SAPIENZA
UNIVERSITÀ DI ROMA

Optimization-Based Methods for Real-Time Generation of Safe Motions in Mobile Robots

Sapienza Università di Roma

Dottorato di Ricerca in Automatica, Bioingegneria e Ricerca Operativa – XXXIV Ciclo

Candidate

Spyridon G. Tarantos

ID number 1840399

Thesis Advisor

Prof. Giuseppe Oriolo

Thesis defended on 25 January 2023
in front of a Board of Examiners composed by:

Prof. Paolo Valigi (chairman)

Prof.ssa Paola Cappanera

Prof. Danilo Pani

Optimization-Based Methods for Real-Time Generation of Safe Motions in Mobile Robots

Ph.D. thesis. Sapienza – University of Rome

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

Version: February 7, 2023

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

To my parents,
Gerasimos and Margarita

Abstract

Having robots operating in unstructured and dynamically changing environments is a challenging task that requires advanced motion generation approaches that are able to perform in real-time while maintaining the robot and environment safety.

The progress in the field of numerical optimization, as well as the development of tailored algorithms, made Nonlinear Model Predictive Control (NMPC) an appealing candidate for real-time motion generation. By considering the robot model as prediction model and through appropriate constraints on the robot states and control inputs, NMPC can enforce safety to the resulting motion in a straightforward way.

This thesis addresses the problem of real-time generation of safe motions for mobile robots and mobile manipulators. The different structure of the considered robots introduces different safety risks during the robot motion and so the motion generation problem for each robot is addressed in separate parts of this thesis.

In the first part, the problem of motion generation for mobile robots navigating in environments populated by static and/or moving obstacles is considered. For the generation of the desired motion, real-time NMPC is used. We argue that, in order to tackle the risk of collision with the environment, traditional distance-based approaches are incapable of maintaining the robot safety when the NMPC uses relatively short prediction horizons. Instead, we propose two NMPC approaches that employ two alternative collision avoidance constraints. The first proposed NMPC approach is applied to a scenario of safe robot navigation in a human crowd. The NMPC serves as a motion generation module in a safe motion generation framework, complete with a crowd prediction module. The considered collision avoidance constraint is built upon an appropriate Control Barrier Function (CBF). The second NMPC approach is applied to a scenario of robot navigation among moving obstacles, where the dynamics of the considered robot are significant. The proposed collision avoidance constraint is built upon the notion of *avoidable collision state*, which considers not only the robot-obstacle distance but also their velocity as well as the robot actuation capabilities. The simulation results indicate that both methods are effective and able to maintain the robot safety even in cases where their purely distance-based counterparts fail.

The second part of the thesis addresses the problem of safe motion generation for mobile manipulators, called to execute tasks that may require aggressive motions. Here, in addition to the risk of collision with its environment, the robot, consisting of multiple articulated bodies, is also susceptible to self-collisions. Moreover, fast motions can always result to loss of balance. To solve the problem, we propose a real-time NMPC scheme that uses the robot full dynamics, in order to enforce kinodynamic feasibility, while it also considers appropriate collision and self-collision avoidance constraints. To maintain the robot balance we enforce a constraint that restricts the feasible set of robot motions to those generating non-negative moments around the edges of the support polygon. This balance constraint, inherently nonlinear, is linearized using the NMPC solution of the previous iteration. In this way, we facilitate the solution of the NMPC in real-time, without compromising the robot safety.

Although the proposed NMPC is effective when applied to MM with low degrees of freedom, when the robot becomes more complex the use of its full dynamic model as a prediction model in an NMPC can lead to unacceptably large computational times that are not compatible with the real-time requirement. However, the use of a simplified model of the robot in an NMPC can compromise the robot safety. For this reason, we propose an optimization-based controller equipped with balance constraints as well as CBF-based collision avoidance constraints. The proposed controller can serve as an intermediate between a motion generation module that does not consider the robot full dynamics and the robot itself in order to ensure that the resulting motion will be at least safe. Simulation results indicate the effectiveness of the proposed method.

Acknowledgments

With this thesis, an important part of my academic life, the period of my Ph.D., comes to an end. With this opportunity, I would like to express my gratitude to those that supported me during these years.

First of all, I would like to thank my supervisor prof. Giuseppe Oriolo for giving me the opportunity to work with him and his team. I can only be grateful for his great support during this period.

I would also like to mention all my colleagues and friends from DIAG Robotics Lab for creating a supportive and fun working environment.

