension. In [131], an incremental Lyapunov function with a corresponding incrementally stabilizing feedback is precomputed offline and used by the MPC to construct the tubes online. Moreover, this feedback action is then added to the input computed by the MPC and applied to the system in closed-loop. In [132], the zero-order Robust Optimization (zoRO) [133], which uses ellipsoidal uncertainty sets to robustify the constraints while neglecting their sensitivities in the optimization, is used in an MPC setting with a precomputed static feedback. The Tube MPC in [134] uses a boundary layer sliding mode controller for the ancillary control law. Notably, it optimizes both the state trajectory and the tubes at, however, a likely high computational cost as pointed out in [133]. Also in the Stochastic Nonlinear MPC setting, a prestabilizing infinite horizon LQR controller is added to the control action to propagate uncertainties [135]. As an alternative to the use of an ancillary controller altogether, [136] uses min-max differential inequalities to obtain Linear Matrix Inequality constraints which are added to the MPC optimization problem to make the (ellipsoidal) tubes forward invariant.

The previously mentioned methods address the Robust MPC problem by proposing solutions on a spectrum with varying degrees of complexity. For example, some methods trade performance for simplicity by fixing the ancillary controller with a static linear feedback [130–132, 135]. This helps in simplifying the uncertainty propagation but can decrease the performance if the system is not operating close to the linearization point. More complex methods can instead provide strong robustness theoretical guarantees, but they also typically have limited practical applicability due to their computation time complexity, which can exceed that of a standard non-robust MPC by several orders, and their reliance on disturbance bounds that can be challenging to certify [131, 136]. Motivated by these considerations and the challenges posed by this problem, the main goal of this work was to propose a *computationally efficient* and *tractable* Robust MPC scheme that falls in the middle of this spectrum.

7.2 Contribution

In this chapter, we introduce a method to robustify MPC control against parametric uncertainties in the robot model by making use of *sensitivity-based tubes*. This allows to optimize for the worst-case scenario of parameter uncertainty in the closed-loop and to explicitly take into account the future feedback actions of the controller, thus reducing conservativeness. We make use of the results of sensitivity analysis of optimization problems [137,138] to compute *online* the equivalent gains of the MPC action, thus freeing from the need for an (often) precomputed and unconstrained ancillary controller, which does not necessarily capture well the effect of the actual feedback action provided by the MPC scheme. Crucially, this allows our proposed robust MPC scheme to be aware of how the presence of hard constraints affects the feedback action itself, making the uncertainty prediction more accurate and potentially less conservative.

Sensitivity analysis has often been used in the MPC context, for instance in differentiating MPC policies for learning purposes [139], or to provide high-frequency feedback to apply during the sampling interval in cases where the possibly complex MPC controller cannot run sufficiently fast [6, 125, 140, 141]. Compared to these works in which the MPC gains are only used for reactive control, we are instead interested in approximating the MPC feedback action over the whole prediction horizon for propagating the closed-loop sensitivity. This requires the development of an efficient way of computing the predicted MPC gains, enabling the proposed method to be suitable for real-time use. It is worth noting that numerical optimal control methods based on the iterative Linear Quadratic Regulator (iLQR) naturally provide linear feedback gains as a byproduct of the Riccati recursion performed to solve the OCP; see e.g., [13] for the unconstrained case and [6, 142] for the extension to equality and inequality constrained problems. In Sect. 7.3.2 (and proof in Sect. 7.3.4) we present an alternative way of computing such gains in conjunction with an off-the-shelf QP solver and a standard direct multiple shooting transcription of the OCP. This makes the proposed method applicable to general constrained MPC problems not necessarily relying on specialized solvers.

Moreover, the resulting "robust MPC" problem has a computational complexity comparable to that of a standard non-robust MPC, with the only additional cost of computing the MPC gains and propagating the closed-loop sensitivity which proves to be computationally light. These features make the proposed MPC scheme computationally efficient and suitable for real-time use in all those cases in which a standard (non-robust) MPC could have been implemented for the robot under consideration. These points, together with the fact that the closed-loop sensitivity represents an accurate and straightforward quantity to analyze the effect of the uncertainties on generic nonlinear systems, allow the application of the proposed MPC scheme to complex robotic systems without having to rely on hand-crafted robustness conditions or on unnecessary simplifications of the robot model. The proposed robust MPC scheme offers a tractable and convenient approach for adding a robustness layer to any standard, and possibly already available, MPC controller, by thus increasing safety and performance during task execution with a very minor computational overhead.

The rest of the chapter is structured as follows. In Sect. 7.3 we describe the

proposed robust MPC algorithm and in Sect. 7.4 we showcase its application to two case studies involving a Quadrotor affected by model uncertainties, providing statistical simulation results over multiple scenarios.

7.3 The proposed approach

The safety and performance of the closed-loop system (robot + MPC controller) are subject to two main conditions: (*i*) the ability of the MPC to find a feasible solution and remain feasible during motion, and (*ii*) the satisfaction of the constraints in closed-loop, i.e., whether or not predicting satisfaction of a constraint will result in its actual satisfaction during motion. As previously discussed, a discrepancy between the real parameters $\boldsymbol{p}$ of the robot and the nominal parameters $\boldsymbol{p}_c$ used by the MPC controller can result in a divergence between the actual closed-loop trajectory and the one generated in the MPC prediction phase. In general, the MPC will cope with this discrepancy by searching for an alternative (but still feasible) optimal solution, and satisfaction of the state/input constraints plays a crucial role in the determination of the feasibility of this MPC "corrective" action.

The goal of this section is to make use of the closed-loop sensitivity (and related quantities) for generating a suitable restriction of the constraints over the prediction horizon that can guarantee feasibility also in the presence of model uncertainties. The aim of this restriction is to account for the fact that, in case of model uncertainties, the state and the input trajectories will deviate from the predicted ones. The proposed constraint restriction will instead force the MPC to plan trajectories that optimize the worst-case deviations due to parameter uncertainty, thus ensuring that enough control authority is *always* left available to counteract the effects of uncertainties on the closed-loop system. This naturally introduces a possible trade-off between robustness and performance that will be discussed in the results of Sect. 7.4.

7.3.1 Overview

The proposed MPC scheme, denoted as *Sensitivity-aware Tube MPC* (ST-MPC), consists of two main steps: (*i*) The sensitivity of the solution guess is evaluated using (6.9) and (6.10), from which the sensitivity-based tube radii are obtained as in (6.23) to find the worst-case predicted constraint deviation; (*ii*) When the feedback for the current state $\hat{\boldsymbol{x}}_k$ is available, the following optimization problem

$$\begin{cases} \underset{\boldsymbol{u}}{\text{minimize}} & \displaystyle\sum_{i=0}^{C-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_C) & \\ \text{subject to} & \boldsymbol{x}_0 = \hat{\boldsymbol{x}}_k & \\ & \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{p}_c) & \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{\rho}_i^{\boldsymbol{h}} \leq \boldsymbol{0} & \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{u}_{\min} + \boldsymbol{\rho}_i^{\boldsymbol{u}} \leq \boldsymbol{u}_i \leq \boldsymbol{u}_{\max} - \boldsymbol{\rho}_i^{\boldsymbol{u}} & \forall i \in \mathbb{I}_0^{C-1} \end{cases} \tag{7.1}$$

is solved by approximating it as a QP using the RTI method described in Sect. 2.2. Note how, in (7.1), the constraints are reduced by the tube radii $\boldsymbol{\rho}_i^{\boldsymbol{h}}$ (being the tube radii for the task constraint function $\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})$) and $\boldsymbol{\rho}_i^{\boldsymbol{u}}$ that act as back-off terms to

account for the effects of parametric uncertainty. These are obtained by following the procedure described in Sect. 6.2 for the input case, and by simply repeating the same procedure on the sensitivity of any function $\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})$ of interest for the case of generic task constraints. Once the optimal solution is found, the input $\boldsymbol{u}_0^*$ is sent to the system and the algorithm is repeated.

In order to compute the tube radii $\boldsymbol{\rho} = (\boldsymbol{\rho^h}, \boldsymbol{\rho^u})$ from the state and the input sensitivities $\boldsymbol{\Pi}$ and $\boldsymbol{\Theta}$, one must first determine the feedback gains of the MPC controller that describe how the solution of (7.1) varies w.r.t. changes in $\hat{\boldsymbol{x}}_k$. In the MPC context, we use the notation $\boldsymbol{F}_{k|k}$ to refer to the first-order approximation of the MPC control action. More in general, the notation $\boldsymbol{F}_{i|j}$ is introduced to express the dependence of the input at time t_i on the state at a time $t_j \leq t_i$. This will become relevant shortly, when we will express how the predicted (future) input depends directly on the initial condition.

The step of computing the MPC feedback gains is not straightforward and the following Sect. 7.3.2 details the proposed procedure for obtaining the term $\boldsymbol{F}_{k|k}$. Finally, Sect. 7.3.3 provides a detailed description of the final algorithm.

7.3.2 Computing the MPC feedback gains

Consider the QP problem (2.24) obtained applying the RTI method to OCP (7.1) and denote its optimal solution as $\boldsymbol{y}^* = (\Delta \boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$, collecting the primal and dual variables and recall that the optimal state and input trajectory can be obtained as $\boldsymbol{w}^* = \bar{\boldsymbol{w}} + \Delta \boldsymbol{w}^*$, where $\bar{\boldsymbol{w}}$ is the solution guess around which the problem has been linearized. Let us partition the output of the MPC controller as

$$\boldsymbol{w}^*(\hat{\boldsymbol{x}}_k, \boldsymbol{p}_c) = \begin{pmatrix} \boldsymbol{x}_i^* & \forall i \in \mathbb{I}_0^C \\ \boldsymbol{u}_0^* & \\ \boldsymbol{u}_i^* & \forall i \in \mathbb{I}_1^{C-1} \end{pmatrix},$$

which renders explicit the control action $\boldsymbol{u}_0^*$ to be applied to the system, the predicted optimal input trajectory $\boldsymbol{u}_i^*$, and the associated state trajectory $\boldsymbol{x}_i^*$, respectively. Note that the MPC solution is essentially an open-loop trajectory to be applied to the system with no information on the behavior of the controller in case of disturbances that would make it deviate from the plan. The goal of this section is to show how the main results of parametric optimization can be exploited to approximate the control policy in the vicinity of the predicted trajectory, and how this information can be leveraged to improve the robustness of the MPC controller against model uncertainties.

The MPC feedback gains are obtained by differentiating the optimal solution $\boldsymbol{y}^*$ and extracting the components relative to the first input $\boldsymbol{u}_0^*$. In fact, note how the whole optimization problem can be seen as a function of the initial state $\hat{\boldsymbol{x}}_k$ and of the nominal model parameters $\boldsymbol{p}_c$. Under mild regularity assumptions [125,137,143], one can compute the sensitivities as

$$\frac{d\boldsymbol{w}^*(\hat{\boldsymbol{x}}_k, \boldsymbol{p}_c)}{d\hat{\boldsymbol{x}}_k} = \begin{pmatrix} \frac{d\boldsymbol{x}_i^*}{d\hat{\boldsymbol{x}}_k} \\ \frac{d\boldsymbol{u}_0^*}{d\hat{\boldsymbol{x}}_k} \\ \frac{d\boldsymbol{u}_i^*}{d\hat{\boldsymbol{x}}_k} \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_{k+i|k} & \forall i \in \mathbb{I}_0^C \\ \boldsymbol{F}_{k|k} & \\ \boldsymbol{F}_{k+i|k} & \forall i \in \mathbb{I}_1^{C-1} \end{pmatrix} \tag{7.2}$$

by applying the implicit function theorem (IFT) to the first-order optimality conditions of the problem, and noting that by definition $\frac{d\boldsymbol{w}^*}{d\hat{\boldsymbol{x}}_k} = \frac{d\Delta\boldsymbol{w}^*}{d\hat{\boldsymbol{x}}_k}$. More in detail, assume that the LICQ, the SOSC, and SC hold at the solution $\boldsymbol{y}^*$ (cfr. Sect. 2.3). The optimal solution of (2.24) is characterized by the KKT conditions:

$$\boldsymbol{\mathcal{R}}(\boldsymbol{y}, \hat{\boldsymbol{x}}_k)|_{\boldsymbol{y}^*} = \begin{pmatrix} \boldsymbol{H}\Delta\boldsymbol{w}^* + \boldsymbol{c} + \boldsymbol{E}^T\boldsymbol{\lambda}^* + \boldsymbol{G}_{\mathcal{A}}^T\boldsymbol{\mu}_{\mathcal{A}}^* \\ \boldsymbol{E}\Delta\boldsymbol{w}^* + \boldsymbol{\eta}(\hat{\boldsymbol{x}}_k) \\ \boldsymbol{G}_{\mathcal{A}}\Delta\boldsymbol{w}^* + \boldsymbol{g}_{\mathcal{A}} \end{pmatrix} = \boldsymbol{0}, \tag{7.3}$$

where subscript $\mathcal{A}$ denotes the set of active inequality constraints at the solution $\boldsymbol{y}^*$, and we have dropped for simplicity the dual feasibility and complementarity slackness conditions as they are assumed to be satisfied and are not involved in the following derivation, provided that SC holds. Under the conditions stated above, we can apply the IFT to (7.3) and differentiate with respect to $\hat{\boldsymbol{x}}_k$ to obtain

$$\begin{aligned} \frac{d}{d\hat{\boldsymbol{x}}_k}\, \boldsymbol{\mathcal{R}}(\boldsymbol{y}(\hat{\boldsymbol{x}}_k), \hat{\boldsymbol{x}}_k)|_{\boldsymbol{y}^*} &= \boldsymbol{0}, \\ \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}} \frac{d\boldsymbol{y}}{d\hat{\boldsymbol{x}}_k} + \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \hat{\boldsymbol{x}}_k} &= \boldsymbol{0}, \end{aligned} \tag{7.4}$$

so that

$$\frac{d\boldsymbol{y}}{d\hat{\boldsymbol{x}}_k} = -\left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}}\right)^{-1} \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \hat{\boldsymbol{x}}_k}. \tag{7.5}$$

Note that $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}}$, being the Jacobian of the KKT conditions (7.3), is nothing else than the KKT matrix of the QP, i.e.,

$$\boldsymbol{\mathcal{K}}^* = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}} = \begin{pmatrix} \boldsymbol{H} & \boldsymbol{E}^T & \boldsymbol{G}_{\mathcal{A}}^T \\ \boldsymbol{E} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}_{\mathcal{A}} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}. \tag{7.6}$$

A crucial point here is that this quantity is *already available* from the construction of the QP problem (2.24), having linearized around the solution guess $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$, by simply removing the rows of $\boldsymbol{G}$ relative to the inactive constraints. We also note that, in the general case, the inversion of the (large) matrix $\boldsymbol{\mathcal{K}}^*$ could have a non-negligible computational cost. However, this is not the case in our context thanks to its structured sparsity, which then allows to efficiently invert matrix $\boldsymbol{\mathcal{K}}^*$ and solve (7.4) by relying on efficient factorization and solution algorithms for sparse linear systems. Recalling (7.2), it is then trivial to extract the quantities

$$\boldsymbol{F}_{k|k},\ \boldsymbol{F}_{k+i|k},\ \text{and}\ \boldsymbol{P}_{k+i|k} \tag{7.7}$$

from (7.5), which correspond to the sensitivities of the control input $\boldsymbol{u}_0$, of the predicted inputs, and of the predicted states w.r.t. $\hat{\boldsymbol{x}}_k$.

We note, however, that the evaluation of the sensitivity (6.9) over the prediction horizon of the MPC requires the availability of the controller gains $\boldsymbol{F}_{k|k}$ over the *whole future horizon*, that is, availability of $\boldsymbol{F}_{k+i|k+i}$, $\forall i \in \mathbb{I}_1^{C-1}$. These quantities represent the sensitivity of the future inputs $\boldsymbol{u}_{k+i}$ with respect to a variation in the future state $\hat{\boldsymbol{x}}_{k+i}$ and they are unfortunately not directly available from (7.7)

that only provides the sensitivity of the full MPC solution with respect to the *current state* $\hat{\boldsymbol{x}}_k$. At first glance, the computation of $\boldsymbol{F}_{k+i|k+i}$, $i \in \mathbb{I}_1^{C-1}$ would require to solve (7.3–7.7) for each $i \in \mathbb{I}_1^{C-1}$, which would quickly become computationally unfeasible in the typically short time available. To circumvent this problem, we propose to employ a suitable approximation of the future gains thanks to the information regarding the future inputs provided by the parametric sensitivity of the MPC. In fact, once the full MPC sensitivity (7.7) has been obtained, one can utilize the sensitivity of the future states and inputs to compute the future gains relative to the current prediction horizon as

$$\boldsymbol{F}_{k+i|k+i} \simeq \tilde{\boldsymbol{F}}_{k+i|k+i} = \frac{d\boldsymbol{u}_i}{d\boldsymbol{x}_i} = \boldsymbol{F}_{k+i|k}\boldsymbol{P}_{k+i|k}^{-1} \quad \forall i \in \mathbb{I}_1^{C-1}, \tag{7.8}$$

which represent the predicted gains at the optimal solution, i.e., how a change in the predicted state $\boldsymbol{x}_i$ would affect the input $\boldsymbol{u}_i$ at that time. The important point here is that (7.8) reuses part of the sensitivities computed w.r.t. the initial state $\hat{\boldsymbol{x}}_k$, and it is thus much more computationally efficient and actually suitable for real-time use.

A detailed proof of Eq. (7.8) will be discussed in Sect. 7.3.4. However, a sketch is proposed here for the reader's convenience. The IFT can be applied to the KKT conditions (7.3) using as explicit variable any element $\boldsymbol{x}_i$ of the state trajectory, from which it would be straightforward to compute the sensitivity $\tilde{\boldsymbol{F}}_{k+i|k+i}$ with respect to that state similarly to (7.5). However, this requires inverting, for each i, a large matrix obtained by differentiating (7.3) with respect to the remaining implicit variables, which would make it computationally expensive. Instead, by noting that this matrix can be written as a rank-n_x correction to matrix $\boldsymbol{\mathcal{K}}^*$ (7.6), one can utilize the Woodbury matrix identity [144] for obtaining an efficient formula to compute $\tilde{\boldsymbol{F}}_{k+i|k+i}$. Moreover, after some simplifications related to the structure of the problem, it is possible to extract (7.8), making explicit the dependence on the previously computed sensitivities.

Remark 7.3.1 (Active constraints selection)**.** *The proposed computation of the feedback gains depends on the (strongly) active constraint set $\mathcal{A}$ which characterizes which inequality constraints are affecting, locally, the solution of the MPC problem, and thus the MPC gains. As the MPC sensitivity is computed around the optimal solution $\boldsymbol{y}^*$, it is in general possible to trivially select the active constraints at the solution to construct $\mathcal{A}$ by looking at the multipliers $\boldsymbol{\mu}^*$. However, since our main goal is to obtain an approximation of the future gains* (7.8)*, we employ a different strategy as this choice would result in a more conservative approximation. In fact, being the future input constraints restricted by the tubes, this will result in the constraint being active even when the nominal constraint without tubes is not, altering the relative gains due to this input restriction. On the other hand, as the MPC controller works with a receding horizon approach, the* actual *future input $\boldsymbol{u}_{k+i}$, first input of the solution of the problem at time t_{k+i}, will not see this restriction and will effectively only have to satisfy the nominal input constraint. For this reason, we select the active constraint set $\mathcal{A}$ by neglecting the input constraints which are only active with $\boldsymbol{\rho}^{\boldsymbol{u}} > 0$. All the other constraints, e.g., state constraints to perform obstacle avoidance, are instead taken into account.*

Remark 7.3.2 (KKT regularization)**.** *In some instances, although the optimization problem is feasible, the KKT matrix might be ill-conditioned or singular due to the LICQ condition not being satisfied at the solution. Most solvers rely on regularization techniques to still be able to produce a solution. In our case, this is necessary for computing the MPC sensitivity. Therefore, we employ a proximal regularization of the Lagrange multipliers of the active inequality constraints [145, 146]. Recall the KKT matrix* $\boldsymbol{\mathcal{K}}^*$ (7.6) *and note that by construction the block*

$$\begin{pmatrix} \boldsymbol{H} & \boldsymbol{E}^T \\ \boldsymbol{E} & \boldsymbol{0} \end{pmatrix}$$

is assumed full rank. In order to regularize the matrix we apply a proximal regularization, obtaining the augmented Lagrangian $\mathcal{L}_A(\Delta\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}_{\mathcal{A}}) = \mathcal{L}(\Delta\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}_{\mathcal{A}}) - \frac{\varrho}{2}\|\boldsymbol{\mu}_{\mathcal{A}} - \boldsymbol{\mu}^*_{\mathcal{A}}\|^2_{\boldsymbol{\Lambda}^{-1}}$*, where* $\boldsymbol{\Lambda} = \operatorname{diag}(\boldsymbol{\mu}^*_{\mathcal{A}})$*,* $\varrho > 0$*, which results in a damping term in the bottom right block as*

$$\boldsymbol{\mathcal{K}}^*_{reg} = \begin{pmatrix} \boldsymbol{H} & \boldsymbol{E}^T & \boldsymbol{G}^T_{\mathcal{A}} \\ \boldsymbol{E} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}_{\mathcal{A}} & \boldsymbol{0} & -\varrho\boldsymbol{\Lambda}^{-1} \end{pmatrix}. \tag{7.9}$$

In practice, this can be interpreted as a normalized constraint relaxation that minimizes the Lagrange multipliers variation [147], ensuring that the KKT matrix is non-singular so that the MPC gains (7.8) *can be computed.*

7.3.3 The ST-MPC algorithm

With reference to Algorithm 1, we can now describe in detail the various steps of the full ST-MPC scheme. First, we perform an offline initialization of the controller and prepare the QP for its next iteration (lines 1-4). While online (lines 6-10), we divide the solution of the problem into two phases: (*i*) a *feedback* phase (lines 6-7), in which the QP is solved when the current state becomes available, and (*ii*) a *preparation* phase (lines 8-10), in which the tubes are computed and the QP for the next iteration at t_{k+1} is constructed on the basis of the current solution guess $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$. Note that, while the feedback phase needs to be performed when the current state $\hat{\boldsymbol{x}}_k$ becomes available, the preparation phase for the next iteration of the scheme at t_{k+1} does not require knowledge of next state $\hat{\boldsymbol{x}}_{k+1}$, and it can thus be performed just after having sent the input command to the system (line 7), reducing the control delay of the scheme [26].

In more detail, Alg. 1 can be described line-by-line as follows:

L1 At the initial state $\hat{\boldsymbol{x}}_0$, start from a trivial initial guess $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ and $\boldsymbol{\rho} \equiv \boldsymbol{0}$, and solve the problem to find a feasible solution without considering the constraint restrictions from the tubes;

L2 Compute the sensitivity of the MPC (7.5) and the future gains using (7.8). Propagate the state and input sensitivities $\boldsymbol{\Pi}$ and $\boldsymbol{\Theta}$ via (6.9–6.10), and compute the evolution of the tube radii $\boldsymbol{\rho}$ over the predicted trajectory using (6.23);

Algorithm 1: ST-MPC

```
1  x*, u*, λ*, μ* ← Initialize(x̂_0)
2  ρ ← ComputeTubes(x*, u*, λ*, μ*)
3  x̄, ū ← x*, u*
4  PrepareQP(x̄, ū, ρ)
5  while running do
6  |   x*, u*, λ*, μ* ← SolveQP(x̂_k)
7  |   SendCommand(u*_0)
8  |   ρ ← ComputeTubes(x*, u*, λ*, μ*)
9  |   x̄, ū ← ShiftSolution(x*, u*)
10 |   PrepareQP(x̄, ū, ρ)
11 end
```

L3-4 Prepare the QP for the next feedback phase, evaluating all the derivatives and residuals (2.23), but this time by including the tubes;

L6-7 Given the current state $\hat{\boldsymbol{x}}_k$, solve optimization problem (7.1) by solving the associated RTI-QP (2.24) and send the control action $\boldsymbol{u}_0^*$ to the robot;

L8 Compute the tube radii $\boldsymbol{\rho}$ as in **L2**;

L9-10 Shift the solution one step in time and prepare the QP for the next feedback phase.

As a final remark, we highlight a strength of the proposed ST-MPC: since the tubes are obtained during the preparation phase, the computational cost of their evaluation is not added to the delay introduced by the MPC controller due to the feedback phase. This feature, along with the fact that the optimization problem has the same number of constraints and decision variables as a standard MPC, makes the impact of the additional computations in ST-MPC practically negligible compared to the computational time of a standard MPC. Therefore, if a standard MPC is amenable to a real-time implementation, its ST-MPC version (with the constraint restrictions from the tubes) will also typically be implementable in real-time. This is, arguably, a major strength of the proposed approach: ST-MPC represents a viable alternative to MPC schemes with an added inherent robustness to model uncertainties.

7.3.4 Efficient computation of the MPC gains over the prediction horizon

Before going further, we review the detailed proof of (7.8).

In the computation of the sensitivity of the MPC — described in Sect. 7.3.2 — the sensitivity of the whole predicted trajectory $(\boldsymbol{x}, \boldsymbol{u})$ is computed with respect to the initial state $\hat{\boldsymbol{x}}_k$ (or, equivalently, $\boldsymbol{x}_0$). We now show how the same reasoning applies to all $\boldsymbol{x}_i$, and how the sensitivities for $i = 1, \dots, C-1$ can be computed without performing any additional KKT factorization and inversion.