Special thanks to my girlfriend Maria for her incredible patience and support during this period.

I would also like to thank my sister Dimitra for being always my greatest supporter in all my endeavors.

Finally, this thesis is dedicated to my parents Gerasimos and Margarita, for being always by my side and supporting me blindly in any way possible. Words are not enough to express my gratitude.

Contents

Chapter 1

Introduction

Over the course of years, human beings invented tools and machines in an attempt to reduce their physical effort, increase productivity and improve their quality of life. Eventually, they managed to create robots able to replace them in performing tasks that would require excessive physical strength, could potentially expose them to danger, or simply were rather trivial and repetitive.

A family of robots, commonly used in industrial applications are the *robot manipulators* (see Fig. 1.1). They consist of a series of rigid bodies, referred to as links, connected with joints that can be either prismatic or revolute serving as means of articulation. The structure of a manipulator is characterized by the way in which the links are arranged. The consecutive links connected with the joints constitute a *kinematic chain*, which is *open* if no loops are formed by the links or *closed* in a different case. In the case of the open kinematic chain, each joint adds a degree of freedom (DOF) to the manipulator, while in the case of the closed kinematic chain, the number of DOF is reduced by the constraints imposed by the kinematic loops. The last link of the manipulator is typically connected to an *end-effector* appropriate for the considered application. Grippers, cameras, welding torches, spray guns and drills are only some of the tools that can be used as end-effectors. Finally, an important characteristic of a manipulator is its *workspace*, i.e., the area that the end-effector can reach, that depends on the manipulator structure and is limited.

The *mobile robots* constitute another big family of robotic systems. Unlike the robot manipulators, whose base is fixed on their environment, mobile robots consist of a mobile base equipped with a locomotion system that permits them to move in the environment, creating in such a way a practically unlimited workspace. Depending on the particular locomotion system, the mobile robots can be categorized into land-based, with representative examples the wheeled and legged robots, air-based

Figure 1.1. Examples of robot manipulators. Starting from the left, the KUKA LBR iiwa robot, the ABB YuMi - IRB 14000 robot and the ABB IRB 360 FlexPicker robot.

Figure 1.2. Examples of mobile robots. Starting from the left, the Robotnik SUMMIT-XL as representative of the land-based mobile robots, the AscTec Firefly for the air-based and the Girona 500 AUV for the water-based.

Figure 1.3. Examples of mobile manipulators. Starting from the left, the TIAGo mobile manipulator, the Spot quadruped equipped with a robotic arm, a Fletcher double-rotor helicopter equipped with an LWR arm and the Girona 500 AUV equipped with a manipulator.

and water-based (see Fig. 1.2).

A combination of these two types of robots results to a family of rather versatile robots, the *mobile manipulators* (MM), that combine the advantages of both subsystems. The MMs can be formed by mounting one or more robot manipulators on a mobile robot and combine the mobility of the mobile robot and the dexterity of the manipulators. In this way, the MM may be called to reach targets that at the beginning of the operation were outside of the manipulator workspace. Depending on the type of the mobile base, the MMs can operate on the land, in the air, or in the water (see Fig. 1.3).

The robots were originally employed in industrial applications in protected environments where the interaction with the humans was limited and monitored, while if humans had to operate in the vicinity of a robot they were trained to do so. Fig. 1.4 (left) illustrates robots working in an isolated space protected by a cage, minimizing the possibility of a human entering unintentionally in their workspace.

One of the greatest challenges in robotics was to let the robots leave the cages and start operating alongside humans not only in industrial but also in urban or even domestic environments. See in Fig. 1.4 (right) a delivery robot navigating through a group of people. Note that, unlike a robot in an industrial environment, robots in an urban environment have to deal with its unstructured nature. On top of that, humans in an urban environment are not typically trained to coexist with a robot, setting additional challenges to its operation.