First, note that with a simple rearrangement of variables, one can apply the implicit function theorem on the KKT system (7.3) with respect to the explicit variable $\boldsymbol{x}_i$. Defining $\boldsymbol{z} = (\boldsymbol{x}_0, \dots, \hat{\boldsymbol{x}}_k, \dots, \boldsymbol{x}_C, \boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ by replacing $\boldsymbol{x}_i$ with $\hat{\boldsymbol{x}}_k$, one can then obtain

$$\begin{aligned} \boldsymbol{\mathcal{R}}(\boldsymbol{z}(\boldsymbol{x}_i), \boldsymbol{x}_i)|_{\boldsymbol{z}^*} &= \boldsymbol{0} \\ \frac{d}{d\boldsymbol{x}_i} \boldsymbol{\mathcal{R}}(\boldsymbol{z}(\boldsymbol{x}_i), \boldsymbol{x}_i)|_{\boldsymbol{z}^*} &= \boldsymbol{0} \\ \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}} \frac{d\boldsymbol{z}}{d\boldsymbol{x}_i} + \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}_i} &= \boldsymbol{0}, \end{aligned} \tag{7.10}$$

from which it follows

$$\frac{d\boldsymbol{z}}{d\boldsymbol{x}_i} = -\left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}}\right)^{-1} \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}_i}.$$

We now show how, leveraging the availability of the inverse of the KKT matrix $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}}$, the sought quantities can be computed without the need of inverting the large matrix $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}}$ for each i. Noting that $\frac{d\boldsymbol{z}}{d\boldsymbol{x}_i}$ contains $\frac{d\boldsymbol{u}_i}{d\boldsymbol{x}_i}$, we define the selection matrix $\boldsymbol{v}_{u_i}$ such that

$$\tilde{\boldsymbol{F}}_{k+i|k+i} = \frac{d\boldsymbol{u}_i}{d\boldsymbol{x}_i} = \boldsymbol{v}_{u_i} \frac{d\boldsymbol{z}}{d\boldsymbol{x}_i}. \tag{7.11}$$

Since $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}_i}$ is the i-th n_x-dimensional column block of the KKT matrix (7.6), we can also define $\boldsymbol{v}_{x_i}$ such that

$$\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}_i} = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}} \boldsymbol{v}_{x_i}.$$

We can then rewrite

$$\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}} = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}} + \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \hat{\boldsymbol{x}}_k} - \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}_i}\right) \boldsymbol{v}_{x_i}^T,$$

which makes it possible to compute the inverse of $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}}$ by only computing a rank-n_x correction via the Woodbury matrix identity [144]. Define

$$\boldsymbol{\mathcal{K}} = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{y}}, \quad \boldsymbol{\kappa}_i = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{x}_i}, \quad \boldsymbol{c} = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \hat{\boldsymbol{x}}_k},$$

and $\boldsymbol{d}_i = \boldsymbol{c} - \boldsymbol{\kappa}_i$. Then

$$\begin{aligned} \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}}\right)^{-1} &= \left(\boldsymbol{\mathcal{K}} + \boldsymbol{d}_i \boldsymbol{v}_{x_i}^T\right)^{-1} \\ &= \boldsymbol{\mathcal{K}}^{-1} - \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{d}_i \left(\boldsymbol{I}_n + \boldsymbol{v}_{x_i}^T \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{d}_i\right)^{-1} \boldsymbol{v}_{x_i}^T \boldsymbol{\mathcal{K}}^{-1}. \end{aligned} \tag{7.12}$$

We can now combine (7.10), (7.11) and (7.12) to find

$$\frac{d\boldsymbol{u}_i}{d\boldsymbol{x}_i} = -\boldsymbol{v}_{u_i} \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{\kappa}_i + \boldsymbol{v}_{u_i} \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{d}_i \left(\boldsymbol{I}_n + \boldsymbol{v}_{x_i}^T \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{d}_i\right)^{-1} \boldsymbol{v}_{x_i}^T \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{\kappa}_i. \tag{7.13}$$

Note that, by definition

$$\begin{aligned} \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{\kappa}_i &= \boldsymbol{v}_{x_i}, \\ \boldsymbol{v}_{u_i} \boldsymbol{\mathcal{K}}^{-1} \boldsymbol{\kappa}_i &= \boldsymbol{v}_{u_i} \boldsymbol{v}_{x_i} = \boldsymbol{0}, \\ \boldsymbol{v}_{x_i}^T \boldsymbol{v}_{x_i} &= \boldsymbol{I}_n. \end{aligned}$$

Then, having defined

$$\boldsymbol{F}_{k+i|k} = \boldsymbol{v}_{u_i}\boldsymbol{\mathcal{K}}^{-1}\boldsymbol{c},$$
$$\boldsymbol{P}_{k+i|k} = \boldsymbol{v}_{x_i}^T\boldsymbol{\mathcal{K}}^{-1}\boldsymbol{c},$$

Eq. (7.13) becomes

$$\tilde{\boldsymbol{F}}_{k+i|k+i} = \boldsymbol{v}_{u_i}\boldsymbol{\mathcal{K}}^{-1}\boldsymbol{c}\left(\boldsymbol{v}_{x_i}^T\boldsymbol{\mathcal{K}}^{-1}\boldsymbol{c}\right)^{-1} = \boldsymbol{F}_{k+i|k}\boldsymbol{P}_{k+i|k}^{-1},$$

finally yielding the desired MPC gains. We highlight again how this formula provides an efficient mean for computing $\tilde{\boldsymbol{F}}_{k+i|k+i}$ as it involves the inversion of the *small* $n_x \times n_x$ matrix $\boldsymbol{P}_{k+i|k}$ instead of the inversion of the *large* matrix $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{z}}$. ∎

7.4 Application to Quadrotor motion control

In order to demonstrate the effectiveness of the proposed ST-MPC algorithm, we present here a series of tests involving the control of a Quadrotor UAV subject to parametric uncertainty. Indeed, Quadrotors are a very popular robotic platform capable of agile and/or aggressive maneuvers but, at the same time, they are also typically subject to some unavoidable degree of uncertainty in their dynamical model because of, e.g., complex aerodynamics (resulting in uncertain drag/thrust coefficients), or uncertain location of the center of mass (CoM). Moreover, the full 3D Quadrotor dynamics is highly nonlinear, which then contributes to showcase the applicability of our framework to non-trivial systems.

The formulation (7.1) is quite general and includes both input saturation constraints $\boldsymbol{u}_{\min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{\max}$ and more generic input/state constraints $\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \leq \boldsymbol{0}$ with their corresponding tubes. However, in the following case studies we only consider tubes on the *input constraints* since, in many instances, input constraints are the main limiting factor for finding a feasible solution that also satisfies other task constraints. Indeed, in many applications, presence of task constraints alone does not threaten the recursive feasibility of the MPC which, given "infinite" control authority, is virtually able to always find a feasible solution. Also when only actuation limits are present, for instance when MPC is used as a reference tracking controller (as in the first of the following case studies), input constraints are still the ones ultimately determining the performance of the system during the most aggressive maneuvers (minimization of the tracking error typically tends to increase the control effort which is limited by the input constraints).

The rest of this section is structured as follows. In Sect. 7.4.1, we describe the two test scenarios, (*i*) tracking of aggressive trajectories and (*ii*) navigation through a narrow aperture, and in Sect. 7.4.2 we discuss the associated ST-MPC formulation. For each scenario, we performed a comparative statistical analysis (Sect. 7.4.3). In the first scenario, where only input constraints are present, we highlight how the proposed ST-MPC enhances the accuracy by reducing unwanted input saturation. In the second scenario, we (purposely) introduce a strict positional constraint that significantly limits the robot motion. This constraint can cause failures due to input saturations if uncertainties are not properly accounted for. By using ST-MPC we then show how our approach enhances the controller feasibility, thereby increasing the overall success rate.

Table 7.1. Trajectory tracking MPC settings

Sampling time	δ_t	0.02 s
Control horizon	C	25
Weights	$\boldsymbol{Q}_r$	$80\ \boldsymbol{I}_3$
	$\boldsymbol{Q}_v$	$2\ \boldsymbol{I}_3$
	$\boldsymbol{Q}_q$	$\mathrm{diag}(10^{-2}, 10^{-2}, 1)$
	$\boldsymbol{Q}_\omega$	$5 \cdot 10^{-2}\ \boldsymbol{I}_3$
	$\boldsymbol{Q}_u$	$\boldsymbol{I}_4$
Regularization	ϱ	10^{-2}
Max. parameter deviation	$\Delta \boldsymbol{p}_{\max}$	$(0.04, 0.04, 0.02, 0.1)$

7.4.1 Test scenarios

In the following, we consider the dynamic model for the Quadrotor with shifted CoM introduced in Sect. 6.3.1. In this case, we assume that a playload of mass $m_p = 0.2$ kg is attached at a distance $d_z = 0.15$ m from the center of the Quadrotor. With these settings, in the subsequent derivations, the set of parameters that is supposed to be *uncertain* is $\boldsymbol{p} = (g_x,\, g_y,\, g_z,\, m_p) \in \mathbb{R}^4$. This is motivated by a series of empirical results and previous works on this topic, such as [112, 113, 115], which highlighted how uncertainties in these parameters are, in practice, the most important ones in affecting the closed-loop performance of a drone, as opposed to, for instance, the thrust coefficients. Indeed the sensitivity w.r.t. the considered parameters is at least one order of magnitude higher than the other ones, when also accounting for the typical expected parameter deviation.

Tracking of aggressive trajectories

The first task consists of tracking a trajectory defined by a series of waypoints $\mathcal{W}_d \in \mathbb{R}^3$, assumed to be generated by an external planner guiding the Quadrotor through the environment. The desired trajectory is obtained by linear interpolation of the waypoints with constant velocity, resulting in a position and velocity trajectory $(\boldsymbol{r}_d(t), \dot{\boldsymbol{r}}_d(t))$. As for the orientation (yaw) we consider two possibilities: (i) keeping a constant yaw angle with a desired orientation $\boldsymbol{q}_d(t) \equiv \boldsymbol{q}_0 = (1, 0, 0, 0)$ or (ii) keeping the yaw direction of the Quadrotor always pointing towards the upcoming waypoint, i.e., by setting $\psi_d(t) = \mathrm{Atan2}(\dot{y}_d(t), \dot{x}_d(t))$ which results in a desired orientation $\boldsymbol{q}_d(t) = (\cos(\psi_d(t)/2), 0, 0, \sin(\psi_d(t)/2))$. This latter possibility is meant to increase the actuation effort during tracking and, therefore, the chances of incurring in actuation saturation. See Fig. 7.1 for a graphical depiction of this scenario.

By construction, these trajectories have discontinuous Cartesian velocities — due to the linear interpolation between the waypoints that are not continuous — and are thus not initially dynamically feasible. As typical in these cases, we rely on the MPC scheme for generating online a *feasible* motion that tracks at best the various trajectory segments. This is obtained by designing the cost function of the MPC

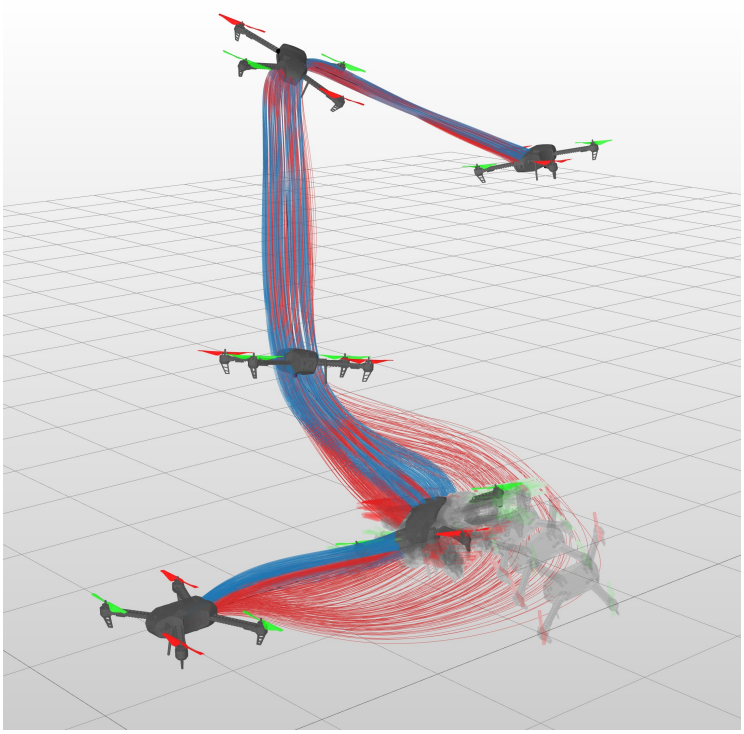

Figure 7.1. Stroboscopic view of the comparison between our proposed method (blue) and a regular MPC controller (red) while performing trajectory tracking. The spread of trajectories is generated by perturbing the physical parameters of a Quadrotor, highlighting how our proposed method is able to remain much closer to its nominal behavior.

controller to minimize the position, velocity, and orientation tracking errors together with a regularization term on the angular velocity $\boldsymbol{\omega}$ for smoothing, and with a feedforward input equal to the nominal hovering condition $\boldsymbol{u}_h = \frac{mg}{4k_f}\mathbf{1}_4$. Letting the desired state and input be $\boldsymbol{x}_d(t) = (\boldsymbol{r}_d(t), \dot{\boldsymbol{r}}_d(t), \boldsymbol{q}_d(t), \mathbf{0}_3)$ and $\boldsymbol{u}_d(t) = \boldsymbol{u}_h$, the running cost then takes the form $\ell(\boldsymbol{x}, \boldsymbol{u}) = \ell_{\boldsymbol{x}} + \ell_{\boldsymbol{u}}$, $\ell_f(\boldsymbol{x}) = \ell_f$ with

$$\ell_{\boldsymbol{x}} = \|\boldsymbol{r} - \boldsymbol{r}_d\|^2_{\boldsymbol{Q}_r} + \|\boldsymbol{v} - \dot{\boldsymbol{r}}_d\|^2_{\boldsymbol{Q}_v} + \|\boldsymbol{q} - \boldsymbol{q}_d\|^2_{\boldsymbol{Q}_q} + \|\boldsymbol{\omega}\|^2_{\boldsymbol{Q}_\omega}, \tag{7.14a}$$

$$\ell_{\boldsymbol{u}} = \|\boldsymbol{u} - \boldsymbol{u}_h\|^2_{\boldsymbol{Q}_u}, \tag{7.14b}$$

$$\ell_f = \ell_{\boldsymbol{x}}. \tag{7.14c}$$

The weights used in all the experiments of this scenario as well as the standard MPC settings are reported in Tab. 7.1.

Passing through a narrow aperture

In the second scenario, illustrated in Fig. 7.2, the robot is tasked with reaching a position goal $\boldsymbol{r}_{sp} \in \mathbb{R}^3$ placed on the opposite side of an aperture while avoiding collisions with the aperture lower and upper sides. This is encoded as a constraint on the z-position of the robot that is activated when the horizontal position (x, y) is inside the aperture.

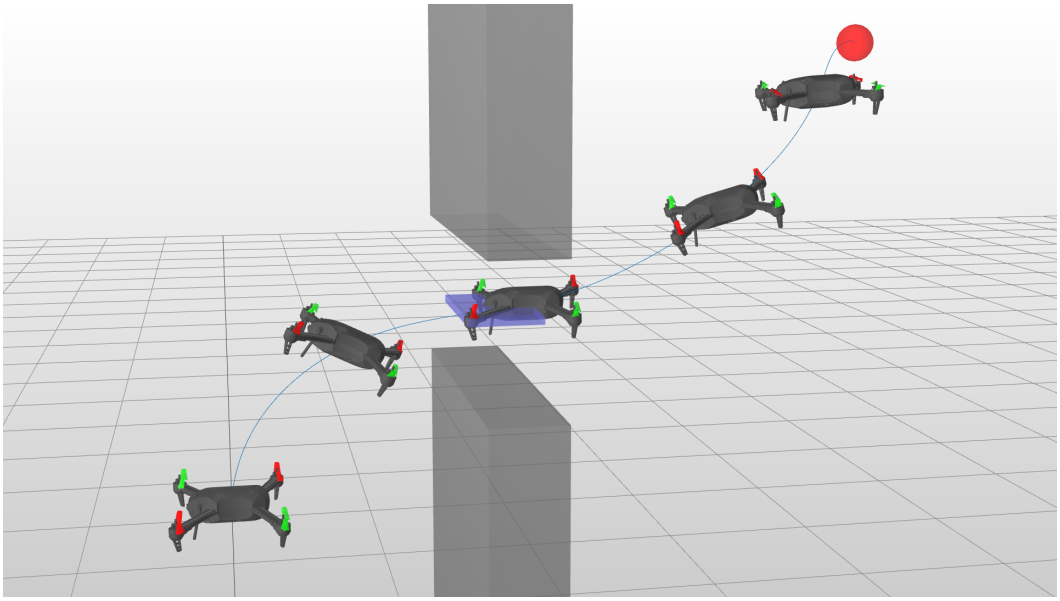

Figure 7.2. Stroboscopic view of the Quadrotor reaching the desired target (red) while passing through the aperture (safe region in light blue) with nominal parameters.

The cost function of the MPC problem is designed to make the system generate the required motion autonomously based only on the desired final state $\boldsymbol{x}_d = (\boldsymbol{r}_{sp}, \mathbf{0}_3, \boldsymbol{q}_0, \mathbf{0}_3)$ and the hovering input $\boldsymbol{u}_h$ as available information. Again, the running cost takes the form $\ell(\boldsymbol{x}, \boldsymbol{u}) = \ell_{\boldsymbol{x}} + \ell_{\boldsymbol{u}}$, $\ell_f(\boldsymbol{x}) = \ell_f$ with

$$\ell_{\boldsymbol{x}} = \|\boldsymbol{r} - \boldsymbol{r}_{sp}\|^2_{\boldsymbol{Q}_r} + \|\boldsymbol{v}\|^2_{\boldsymbol{Q}_v} + \|\boldsymbol{q} - \boldsymbol{q}_0\|^2_{\boldsymbol{Q}_q} + \|\boldsymbol{\omega}\|^2_{\boldsymbol{Q}_\omega}, \tag{7.15a}$$

$$\ell_{\boldsymbol{u}} = \|\boldsymbol{u} - \boldsymbol{u}_h\|^2_{\boldsymbol{Q}_u}, \tag{7.15b}$$

$$\ell_f = 10\ \ell_{\boldsymbol{x}}. \tag{7.15c}$$

The weights used in all the experiments of this scenario as well as the common MPC settings are reported in Tab. 7.2.

In order to avoid collisions with the environment, we design task constraints that limit the position of the Quadrotor when passing through the aperture. Let $A_{\text{fp}} \subset \mathbb{R}^2$ be the (x, y) projection of the aperture on the ground, h_z the z coordinate of the center of the aperture, and $a_z > 0$ the maximum deviation that the center of the robot can safely sustain from h_z. Also, let the task constraint function be

$$h_a(\boldsymbol{x}) = \begin{cases} z - h_z & \text{if } (x, y) \in A_{\text{fp}} \\ 0 & \text{otherwise} \end{cases}.$$

Then, the double-sided constraint

$$-a_z \leq h_a(\boldsymbol{x}_i) \leq a_z \quad \forall i \in \mathbb{I}_1^C, \tag{7.16}$$

expressed in a form compatible with (7.1)

$$\boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i) = \begin{pmatrix} h_a(\boldsymbol{x}_{i+1}) - a_z \\ -h_a(\boldsymbol{x}_{i+1}) - a_z \end{pmatrix}, \tag{7.17}$$

is able to encode the desired collision avoidance behavior.

Table 7.2. Regulation MPC settings

Sampling time	δ_t	0.02 s
Control horizon	C	$\{25, 35\}$
Weights	$\boldsymbol{Q}_r$	$\boldsymbol{I}_3$
	$\boldsymbol{Q}_v$	$\{2, 5\} \cdot 10^{-2}\ \boldsymbol{I}_3$
	$\boldsymbol{Q}_q$	$10^{-2}\ \boldsymbol{I}_3$
	$\boldsymbol{Q}_\omega$	$10^{-2}\ \boldsymbol{I}_3$
	$\boldsymbol{Q}_u$	$0.5\ \boldsymbol{I}_4$
Regularization	ϱ	10^{-2}
Max. parameter deviation	$\Delta \boldsymbol{p}_{\max}$	$(0.02, 0.02, 0.01, 0.1)$

7.4.2 Application of ST-MPC to the test scenarios

We apply the proposed ST-MPC to the two above-mentioned test scenarios following the procedure detailed in Sect. 7.3.3. For the second scenario, where the state constraint (7.16) is present, we neglect the associated tube by setting $\boldsymbol{\rho}^{\boldsymbol{h}} = \boldsymbol{0}$ as explained at the beginning of the section. Indeed, (7.16) is a stringent positional constraint that has to be active while the Quadrotor is passing through the aperture. Therefore, introducing an additional restriction might be counterproductive. On the other hand, the input tubes radii $\boldsymbol{\rho}^{\boldsymbol{u}}$ will still account for the constraint being active, providing a sufficient level of robustness as demonstrated by the results in Sect. 7.4.3.

The controller is then obtained as the solution of the following OCP:

$$\begin{cases} \underset{\boldsymbol{u}}{\text{minimize}} & \displaystyle\sum_{i=0}^{C-1} \ell(\boldsymbol{x}_i, \boldsymbol{u}_i) + \ell_f(\boldsymbol{x}_C) & \\ \text{subject to} & \boldsymbol{x}_0 = \hat{\boldsymbol{x}}_k & \\ & \boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{p}_c) & \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq \boldsymbol{0} & \forall i \in \mathbb{I}_0^{C-1} \\ & \boldsymbol{u}_{\min} + \boldsymbol{\rho}_i^{\boldsymbol{u}} \leq \boldsymbol{u}_i \leq \boldsymbol{u}_{\max} - \boldsymbol{\rho}_i^{\boldsymbol{u}} & \forall i \in \mathbb{I}_0^{C-1} \end{cases} \tag{7.18}$$

with the cost function being either (7.14) or (7.15), the Quadrotor dynamic model discretized from (6.25), the task constraint (7.17) for the second scenario, and the input saturation constraints with $(\boldsymbol{u}_{\min},\ \boldsymbol{u}_{\max}) = (0,\ 1.3 \cdot 10^6)$ for both scenarios.

For each scenario, we performed a comparative analysis of the performance between the proposed ST-MPC and a standard (non-robust) MPC controller denoted in the following as *standard MPC*, obtained by neglecting the input tubes (i.e., by setting $\boldsymbol{\rho}_i^{\boldsymbol{u}} = \boldsymbol{0}$). This is meant to highlight the benefits of considering the additional constraint restrictions from the tubes on the performance and safety of the system.

Moreover, we are also interested in showing how the use of the full sensitivity analysis of the KKT conditions described in Section 7.3.2 — which accounts for the presence of active state constraints — is crucial for the success of the ST-MPC scheme. In fact, one might wonder whether the use of an *unconstrained approximation* of the feedback action of the controller, similar to applying a pre-stabilizing action like the methods using an unconstrained ancillary controller (cfr.

Sect. 7.1), could already provide enough information to compute the tubes. We then build such an approximation by neglecting all the active inequality constraints in the KKT conditions (7.3) and by computing the feedback gains (7.8) from this reduced system. In analogy to the iterative Linear Quadratic Regulator (iLQR), which yields similar gains over the prediction horizon, we then denote this variant as *ST-MPC-LQR*.

We stress that this ST-MPC-LQR variant is introduced only as a mean to provide an additional baseline for comparison in the second test scenario, where accounting for the positional constraint proves to be essential to obtain the maximum performance. On the other hand, since the ST-MPC-LQR variant is essentially equivalent to ST-MPC when only input constraints are present, we ignore it for comparison in the first test scenario.

7.4.3 Results

We now present some statistical results for illustrating how the introduction of the input tubes improves the tracking performance, the feasibility, and thus the safety of the system. The simulations have been performed by integrating numerically the Quadrotor dynamics (6.25) with various combinations of parameters $\boldsymbol{p}$. The accompanying video[1] provides a visualization of the various simulations.

Tracking of aggressive trajectories

Two trajectories, denoted as A and B in Fig. 7.3 and Tab. 7.3, have been tracked repeatedly by selecting 450 parameter perturbations $\Delta\boldsymbol{p} \in \mathcal{P}$ (see (6.12)) in a grid, with $\Delta\boldsymbol{p}_{\max} = (0.04, 0.04, 0.02, 0.1)$, and by employing either the ST-MPC (with input tubes) or the standard MPC (without input tubes) controllers. Note that some of the parameter deviations generated in $\mathcal{P}$ will lie outside the corresponding ellipsoid $\mathcal{E}_{\boldsymbol{p}}$. Therefore, the effect of their perturbation on the closed-loop trajectory could be underestimated by the input tubes (which assume a parametric uncertainty lying in $\mathcal{E}_{\boldsymbol{p}}$). This does not constitute an issue in this tracking task where only the tracking error is affected by the uncertainty, but it will have an impact in the second test scenario which also involves the feasibility of the motion.

For each trajectory, we further consider the case where the desired yaw is kept constant $\psi_d(t) = 0$ (cases "A" and "B" in Fig. 7.3 and Tab. 7.3) and where the yaw has to track a desired trajectory $\psi_d(t) = \text{Atan2}(\dot{y}_d(t), \dot{x}_d(t))$ (cases "A-yaw" and "B-yaw" in Fig. 7.3 and Tab. 7.3).