Clearly, the problem of *motion generation* for such robots is challenging as the robot is not only required to execute the assigned task, but also to ensure that throughout its motion its safety as well as the safety of its surrounding environment is preserved. For this to be achieved, the motion generation module is required to be equipped with an accurate model of the considered robot, to receive the latest information about the robot surroundings and to be able to react to any change in

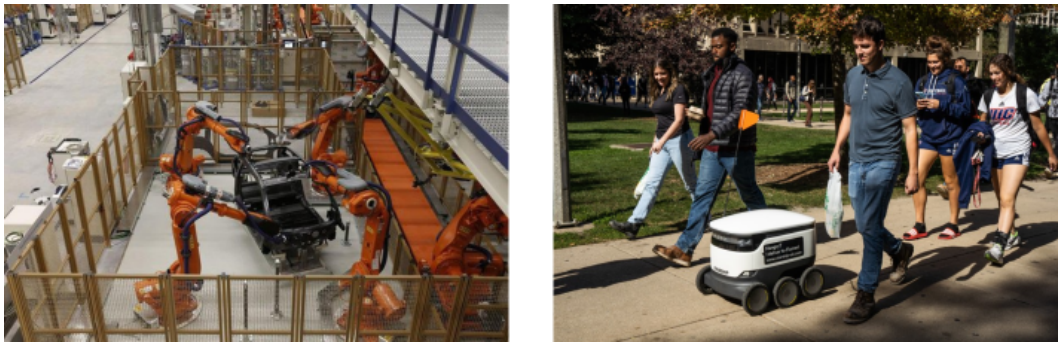

Figure 1.4. Robots employed in industrial (left) and urban (right) environments. One can notice that in the industrial environment, the robots are isolated within a cage limiting in such a way the access to humans, while in the urban environment, the robot operates in close proximity to humans.

the environment that might occur by adapting appropriately the robot motion. As a result, real-time performance is also a requirement.

One of the most successful and advanced approaches for real-time motion generation is the Nonlinear Model Predictive Control (NMPC). The motion planning problem is formulated as an Optimal Control Problem (OCP) over a finite time horizon. At each control cycle, the latest measurements for the robot and environment state are considered and the OCP is solved. The resulting control action is applied to the robotic system for a short time interval, until new measurements for the robot and the environment are available, so that a new OCP is solved to obtain the new control action. This recomputation constitutes the feedback mechanism of NMPC. The recent algorithmic and hardware advancements permit a significant reduction of the time interval between two consecutive measurements, enabling NMPC to perform in real-time.

This thesis addresses the problem of generating safe motions for wheeled mobile robots and wheeled mobile manipulators, using real-time NMPC. The thesis is divided into two parts in order to address separately the motion generation problem for each of the two types of robots.

The first part addresses the problem of real-time motion generation for mobile robots navigating in environments populated by static and/or moving obstacles. Of particular importance is the robot safety in terms of collision with the environment obstacles. Our motion generation approach is based on a real-time NMPC algorithm where the robot safety is enforced via appropriate collision avoidance constraints. We argue that the use of purely distance-based collision avoidance constraints can jeopardize the robot safety when the considered prediction horizons are short. Instead, we propose the use of collision avoidance constraints that, in addition to distance information, consider the robot-obstacle relative velocity as well as the robot actuation capabilities. Simulation results indicate the effectiveness of the proposed methods and their superior performance compared to their purely distance-based counterparts.

The second part concerns the problem of real-time motion generation for mobile manipulators called to execute end-effector tasks that might require aggressive

motions in environments populated by obstacles. Here central role plays the robot safety, which in the case of a MM is defined in terms of collision with the environment obstacles, self-collision and loss of balance. The proposed motion generation approach is based on a real-time NMPC algorithm that considers the robot full dynamics, while the robot safety is enforced via appropriate collision and balance constraints. Although this method effectively generates safe motions for MMs of low DOF, as the simulations indicate, when the robot becomes more complex, the computational time required by a motion generation module that considers the robot full dynamics can be unacceptably large for real-time applications. In cases like this, the literature suggests the use of simplified models that enable real-time performance. However, the use of such models cannot guarantee the robot safety. To tackle this problem, we also propose an optimization-based controller that can be interposed between the motion generation module and the robot, in order to ensure that the resulting motion is at least safe. Simulation results indicate the effectiveness of the proposed controller even in cases where the motion generation module considers only the robot kinematics.

The thesis is organized as follows

- Chapter 2 offers some preliminaries on real-time NMPC and presents the algorithms that are going to be used in the following chapters as motion generation approaches.

Part I: Motion generation for mobile robots

- Chapter 3 formulates the problem of mobile robot navigation in environments populated by obstacles. After obtaining the kinematic and dynamic model of a differential-drive mobile robot, the proposed NMPC for the real-time generation of the desired motion is presented. The chapter concludes with a discussion on appropriate collision avoidance constraints within a real-time NMPC, paving the road for the following chapters.