Figure 7.3 visually depicts how, in particular for the case "A", the spread of trajectories obtained with ST-MPC remains much closer around the nominal behavior, while the standard MPC produces much larger deviations during more aggressive turns. This result is due to the input saturations that affect the Quadrotor because of the modeling errors: thanks to the input tubes, these are much better accounted for by ST-MPC which is able to generate a feasible trajectory more robust to model uncertainties while, on the other hand, the standard MPC lacks this inherent robustness and ultimately deviates more from the nominal trajectory. Comparing the two trajectories, for the case "A" the deviation is mostly localized in the first

[1] Available at https://youtu.be/ipfthFPdQLc

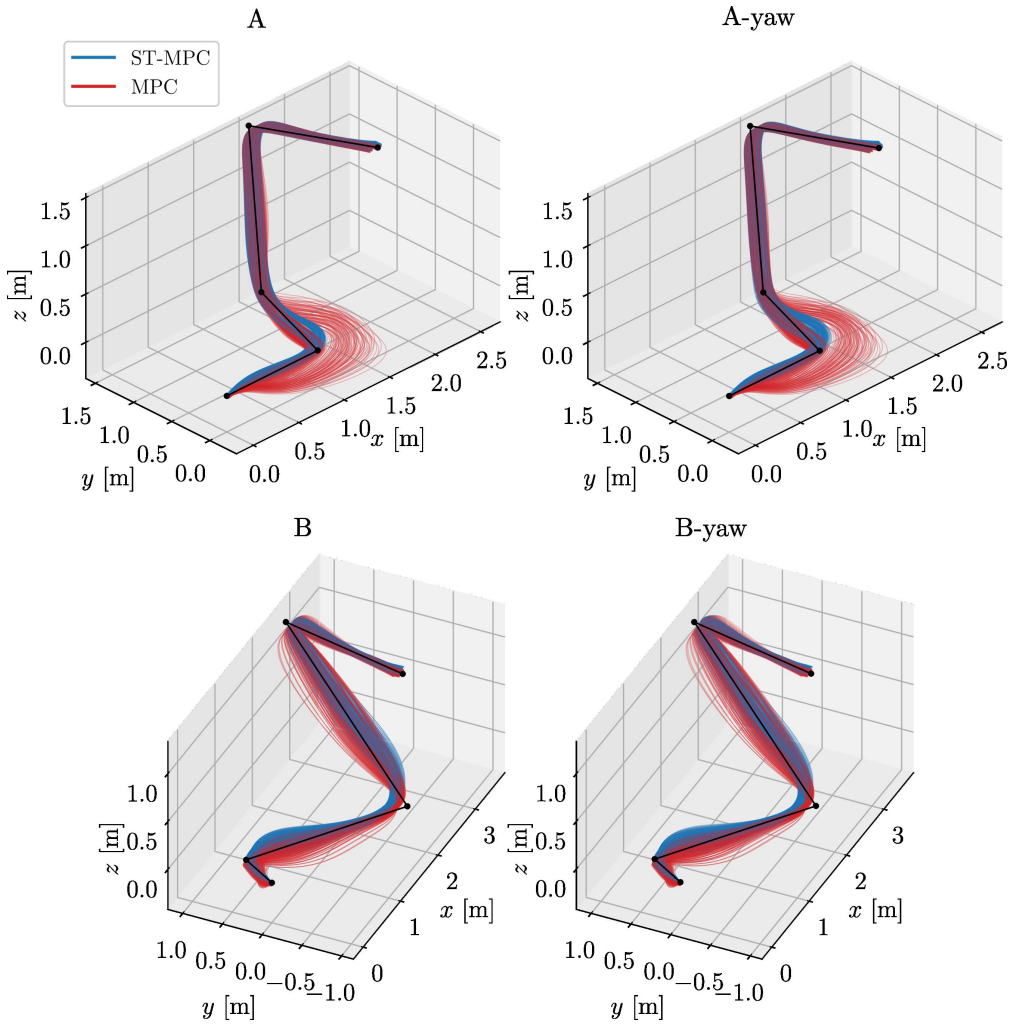

Figure 7.3. Spread of trajectories obtained for different parameter perturbations while tracking the desired trajectory (with and without yaw). Note how the proposed ST-MPC method (blue) yields a tighter spread than a standard MPC controller (red).

part of the trajectory, where input saturations are concentrated, while for the case "B" deviations are sustained over the entirety of the trajectory. As expected, when saturations are not involved in the motion generation, ST-MPC and MPC have a similar behavior in the proposed scenario.

Fig. 7.4 shows an example of the resulting input trajectory for a particular parameter deviation $\Delta \boldsymbol{p} = (0.02, -0.02, 0.0, 0.05)$. Note how, after the first instants, the standard MPC saturates three out of the four control inputs, resulting in a large deviation from the nominal trajectory, measured with the Cartesian position error $\boldsymbol{e_r} = \boldsymbol{r} - \boldsymbol{r}_{\text{nom}}$, with $\boldsymbol{r}_{\text{nom}}$ being the trajectory obtained by performing the simulation in the nominal case[2], that is with $\Delta \boldsymbol{p} = \boldsymbol{0}$. On the other hand, ST-MPC minimizes the deviation while still being able to utilize the full actuation capabilities. This is

[2]Note that, due to the receding-horizon nature of MPC, in general, this nominal trajectory differs from the nominal predicted trajectory $\bar{\boldsymbol{x}}$.

Table 7.3. Position Error while tracking discontinuous trajectories

Case	Method	RMSE deviation [m] mean (± std. dev.)	Maximum deviation [m]	Nominal RMSE [m]
A	ST-MPC	**0.049 (±0.018)**	**0.112**	0.078
	MPC	0.071 (±0.055)	0.697	**0.062**
A-yaw	ST-MPC	**0.049 (±0.018)**	**0.126**	0.079
	MPC	0.072 (±0.056)	0.697	**0.062**
B	ST-MPC	**0.057 (±0.022)**	**0.241**	0.099
	MPC	0.079 (±0.037)	0.395	**0.070**
B-yaw	ST-MPC	**0.058 (±0.025)**	**0.226**	0.099
	MPC	0.080 (±0.039)	0.414	**0.070**

also due to the fact that the input tube $\boldsymbol{\rho}^{\boldsymbol{u}}$ is by construction zero for the first input $\boldsymbol{u}_0$ of the predicted trajectory. In fact, the tubes which encode the uncertainty over the time horizon $t \in [t_k,\, t_{k+C}]$ starting from the current state $\hat{\boldsymbol{x}}_k$ are always recomputed around the current predicted trajectory $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$: since the state sensitivity $\boldsymbol{\Pi}_0$ of the predicted trajectory is always zero at t_k, it follows that $\boldsymbol{\rho}^{\boldsymbol{u}}_0 = \mathbf{0}$ as well. Therefore, for what concerns the first input $\boldsymbol{u}_0$ at time t_k, the input constraint always reduces to $\boldsymbol{u}_{\min} \leq \boldsymbol{u}_0 \leq \boldsymbol{u}_{\max}$.

Quantitative results are reported in Tab. 7.3, where in all cases the Root Mean Square Error (RMSE) deviation with respect to the nominal trajectory $\boldsymbol{r}_{\text{nom}}$ is smaller in mean and standard deviation for ST-MPC. Similarly, the maximum deviation is about 2-6 times larger for the standard MPC, confirming the better performances of ST-MPC. The additional propeller speeds required to track a time-varying yaw angle (A-yaw, B-yaw) can indeed lead to an increase in the maximum deviation under perturbations. However, we do not observe a clear-cut trend with one method outperforming the other, likely due to some particular correlation between the shape of the trajectory and the occurrence of the maximum deviation point in different instances for the two methods. Similar considerations can also be drawn by analyzing Fig. 7.5 that reports the evolution over time of the deviation from the nominal trajectory $\|\boldsymbol{e}_{\boldsymbol{r}}\|$. Looking at Fig. 7.5, it is clear how the standard MPC experiences larger deviations in all cases and during the whole motion. Finally, we note from the last column of Tab. 7.3 how the RMSE for the tracking of the desired position $\boldsymbol{r}_d$ in the *nominal* (unperturbed) case is larger for ST-MPC: this is expected and due to the trade-off between performance and robustness introduced by the constraint restriction. Still, the overall performance in any non-nominal (real-world) case is in clear favor of ST-MPC.

Passing through a narrow aperture

For this scenario, we performed repeated tests with 450 parameter perturbations $\Delta\boldsymbol{p} \in \mathcal{P}$ in a grid with $\Delta\boldsymbol{p}_{\max} = (0.02, 0.02, 0.01, 0.1)$. As the parameter variations make the system deviate from the predicted trajectory, the MPC will have to progressively compensate to satisfy — at least in its prediction — the safety constraint.

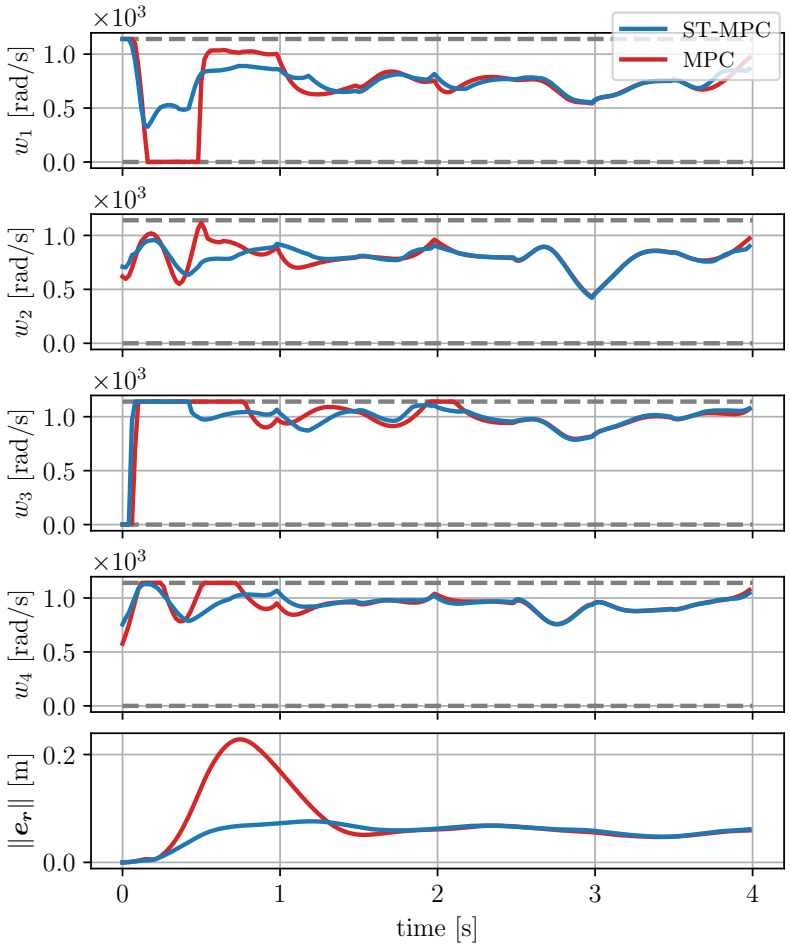

Figure 7.4. Executed propeller speed trajectory (top four) and norm of the deviation from the nominal trajectory (bottom) while tracking trajectory A with $\Delta \boldsymbol{p} = (0.02, -0.02, 0.0, 0.05)$. Dashed grey lines indicate the rotor speed limits. Note how even with the input restriction introduced by the tubes, ST-MPC fully utilizes the actuation capabilities of the drone. See the accompanying video for an animation including the predicted trajectory and tubes.

Since the system possesses limited control authority at any given time, the unforeseen deviation from the plan might lead it to enter states from which no feasible solution exists, even for the nominal prediction model. This emerges as the main cause of failure for the controller, which has no recursive feasibility guarantee [148]. We therefore assess the effectiveness of the proposed method using two metrics:

- The success rate, defined as the percentage of tests in which the Quadrotor is able to safely reach the goal with no collision[3];
- The time t_p at which the Quadrotor has completely passed the aperture, in order to measure the possible motion conservativeness introduced by adding the tube radii restriction on the input constraints. Here we consider only

[3]We consider as failed any test in which either the closed-loop trajectory does not satisfy the constraints or the MPC problem becomes infeasible.

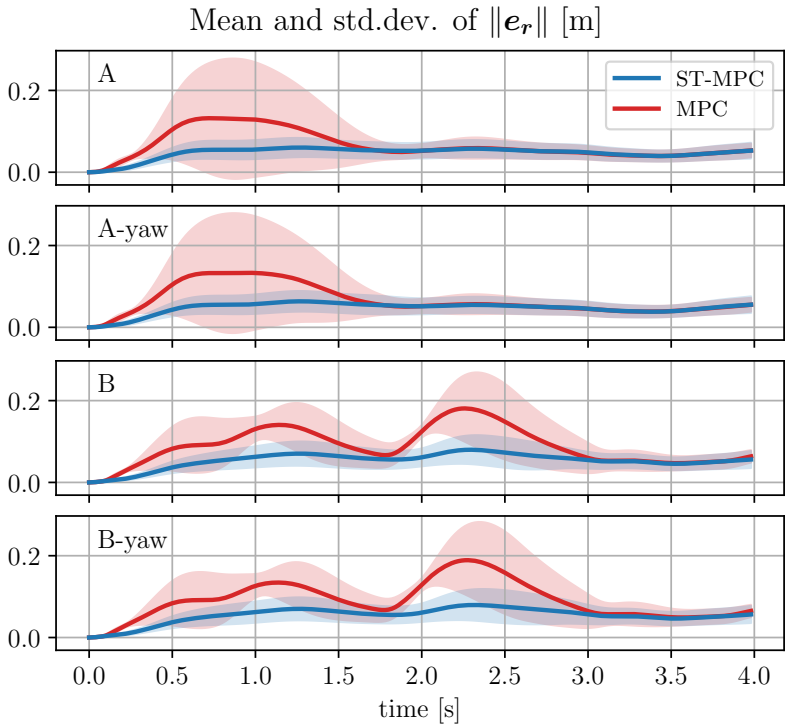

Figure 7.5. Evolution of the mean and the standard deviation (shaded) of the position deviation $\|\boldsymbol{e_r}\|$ from the nominal trajectory over the 450 perturbed simulations for the four tested trajectories. Note how the proposed method achieves a smaller deviation in all cases.

simulations that are successful for both the ST-MPC (also in the LQR variant) and the standard MPC[4].

To analyze the results, we distinguish between two different subsets of parameters: (*i*) $\mathcal{P}_{\text{ie}}$ when $\boldsymbol{\Delta p}$ is inside of the ellipsoid $\mathcal{E}_{\boldsymbol{p}}$, and (*ii*) $\mathcal{P}_{\text{nie}}$ when it is outside. Moreover, we denote as $\mathcal{P}_{\text{nie,nbs}}$ the subset of $\mathcal{P}_{\text{nie}}$ in which at least one of the two controllers fails. The reason for introducing this distinction is that, in some instances, even a large parameter variation lying outside the ellipsoid could result in a motion for which the task is only slightly affected by the disturbance. These favorable configurations result in both controllers completing the task but are not necessarily indicative of the effect of the input tubes on the success rate.

The MPC design is based on nominal performance, i.e., the tuning of the cost function weights is performed by trial and error simulating the nominal system dynamics. Also, the choice of the control horizon C naturally affects the performance and feasibility even in the nominal case, although it is typically restricted by the real-time requirements of the platform. Nonetheless, it is interesting to analyze the increase (or lack thereof) in robustness related to changes in these settings, as they can greatly affect both the nominal and the perturbed behaviors of the system. To

[4]This choice, aimed at performing a one-to-one comparison between the methods, has been found to be slightly pejorative for ST-MPC, due to the exclusion of many data points in which ST-MPC succeeds and the standard MPC does not.

Table 7.4. Success Rate and Time required to pass through the aperture

Method	Settings		Success rate [%]			Time t_p [s]
	C	Q_v	$\mathcal{P}_{\text{ie}}$	$\mathcal{P}_{\text{nie}}$ ($\mathcal{P}_{\text{nie,nbs}}$)	total	mean (± std. dev.)
ST-MPC	25	$2 \cdot 10^{-2}$	90.5	62.5 (31.3)	75.6	0.677 (±0.032)
		$5 \cdot 10^{-2}$	99.5	84.2 (67.0)	91.3	0.730 (±0.039)
	35	$2 \cdot 10^{-2}$	**100**	92.5 (85.5)	**96.0**	0.721 (±0.032)
		$5 \cdot 10^{-2}$	97.1	**92.9 (85.8)**	94.9	0.732 (±0.037)
MPC	25	$2 \cdot 10^{-2}$	59.0	45.4 (0.0)	51.8	**0.652 (±0.044)**
		$5 \cdot 10^{-2}$	68.6	52.1 (0.0)	59.8	0.695 (±0.049)
	35	$2 \cdot 10^{-2}$	64.8	48.3 (0.0)	56.0	0.673 (±0.040)
		$5 \cdot 10^{-2}$	67.6	50.4 (0.8)	58.4	0.682 (±0.041)
ST-MPC-LQR	25	$2 \cdot 10^{-2}$	75.2	54.6 (16.8)	64.2	0.668 (±0.031)
		$5 \cdot 10^{-2}$	94.8	69.2 (35.7)	81.1	0.724 (±0.036)

this end, Tab. 7.4 reports statistics on the success rates and times t_p with different settings for the velocity weight $\boldsymbol{Q}_v = Q_v \boldsymbol{I}_3$ and the control horizon C. Three different methods are compared: (*i*) ST-MPC, i.e., the proposed method, (*ii*) a standard MPC, and (*iii*) ST-MPC-LQR obtained, as explained, by neglecting all constraints in the MPC gains computation.

From the analysis of Tab. 7.4, we can appreciate how the introduction of the input tubes in ST-MPC is always followed by a significant improvement in robustness, increasing from the 60-70% range to 90-100% when the parameter deviation is in $\mathcal{E}_{\boldsymbol{p}}$ (case $\mathcal{P}_{\text{ie}}$). Moreover, when the parameter deviation falls outside the ellipsoid (cases $\mathcal{P}_{\text{nie}}$ and $\mathcal{P}_{\text{nie,nbs}}$), all methods experience a lower success rate, confirming that the satisfaction of the ellipsoid condition is important for the task success. This also shows that the approximation (6.14) used for deriving the tubes is, in fact, not too conservative since the effect of parameter deviations lying outside the ellipsoid (6.13) are (correctly) not captured by the tubes.

Increasing the control horizon C from 25 to 35 leads to a marginal increase in the success rate for both ST-MPC and the standard MPC. However, while ST-MPC reaches almost 100% success rate, the standard MPC is limited to less than 60%, indicating that the longer control horizon is not the main factor for substantially improving the recursive feasibility of the system. Similarly, the increase of the velocity weight Q_v from $2{\cdot}10^{-2}$ to $5{\cdot}10^{-2}$ does not provide significant improvements, although it would intuitively yield slower and more conservative trajectories (as confirmed by the statistics on t_p), and thus appear as an almost obvious means to improve safety.

Concerning the performance in terms of speed and agility, we evaluate the time t_p at which the Quadrotor clears the aperture, independently of the fact that the parameter variation is inside the ellipsoid or not. Although ST-MPC does indeed generate a slightly more conservative motion with a $\sim 6\%$ increase (at most) in the average t_p, this is met with an important improvement in the success rate, as already discussed. Moreover, we note how for at least one setting ($C = 25$, $Q_v = 2 \cdot 10^{-2}$)

ST-MPC has a performance comparable to that of the standard MPC while still providing more safety.

Lastly, the direct comparison between ST-MPC and its simplified variant ST-MPC-LQR (performed with the $C = 25$ setting) shows how the latter is not able to perform as well as ST-MPC with a total success rate reduced by 10% in all instances and with a 15% reduction in the success rate for $\mathcal{P}_{\text{ie}}$ in the $Q_v = 2{\cdot}10^{-2}$ setting. This is expected since ST-MPC-LQR does not account for the active obstacle avoidance constraints (7.16) when computing the MPC gains, which does not allow to correctly capture the behavior of the MPC in the presence of active constraints. This result then further motivates the use of the proposed ST-MPC scheme.

In conclusion, it is clear how the proposed method can provide a substantial improvement on the safety of the system with parameter uncertainty, which can be traded for a reduction in performance (in terms of speed to execute the task), while standard ways of robustifying the standard MPC, such as tuning the cost function or increasing the control horizon, are not able to obtain the same results.

7.4.4 Implementation details

The proposed method has been implemented in C++ on a 11th Gen Intel® Core™ i7-11700 @ 3.60GHz by making use of the autodiff library [149] for Automatic Differentiation (AD) and using the ProxQP sparse solver [31]. To solve the linear system (7.4) yielding (7.5) we employ Eigen's linear solver to perform a sparse LU factorization. Considering the most demanding scenario (passing through the aperture) with $C = 25$, one iteration of the algorithm runs in real-time at 50 Hz ($\delta_t = 0.02$ s), with the MPC gains and tubes computation only requiring 3-4 ms, compared to the 1-10 ms to solve the QP during the feedback phase. Being the structure of the OCP and the number of decision variables and constraints the same, the only additional computational cost of ST-MPC compared to the standard MPC is due to the computation of the tubes. This takes a few milliseconds in the current implementation, thus making ST-MPC suitable for real-time use. Moreover, one could further improve the current sensitivity computation, by leveraging Eigen's sparse linear solvers and more tightly integrating it with the QP solver for reusing the KKT factorization performed by the solver itself or by using differentiable QP solvers [150]. Additionally, it could be possible to exploit the structure of the optimal control problem to compute the MPC gains more efficiently by adapting, for instance, the work proposed in [142]. The sensitivity propagation can, on the other hand, reuse the derivatives of the dynamics that have been computed to prepare the QP. Compared to the current implementation based on AD, significant gains on the computation of the derivatives of the dynamics could be expected by switching to a library such as Pinocchio [83], particularly if more complex articulated robots are considered.

Chapter 8

Conclusions

This thesis presented a series of motion generation strategies aimed at addressing two fundamental issues arising in several robotics applications: (*i*) the instability of systems that exhibit a non-minimum phase behavior, which has been addressed through the application of the IS-MPC method; (*ii*) the robustness of the system evolution against uncertainties in the model parameters, which has been shown to be effectively described by the closed-loop sensitivity metric that has then been exploited in the novel ST-MPC method.

To conclude, we recall the main results of Chapters 4, 5, 6, 7 and provide some insight into possible future research directions.

Chapter 4 presented a general output tracking MPC controller for a WIP balancing robot with arms, which utilizes IS-MPC to handle the unstable pitch dynamics. We have discussed its application to navigation and loco-manipulation tasks, and validated its performance by simulations in such scenarios. Results show that the proposed approach is able to generate stable motions that guarantee accurate task tracking. Still, the method in its current form presents the following shortcomings:

- the working environment being limited to flat and even ground;
- the performance being limited by the use of a truncated tail.

To improve on the former, it could be possible to explicitly model the slope of the ground to traverse e.g., ramps, and to robustify the method against disturbances in the ground modeling by applying the methodology of [151]. Moreover, if the controlled robot is equipped with legs (see, e.g., [152]), it could be possible to extend the method to perform jumps, allowing the system to overcome steps and small gaps. Concerning the latter, it could be straightforward to include an anticipative tail for the navigation task, similarly to the planar WIP example of Sect. 3.3.1, which has not been done to keep the formulation generic. When more complex tasks are involved, it could be possible to include a pre-computed *motion library* [153] to pick the most suitable trajectory from — albeit going against the *online* nature of the method. Moreover, possible future work include the experimental validation, a nonlinear formulation of the stability constraint based on [59], and the analysis of recursive feasibility.

Chapter 5 introduced a control approach which actively prevents the onset of jackknifing during backward trajectory tracking for Tractor-Trailer vehicles. In particular, a feedback control law was designed as the combination of two actions: a tracking term, computed using input-output linearization, and a corrective term, generated via IS-MPC. The proposed method has been first verified in simulation and then experimentally validated on a purposely built prototype. To show the generality of the proposed approach, we have also presented successful experimental results obtained for a two-trailer vehicle.

This work can be extended in several directions, among which we mention:

- designing a robust version of the anti-jackknifing controller which can handle external disturbances, following the ideas in [151];
- modifying the IS-MPC formulation in order to perform obstacle avoidance during tracking;
- studying the applicability of the proposed approach for counteracting the dynamic jackknife phenomenon associated to wheel slippage in high-speed forward motion.

Chapter 6 first introduced the closed-loop sensitivity concept, allowing to conveniently express the effect that uncertainties in the model parameters have on the closed-loop trajectories of a control system. We have illustrated how these quantities can be used to construct ellipsoidal tubes which approximately envelop the ensemble of possible trajectories that are generated when the model is perturbed. From these tubes, it is then possible to compute the directional radius representing the worst-case deviation of the perturbed trajectory along a direction of interest, which can then be used to robustify constraints against, e.g, input saturation. We have then experimentally validated an offline trajectory optimization problem in which the effect of parametric uncertainties in a quadrotor model (quantified by the notion of *state sensitivity*) can be minimized by acting on the reference trajectory to be tracked. The experimental results demonstrate the effectiveness of the proposed optimization in reducing the effects of uncertainties stemming from physical parameter perturbations, enabling more precise target attainment at relatively high speeds. We also showcased the significance of *input sensitivity* within the constraints of our optimization problem. Indeed, by using the input tubes in the constraints, one can ensure the robustness against unplanned input saturation due to parametric uncertainties.

Chapter 7 introduced an efficient Robust Model Predictive Control algorithm, denoted ST-MPC, for robots affected by parametric uncertainties, and showed its effectiveness in improving the tracking performance and the success rate during navigation in a tight environment. By leveraging the notion of closed-loop sensitivity and ellipsoidal tubes for enveloping the perturbed trajectories, introduced in Chapter 6, we were able to introduce a time-varying restriction on the input constraints to make, at each instant, the MPC controller aware of the possible additional input requirements needed to cope with parametric uncertainties. The resulting ST-MPC has the same computational complexity as a standard MPC, only adding the computation of the MPC gains and tube propagation during consecutive control instants

by analyzing the sensitivity of the previous MPC solution. Since this computation is performed in the *preparation* phase, it does not introduce any additional delay.

Being the first *online* application of robust trajectory optimization based on the closed-loop sensitivity, it is possible to extend the method in several directions. For instance:

- including an online optimization of a sensitivity metric for generating trajectories that are minimally sensitive and, thus, reduce the tube radius;
- adding online parameter estimation schemes in the sensitivity calculation for updating the nominal parameter values and thus reduce the conservativeness of the motion.

From a computational point of view, a tight integration with the QP solver of choice would allow to streamline the calculation of the sensitivity, strengthening even more the ability to apply the method at a very little computational cost.

Bibliography

[1] D. Bicego, J. Mazzetto, R. Carli, M. Farina, and A. Franchi, "Nonlinear model predictive control with enhanced actuator model for multi-rotor aerial vehicles with generic designs," *Journal of Intelligent & Robotic Systems*, vol. 100, pp. 1213–1247, 2020.

[2] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3357–3373, Dec. 2022.

[3] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[4] V. Vulcano, S. G. Tarantos, P. Ferrari, and G. Oriolo, "Safe robot navigation in a crowd combining nmpc and control barrier functions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 3321–3328.

[5] S. G. Tarantos, T. Belvedere, and G. Oriolo, "Dynamics-aware navigation among moving obstacles with application to ground and flying robots," *Robotics and Autonomous Systems*, vol. 172, p. 104582, 2024.

[6] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback mpc for torque-controlled legged robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4730–4737.

[7] J. Carpentier and P.-B. Wieber, "Recent progress in legged robots locomotion control," *Current Robotics Reports*, vol. 2, no. 3, pp. 231–238, 2021.

[8] G. Romualdi, S. Dafarra, G. L'Erario, I. Sorrentino, S. Traversaro, and D. Pucci, "Online non-linear centroidal mpc for humanoid robot locomotion with step adjustment," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10 412–10 419.

[9] M. Kanneworff, T. Belvedere, N. Scianca, F. M. Smaldone, L. Lanari, and G. Oriolo, "Task-oriented generation of stable motions for wheeled inverted pendulum robots," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 214–220.

[10] M. Beglini, T. Belvedere, L. Lanari, and G. Oriolo, "An intrinsically stable MPC approach for anti-jackknifing control of tractor-trailer vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 6, pp. 4417–4428, 2022.