- Chapter 4 considers the problem of safe robot navigation in a human crowd. The proposed NMPC algorithm serves as a motion generation module in a sensor-based scheme for safe robot navigation that is complete with a crowd prediction module. To ensure safety, the NMPC considers collision avoidance constraints built upon appropriate discrete-time CBFs. The method presented in this chapter was originally presented in our paper [1].

- Chapter 5 presents a dynamics-aware NMPC method for robot navigation among moving obstacles. Here we propose a collision avoidance constraint that essentially requires the robot to be always at a state from which it can avoid collision with a certain obstacle. The method presented in this chapter was originally presented in our paper [2].

Part II: Motion generation for mobile manipulators

- Chapter 6 formulates the problem of real-time motion generation for a mobile manipulator called to execute tasks that require aggressive motions. After obtaining the kinematic and dynamic model of a general wheeled mobile

manipulator, we propose an NMPC algorithm for real-time motion generation that considers the robot full dynamics and appropriate constraints to enforce safety. To enable real-time performance we proposed a linearization of the considered balance constraint using the solution of the previous NMPC iteration. The method presented in this chapter was originally presented in our paper [3].

- Chapter 7 considers the problem of motion generation for mobile manipulators using a simplified model of the robot. In this chapter, we propose an optimization-based controller that considers the robot full dynamics as well as collision avoidance constraints built upon appropriate CBFs and balance constraints. The proposed controller can be interposed between the motion generation module and the considered robot in order to ensure that the resulting motion will be at least safe.

- Chapter 8 offers some concluding remarks and directions for future work.

Chapter 2

Real-time nonlinear model predictive control

Model Predictive Control (MPC) is a control technique where the control action is obtained via the on-line solution of a finite horizon, open-loop Optimal Control Problem (OCP), which is subject to a prediction model of the system and constraints on the system state and control inputs [4]. At a given time instant, the controller, based on measurements received from the system, solves the OCP and obtains a control action that respects the imposed constraints and minimizes certain performance criteria (imposed as a cost function in the OCP). The computed control action is applied to the system only for a small time interval, typically until a new measurement is available. Then, based on the new measurement, a new, updated, control action is computed and applied to the system. This recomputation of the control action, based on the latest measurements, constitutes the feedback mechanism of the MPC.

This control approach became rather attractive over the last years and has been used in numerous applications [5]. Its conceptual simplicity in combination with its ability to control multivariable systems and to incorporate hard constraints in a straightforward way justify its popularity.

Linear MPC (LMPC) is a category of MPC where the prediction model and the imposed constraints are linear while the cost function is quadratic [6]. LMPC is a rather mature technology while the developed algorithms for the solution of the resulting quadratic program (QP) allow low computational times [7, 8, 9, 10, 11]. On the downside, the majority of the systems that we are called to control, especially in the field of robotics, are inherently nonlinear. In principle, one can apply LMPC to a nonlinear system by linearizing it around a reference trajectory (possibly generated off-line) [12]. However, a nonlinear model can describe more accurately the behavior of a nonlinear system. Moreover, tight performance requirements that bring the system to its limits (e.g., high-speed robot motion at the vicinity of obstacles or close to balance loss) cannot be efficiently handled by a linear model over large time intervals. This motivates the development of Nonlinear MPC (NMPC) which permits the use of a nonlinear model as a prediction model as well as nonlinear constraints and cost functions. Nevertheless, due to its higher complexity compared to its linear counterpart, NMPC was originally deployed in slower processes, where the time

between consecutive requests for a new control action was large enough to permit the computationally intensive algorithms to converge. Recently, the use of powerful processing platforms in combination to the development of appropriate algorithms, reduced the computational time of NMPC making it a competitive candidate for real-time applications [13, 14, 15, 16, 17].

This chapter serves as a preliminary to the following ones where real-time NMPC is used as a motion generation approach. The NMPC formulation is given in Sect. 2.1 describing an OCP appropriate for the considered applications of this thesis. Sect. 2.2 is dedicated to the numerical solution of the considered OCP and of the NMPC in general. Sect. 2.3 is dedicated to the real-time implementation of an NMPC algorithm focusing in particular on the real-time iteration scheme. Finally, some concluding remarks are offered in Sect. 2.4.