[11] A. Srour, S. Marcellini, T. Belvedere, M. Cognetti, A. Franchi, and P. Robuffo Giordano, "Experimental validation of sensitivity-aware trajectory planning for a quadrotor uav under parametric uncertainty," in *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2024.

[12] T. Belvedere, M. Cognetti, G. Oriolo, and P. Robuffo Giordano, "Sensitivity-aware Model Predictive Control for robots with parametric uncertainty," *IEEE Transactions on Robotics*, 2024, (submitted).

[13] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913.

[14] D. Liberzon, *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2012.

[15] J. Yong and X. Y. Zhou, *Stochastic controls: Hamiltonian systems and HJB equations*. Springer Science & Business Media, 1999, vol. 43.

[16] R. F. Hartl, S. P. Sethi, and R. G. Vickson, "A survey of the maximum principles for optimal control problems with state constraints," *SIAM review*, vol. 37, no. 2, pp. 181–218, 1995.

[17] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. Society for Industrial and Applied Mathematics, Jan. 2010.

[18] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," *Fast motions in biomechanics and robotics: optimization and feedback control*, pp. 65–93, 2006.

[19] T. H. Tsang, D. M. Himmelblau, and T. F. Edgar, "Optimal control via collocation and non-linear programming," *International Journal of Control*, vol. 21, no. 5, p. 763–768, May 1975.

[20] R. W. H. Sargent and G. R. Sullivan, "The development of an efficient optimal control package," in *Optimization Techniques*, ser. Lecture Notes in Control and Information Sciences, J. Stoer, Ed. Berlin, Heidelberg: Springer, 1978, p. 158–168.

[21] H. Bock and K. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, p. 1603–1608, July 1984.

[22] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.

[23] T. Ohtsuka, "A continuation/gmres method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40, no. 4, pp. 563–574, Apr. 2004.

[24] J. V. Kadam and W. Marquardt, "Sensitivity-based solution updates in closed-loop dynamic optimization," *IFAC Proceedings Volumes*, vol. 37, no. 9, pp. 947–952, July 2004.

[25] L. Biegler, X. Yang, and G. Fischer, "Advances in sensitivity-based nonlinear model predictive control and dynamic real-time optimization," *Journal of Process Control*, vol. 30, pp. 104–116, June 2015.

[26] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear mpc: bridging the gap via the real-time iteration," *International Journal of Control*, vol. 93, no. 1, pp. 62–80, 2020.

[27] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge, UK; New York: Cambridge University Press, 2004.

[28] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear mpc and moving horizon estimation," in *Nonlinear model predictive control: towards new challenging applications*, L. Magni, D. M. Raimondo, and F. Allgöwer, Eds. Springer, 2009, pp. 391–417.

[29] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[30] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[31] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond," in *RSS 2022 - Robotics: Science and Systems*, New York, United States, June 2022.

[32] M. Diehl, "Real-time optimization for large scale nonlinear processes," Ph.D. dissertation, University of Heidelberg, 2001.

[33] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[34] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust Region Methods*. Society for Industrial and Applied Mathematics, 2000.

[35] A. L. Tits, A. Wächter, S. Bakhtiari, T. J. Urban, and C. T. Lawrence, "A primal-dual interior-point method for nonlinear programming with strong global and local convergence properties," *SIAM Journal on Optimization*, vol. 14, no. 1, pp. 173–199, 2003.

[36] R. Fletcher and S. Leyffer, "Nonlinear programming without a penalty function," *Mathematical programming*, vol. 91, pp. 239–269, 2002.

[37] R. Quirynen, B. Houska, M. Vallerio, D. Telen, F. Logist, J. Van Impe, and M. Diehl, "Symmetric algorithmic differentiation based exact hessian sqp method and software for economic mpc," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 2752–2757.

[38] W. Huang, X. Huang, C. Majidi, and M. K. Jawed, "Dynamic simulation of articulated soft robots," *Nature communications*, vol. 11, no. 1, p. 2233, 2020.

[39] K. Erleben and S. Andrews, "Solving inverse kinematics using exact hessian matrices," *Computers & Graphics*, vol. 78, pp. 1–11, 2019.

[40] K. Pfeiffer, "Efficient Kinematic and Algorithmic Singularity Resolution for Multi-Contact and Multi-Level Constrained Dynamic Robot Control," Theses, Université Montpellier, Dec. 2019. [Online]. Available: https://hal.science/tel-02867294

[41] M. Khadem, J. O'Neill, Z. Mitros, L. d. Cruz, and C. Bergeles, "Autonomous steering of concentric tube robots for enhanced force/velocity manipulability," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2197–2204.

[42] M. Giftthaler, M. Neunert, M. Stauble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, Oct. 2018, pp. 1–9.

[43] I. K. Erunsal, R. Ventura, and A. Martinoli, "Nonlinear model predictive control for formations of multi-rotor micro aerial vehicles: An experimental approach," in *Experimental Robotics (ISER 2020)*, ser. Springer Proceedings in Advanced Robotics. 19, B. Siciliano, C. Laschi, and O. Khatib, Eds. Springer, 2021, pp. 449–461.

[44] 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, no. 1, pp. 43–53, 1987.

[45] P.-B. Wieber, R. Tedrake, and S. Kuindersma, *Modeling and Control of Legged Robots*. Springer International Publishing, 2016, pp. 1203–1234.

[46] A. Isidori and C. H. Moog, *On the nonlinear equivalent of the notion of transmission zeros*, ser. Lecture Notes in Control and Information Sciences. Berlin/Heidelberg: Springer-Verlag, 1988, vol. 105, p. 146–158.

[47] C. Byrnes, A. Isidori, and J. Willems, "Passivity, feedback equivalence, and the global stabilization of minimum phase nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 36, no. 11, pp. 1228–1240, Nov. 1991.

[48] J. Chen, S. Fang, and H. Ishii, "Fundamental limitations and intrinsic limits of feedback: An overview in an information age," *Annual Reviews in Control*, vol. 47, pp. 155–177, 2019.

[49] A. De Luca, S. Panzieri, and G. Ulivi, "Stable inversion control for flexible link manipulators," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 1, May 1998, pp. 799–805 vol.1.

[50] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo, "MPC for humanoid gait generation: Stability and feasibility," *IEEE Trans. on Robotics*, vol. 36, no. 4, pp. 1171–1188, 2020.

[51] P.-B. Wieber, "Viability and predictive control for safe locomotion," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nice: IEEE, Sept. 2008, pp. 1103–1108.

[52] A. Isidori, *Nonlinear Control Systems*. Springer, 1995.

[53] ——, "The zero dynamics of a nonlinear system: From the origin to the latest progresses of a long successful story," *European Journal of Control*, vol. 19, no. 5, pp. 369–378, Sept. 2013.

[54] J. B. Hoagg and D. S. Bernstein, "Nonminimum-phase zeros - much to do about nothing - classical control - revisited part ii," *IEEE Control Systems Magazine*, vol. 27, no. 3, pp. 45–57, 2007.

[55] P. Moylan, "Stable inversion of linear systems," *IEEE Transactions on Automatic Control*, vol. 22, no. 1, pp. 74–78, Feb. 1977.

[56] L. Lanari and J. Wen, "Feedforward calculation in tracking control of flexible robots," in *1991 Proceedings of the 30th IEEE Conference on Decision and Control*. Brighton, UK: IEEE, 1991, pp. 1403–1408.

[57] S. Devasia and B. Paden, "Exact output tracking for nonlinear time-varying systems," in *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, vol. 3, Dec. 1994, pp. 2346–2355 vol.3.

[58] S. Devasia, D. Chen, and B. Paden, "Nonlinear inversion-based output tracking," *IEEE Transactions on Automatic Control*, vol. 41, no. 7, pp. 930–942, July 1996.

[59] L. Hunt and G. Meyer, "Stable inversion for nonlinear systems," *Automatica*, vol. 33, no. 8, pp. 1549–1554, Aug. 1997.

[60] S. Devasia and B. Paden, "Stable inversion for nonlinear nonminimum-phase time-varying systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 2, pp. 283–288, Feb. 1998.

[61] L. Lanari, S. Hutchinson, and L. Marchionni, "Boundedness issues in planning of locomotion trajectories for biped robots," in *2014 IEEE-RAS International Conference on Humanoid Robots*. Madrid, Spain: IEEE, Nov. 2014, pp. 951–958.

[62] S. Caron, A. Escande, L. Lanari, and B. Mallein, "Capturability-based pattern generation for walking with variable height," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 517–536, 2020.

[63] W. A. Coppel, "Dichotomies and reducibility," *Journal of Differential Equations*, vol. 3, no. 4, pp. 500–521, Oct. 1967.

[64] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *6th IEEE-RAS Int. Conf. on Humanoid Robots*, 2006, pp. 200–207.

[65] T. Takenaka, T. Matsumoto, and T. Yoshiike, "Real time motion generation and control for biped robot - 1st report: Walking gait pattern generation," in *2009 Int. Conf. on Intelligent Robots and Systems*, 2009, pp. 1084–1091.

[66] S. Devasia, "Output tracking with nonhyperbolic and near nonhyperbolic internal dynamics: Helicopter hover control," *J. of guidance, control, and dynamics*, vol. 20, no. 3, pp. 573–580, 1997.

[67] H. G. Nguyen, J. Morrell, K. D. Mullens, A. B. Burmeister, S. Miles, N. Farrington, K. M. Thomas, and D. W. Gage, "Segway robotic mobility platform," in *Mobile Robots XVII*, vol. 5609. SPIE, 2004, pp. 207–220.

[68] R. O. Ambrose, R. T. Savely, S. M. Goza, P. Strawser, M. A. Diftler, I. Spain, and N. Radford, "Mobile manipulation using NASA's Robonaut," in *2004 IEEE Int. Conf. on Robotics and Automation*, 2004, pp. 2104–2109.

[69] P. Deegan, B. J. Thibodeau, and R. Grupen, "Designing a self-stabilizing robot for dynamic mobile manipulation," Massachusetts University Amherst, Dept. of Computer Science, Tech. Rep., 01 2006.

[70] S. Jeong and T. Takahashi, "Wheeled inverted pendulum type assistant robot: design concept and mobile control," *Intelligent Service Robotics*, vol. 1, no. 4, pp. 313–320, 2008.

[71] M. Stilman, J. Olson, and W. Gloss, "Golem Krang: Dynamically stable humanoid robot for mobile manipulation," in *2010 IEEE Int. Conf. on Robotics and Automation*, 2010, pp. 3304–3309.

[72] G. Lentini, A. Settimi, D. Caporale, M. Garabini, G. Grioli, L. Pallottino, M. G. Catalano, and A. Bicchi, "ALTER-EGO: a mobile robot with a functionally anthropomorphic upper body designed for physical interaction," *IEEE Robotics & Automation Magazine*, vol. 26, no. 4, pp. 94–107, 2019.

[73] Y. Zhao, C. Woo, and J. Lee, "Balancing control of mobile manipulator with sliding mode controller," in *15th Int. Conf. on Control, Automation and Systems*, 2015, pp. 802–805.

[74] Y.-G. Bae and S. Jung, "Balancing control of a mobile manipulator with two wheels by an acceleration-based disturbance observer," *Int. Journal of Humanoid Robotics*, vol. 15, no. 03, p. 1850005, 2018.

[75] G. Zambella, G. Lentini, M. Garabini, G. Grioli, M. G. Catalano, A. Palleschi, L. Pallottino, A. Bicchi, A. Settimi, and D. Caporale, "Dynamic whole-body control of unstable wheeled humanoid robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3489–3496, 2019.

[76] C. Acar and T. Murakami, "Multi-task control for dynamically balanced two-wheeled mobile manipulator through task-priority," in *2011 IEEE Int. Symposium on Industrial Electronics*, 2011, pp. 2195–2200.

[77] M. Zafar and H. I. Christensen, "Whole body control of a wheeled inverted pendulum humanoid," in *16th IEEE-RAS Int. Conf. on Humanoid Robots*, 2016, pp. 89–94.

[78] M. Yue, C. An, and J.-Z. Sun, "An efficient model predictive control for trajectory tracking of wheeled inverted pendulum vehicles with various physical constraints," *Int. Journal of Control, Automation and Systems*, vol. 16, no. 1, pp. 265–274, 2018.

[79] M. Zafar, S. Hutchinson, and E. A. Theodorou, "Hierarchical optimization for whole-body control of wheeled inverted pendulum humanoids," in *2019 IEEE Int. Conf. on Robotics and Automation*, 2019, pp. 7535–7542.

[80] M. V. Minniti, F. Farshidian, R. Grandia, and M. Hutter, "Whole-body MPC for a dynamically stable mobile manipulator," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3687–3694, 2019.

[81] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2009.

[82] S. Kim and S. Kwon, "Dynamic modeling of a two-wheeled inverted pendulum balancing mobile robot," *Int. Journal of Control, Automation and Systems*, vol. 13, no. 4, pp. 926–933, 2015.

[83] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[84] U. Nagarajan and R. Hollis, "Shape space planner for shape-accelerated balancing mobile robots," *The Int. Journal of Robotics Research*, vol. 32, no. 11, pp. 1323–1341, 2013.

[85] M. M. Michałek, "Agile maneuvering with intelligent articulated vehicles: a control perspective," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 458–473, 2019.

[86] F. Lamiraux and J. P. Laumond, "A practical approach to feedback control for a mobile robot with trailer," in *1998 IEEE Int. Conf. on Robotics and Automation*, 1998, pp. 3291–3295.

[87] A. González-Cantos and A. Ollero, "Backing-up maneuvers of autonomous tractor-trailer vehicles using the qualitative theory of nonlinear dynamical systems," *The Int. J. of Robotics Research*, vol. 28, no. 1, pp. 49–65, 2009.

[88] W. Chung, M. Park, K. Yoo, J. I. Roh, and J. Choi, "Backward-motion control of a mobile robot with n passive off-hooked trailers," *J. of Mechanical Science and Technology*, vol. 25, no. 11, pp. 2895–2905, 2011.

[89] J. Morales, J. L. Martínez, A. Mandow, and A. J. García-Cerezo, "Steering the last trailer as a virtual tractor for reversing vehicles with passive on-and off-axle hitches," *IEEE Trans. on Industrial Electronics*, vol. 60, no. 12, pp. 5729–5736, 2013.

[90] Z. Leng and M. A. Minor, "Curvature-based ground vehicle control of trailer path following considering sideslip and limited steering actuation," *IEEE Trans. on Intelligent Transportation Systems*, vol. 18, no. 2, pp. 332–348, 2016.

[91] M. Sampei, T. Tamura, T. Kobayashi, and N. Shibui, "Arbitrary path tracking control of articulated vehicles using nonlinear control theory," *IEEE Trans. on Control System Technology*, vol. 3, no. 1, pp. 125–131, 1995.

[92] R. M. De Santis, "Path-tracking for articulated vehicles via exact and jacobian linearization," in *IFAC Intelligent Autonomous Vehicles*, 1998, pp. 159–164.

[93] A. Astolfi, P. Bolzern, and A. Locatelli, "Path-tracking of a tractor-trailer vehicle along rectilinear and circular paths: A Lyapunov-based approach," *IEEE Trans. on Robotics and Automation*, vol. 20, no. 1, pp. 154–160, 2004.

[94] P. Bolzern, R. M. De Santis, A. Locatelli, and D. Masciocchi, "Path-tracking for articulated vehicles with off-axle hitching," *IEEE Trans. on Control System Technology*, vol. 6, no. 4, pp. 515–523, 1998.

[95] C. Altafini, A. Speranzon, and B. Wahlberg, "A feedback control scheme for reversing a truck and trailer vehicle," *IEEE Trans. on Robotics and Automation*, vol. 17, no. 6, pp. 915–922, 2001.

[96] C. Pradalier and K. Usher, "A simple and efficient control scheme to reverse a tractor-trailer system on a trajectory," in *2007 IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 2208–2214.

[97] O. Ljungqvist, N. Evestedt, D. Axehill, M. Cirillo, and H. Pettersson, "A path planning and path-following control framework for a general 2-trailer with a car-like tractor," *J. of Field Robotics*, vol. 36, no. 8, pp. 1345–1377, 2019.

[98] E. Kayacan, H. Ramon, and W. Saeys, "Robust trajectory tracking error model-based predictive control for unmanned ground vehicles," *IEEE/ASME Trans. on Mechatronics*, vol. 21, no. 2, pp. 806–814, 2015.

[99] H. Guo, C. Shen, H. Zhang, H. Chen, and R. Jia, "Simultaneous trajectory planning and tracking using an MPC method for cyber-physical systems: A case study of obstacle avoidance for an intelligent vehicle," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 9, pp. 4273–4283, 2018.

[100] O. Ljungqvist, D. Axehill, and H. Pettersson, "On sensing-aware model predictive path-following control for a reversing general 2-trailer with a car-like tractor," in *2020 IEEE Int. Conf. on Robotics and Automation*, 2020, pp. 8813–8819.

[101] C. Sentouh, A. Nguyen, M. A. Benloucif, and J. Popieul, "Driver-automation cooperation oriented approach for shared control of lane keeping assist systems," *IEEE Trans. on Control System Technology*, vol. 27, no. 5, pp. 1962–1978, 2019.

[102] K. J. Astrom and B. Wittenmark, "Adaptive control 2nd edition," *Addison-Wesley Pub Co.*, vol. 1994, 1994.

[103] M. Achtelik, T. Bierling, J. Wang, L. Höcht, and F. Holzapfel, "Adaptive control of a quadcopter in the presence of large/complete parameter uncertainties," in *Infotech@ Aerospace 2011*, 2011, p. 1485.

[104] A. Haseltalab and R. R. Negenborn, "Adaptive control for autonomous ships with uncertain model and unknown propeller dynamics," *Control Engineering Practice*, vol. 91, p. 104116, 2019.

[105] K. Zhou and J. C. Doyle, *Essentials of robust control*. Prentice hall Upper Saddle River, NJ, 1998, vol. 104.

[106] R. Sanz, P. Garcia, Q.-C. Zhong, and P. Albertos, "Robust control of quadrotors based on an uncertainty and disturbance estimator," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 7, p. 071006, 2016.

[107] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2021.

[108] A. Ansari and T. Murphey, "Minimum Sensitivity Control for Planning with Parametric and Hybrid Uncertainty," *The International Journal of Robotics Research (IJRR)*, vol. 35, no. 7, pp. 823–839, October 2016.

[109] S. Candido and S. Hutchinson, "Minimum Uncertainty Robot Path Planning using a POMDP Approach," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, December 2010, pp. 1408–1413.

[110] ——, "Minimum uncertainty robot navigation using information-guided pomdp planning," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 6102–6108.

[111] P. Robuffo Giordano, Q. Delamare, and A. Franchi, "Trajectory generation for minimum closed-loop state sensitivity," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 286–293.

[112] P. Brault, Q. Delamare, and P. Robuffo Giordano, "Robust trajectory planning with parametric uncertainties," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 095–11 101.

[113] C. Bohm, P. Brault, Q. Delamare, P. Robuffo Giordano, and S. Weiss, "COP: Control & Observability-aware Planning," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 3364–3370.

[114] S. Wasiela, P. Robuffo Giordano, J. Cortes, and T. Simeon, "A Sensitivity-Aware Motion Planner (SAMP) to Generate Intrinsically-Robust Trajectories," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 12 707–12 713.

[115] A. Srour, A. Franchi, and P. Robuffo Giordano, "Controller and Trajectory Optimization for a Quadrotor UAV with Parametric Uncertainty," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 1–7.

[116] A. Pupa, P. Robuffo Giordano, and C. Secchi, "Optimal energy tank initialization for minimum sensitivity to model uncertainties," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 8192–8199.

[117] G. Antonelli, E. Cataldi, F. Arrichiello, P. Robuffo Giordano, S. Chiaverini, and A. Franchi, "Adaptive Trajectory Tracking for Quadrotor MAVs in Presence of Parameter Uncertainties and External Disturbances," *IEEE Trans. on Control Systems Technology*, vol. 26, no. 1, pp. 248–254, 2018.

[118] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[119] M. J. Van Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control*, vol. 8, no. 11, pp. 995–1020, 1998.

[120] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control: Technical report," ETH Zurich, Tech. Rep., 2013.

[121] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Springer Science & Business Media, 2007, vol. 37.

[122] F. Zhou, B. Song, and G. Tian, "Bézier curve based smooth path planning for mobile robot," *Journal of Information & Computational Science*, vol. 8, no. 12, pp. 2441–2450, December 2011.

[123] A. R. Conn, K. Scheinberg, and P. L. Toint, "On the convergence of derivative-free methods for unconstrained optimization," *Approximation theory and optimization: tributes to MJD Powell*, pp. 83–108, 1997.

[124] G. Ansmann, "Efficiently and easily integrating differential equations with jitcode, jitcdde, and jitcsde," *Chaos: An interdisciplinary journal of nonlinear science*, vol. 28, no. 4, 2018.

[125] V. M. Zavala and L. T. Biegler, "The advanced-step nmpc controller: Optimality, stability and robustness," *Automatica*, vol. 45, no. 1, p. 86–93, Jan. 2009.

[126] B. Houska and M. E. Villanueva, *Robust optimization for MPC*. Springer, 2019.

[127] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[128] D. Q. Mayne, E. C. Kerrigan, E. Van Wyk, and P. Falugi, "Tube-based robust nonlinear model predictive control," *International journal of robust and nonlinear control*, vol. 21, no. 11, pp. 1341–1353, 2011.

[129] M. Cannon, J. Buerger, B. Kouvaritakis, and S. Rakovic, "Robust Tubes in Nonlinear Model Predictive Control," *IEEE Transactions on Automatic Control*, vol. 56, no. 8, pp. 1942–1947, Aug. 2011.

[130] G. Garimella, M. Sheckells, J. L. Moore, and M. Kobilarov, "Robust obstacle avoidance using tube nmpc." in *Robotics: Science and Systems*, 2018.

[131] J. Köhler, R. Soloperto, M. A. Müller, and F. Allgöwer, "A Computationally Efficient Robust Model Predictive Control Framework for Uncertain Nonlinear Systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 2, pp. 794–801, Feb. 2021.

[132] Y. Gao, F. Messerer, J. Frey, N. v. Duijkeren, and M. Diehl, "Collision-free Motion Planning for Mobile Robots by Zero-order Robust Optimization-based MPC," in *2023 European Control Conference (ECC)*, June 2023, pp. 1–6.

[133] A. Zanelli, J. Frey, F. Messerer, and M. Diehl, "Zero-Order Robust Nonlinear Model Predictive Control with Ellipsoidal Uncertainty Sets," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 50–57, 2021.

[134] B. T. Lopez, J.-J. E. Slotine, and J. P. How, "Dynamic Tube MPC for Nonlinear Systems," in *2019 American Control Conference (ACC)*, July 2019, pp. 1655–1662.

[135] X. Feng, S. Di Cairano, and R. Quirynen, "Inexact adjoint-based sqp algorithm for real-time stochastic nonlinear mpc," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6529–6535, 2020.

[136] M. E. Villanueva, R. Quirynen, M. Diehl, B. Chachuat, and B. Houska, "Robust mpc via min–max differential inequalities," *Automatica*, vol. 77, p. 311–321, Mar. 2017.

[137] A. V. Fiacco, *Introduction to sensitivity and stability analysis in non linear programming*. New York: Academic Press, 1983.

[138] A. Shapiro, "Sensitivity Analysis of Nonlinear Programs and Differentiability Properties of Metric Projections," *SIAM J. Control Optim.*, vol. 26, no. 3, pp. 628–645, May 1988, publisher: Society for Industrial and Applied Mathematics.

[139] M. Zanon, V. Kungurtsev, and S. Gros, "Reinforcement Learning Based on Real-Time Iteration NMPC," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5213–5218, 2020.

[140] J. Jäschke, X. Yang, and L. T. Biegler, "Fast economic model predictive control based on nlp-sensitivities," *Journal of Process Control*, vol. 24, no. 8, p. 1260–1272, Aug. 2014.

[141] E. Dantec, M. Taïx, and N. Mansard, "First Order Approximation of Model Predictive Control Solutions for High Frequency Feedback," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4448–4455, Apr. 2022.

[142] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 93–100.

[143] C. Büskens and H. Maurer, *Sensitivity Analysis and Real-Time Optimization of Parametric Nonlinear Programming Problems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 3–16.

[144] H. V. Henderson and S. R. Searle, "On Deriving the Inverse of a Sum of Matrices," *SIAM Rev.*, vol. 23, no. 1, pp. 53–60, Jan. 1981, publisher: Society for Industrial and Applied Mathematics.

[145] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and Sparse Resolution of Constrained Dynamic Equations," in *Robotics: Science and Systems XVII.* Robotics: Science and Systems Foundation, July 2021.

[146] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and trends in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[147] D. P. Bertsekas, "On penalty and multiplier methods for constrained minimization," *SIAM Journal on Control and Optimization*, vol. 14, no. 2, pp. 216–235, 1976.

[148] A. Boccia, L. Grüne, and K. Worthmann, "Stability and feasibility of state constrained MPC without stabilizing terminal constraints," *Systems & Control Letters*, vol. 72, pp. 14–21, 2014.

[149] A. M. M. Leal, "autodiff, a modern, fast and expressive C++ library for automatic differentiation," `https://autodiff.github.io`, 2018.

[150] A. Bambade, F. Schramm, A. Taylor, and J. Carpentier, "QPLayer: efficient differentiation of convex quadratic optimization," June 2023, working paper or preprint. [Online]. Available: https://inria.hal.science/hal-04133055

[151] F. M. Smaldone, N. Scianca, V. Modugno, L. Lanari, and G. Oriolo, "Gait generation using intrinsically stable MPC in the presence of persistent disturbances," in *19th IEEE-RAS Int. Conf. on Humanoid Robots*, 2019, pp. 682–687.

[152] H. Chen, B. Wang, Z. Hong, C. Shen, P. M. Wensing, and W. Zhang, "Underactuated motion planning and control for jumping with wheeled-bipedal robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, p. 747–754, Apr. 2021.

[153] M. Bjelonic, R. Grandia, M. Geilinger, O. Harley, V. S. Medeiros, V. Pajovic, E. Jelavic, S. Coros, and M. Hutter, "Offline motion libraries and online mpc for advanced mobility skills," *The International Journal of Robotics Research*, vol. 41, no. 9–10, p. 903–924, Aug. 2022.