2.1 Model predictive control formulation

In this work, we consider robotic systems whose equations of motion can be described by ordinary differential equations (ODE)

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{\phi}\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) \tag{2.1}$$

with initial value for $t = t_0$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0.$$

The vectors $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and $\boldsymbol{u} \in \mathbb{R}^{n_u}$ correspond to the system states and control inputs respectively. In principle, these vectors (or some elements of them) are constrained with lower and upper bounds enforced by the system hardware limitations (e.g., actuator limits) and/or the specific application (e.g., maximum permitted driving velocity for a mobile robot)

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{\max}$$

where $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$ and $\boldsymbol{u}_{\min}$, $\boldsymbol{u}_{\max}$ are the lower and upper bounds of the system state and control inputs.

Note that depending on the application, the system may be also subject to additional constraints on the state, the inputs or both (e.g., collision avoidance constraints, balance constraints, occlusion constraints, etc.). These constraints can be expressed as path constraints

$$\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \leq \boldsymbol{0}.$$

At each time instant, given a state estimation of the system, NMPC solves an OCP defined over a finite horizon and subject to a prediction model of the system and constraints on the system state and control inputs. The resulting control action is applied to the system for a small time interval until a new measurement is available. Given the new measurement, the NMPC solves the new OCP in order to obtain the control action.

Algorithm 1: NMPC algorithm

Input: the initial state of the system $\boldsymbol{x}_0$ at t_0

$t_k \leftarrow t_0$;
$\bar{\boldsymbol{x}}_k \leftarrow \boldsymbol{x}_0$;
while *true* **do**
  receive the estimated state of the system at time t_k, $\bar{\boldsymbol{x}}_k$;
  solve the OCP (2.2) over the time interval $[t_k, t_k + T]$ and obtain the optimal control action $\boldsymbol{u}_k^*(\cdot)$;
  isolate the control action that corresponds to the time interval $[t_k, t_k + \delta]$, $\boldsymbol{u}_k^{\mathrm{NMPC}}(\cdot)$, and apply it to the system;
  $t_k \leftarrow t_k + \delta$;
end

Let us consider the generic time instant t_k and denote by $\bar{\boldsymbol{x}}_k$ the estimated state of the system at the considered time instant. Let us also consider the finite horizon OCP of interest defined over a time interval $[t_k, t_k + T]$ as

$$\min_{\boldsymbol{u}(\cdot)} \int_{t_k}^{t_k+T} V_r(\boldsymbol{x}(t), \boldsymbol{u}(t))dt + V_t(\boldsymbol{x}(t_k + T)) \tag{2.2a}$$

subject to:

$$\boldsymbol{x}(t_k) - \bar{\boldsymbol{x}}_k = \boldsymbol{0} \tag{2.2b}$$

$$\dot{\boldsymbol{x}}(t) - \boldsymbol{\phi}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \boldsymbol{0}, \qquad t \in [t_k, t_k + T] \tag{2.2c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{\max}, \qquad t \in [t_k, t_k + T] \tag{2.2d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{\max}, \qquad t \in [t_k, t_k + T] \tag{2.2e}$$

$$\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \leq \boldsymbol{0}, \qquad t \in [t_k, t_k + T] \tag{2.2f}$$

$$\boldsymbol{r}(\boldsymbol{x}(t_k + T)) \leq \boldsymbol{0} \tag{2.2g}$$

where T the prediction horizon, $V_r(\boldsymbol{x}(t), \boldsymbol{u}(t))$ and $V_t(\boldsymbol{x}(t_0 + T))$ the running and terminal cost functions respectively, while (2.2g) represents state constraints at the final time instant.

Solution of the OCP (2.2) gives an optimal control action $\boldsymbol{u}_k^*(\cdot) : [t_k, t_k + T] \rightarrow \mathcal{U}$ where $\mathcal{U}$ is the feasible set of control inputs as defined by the constraints considered in the OCP. At the time instant t_k and given the state estimate $\bar{\boldsymbol{x}}_k$ the NMPC solves the OCP (2.2) and applies to the system a control action $\boldsymbol{u}_k^{\mathrm{NMPC}}(\cdot)$ for a time interval $[t_k, t_k + \delta]$ such that

$$\boldsymbol{u}_k^{\mathrm{NMPC}}(t) = \boldsymbol{u}_k^*(t), \quad t \in [t_k, t_k + \delta] \tag{2.3}$$

with δ being the considered sampling time. When a new state estimation becomes available, the OCP is solved again and a new control action is applied to the system. The steps of an NMPC algorithm are presented at Algorithm 1.

2.2 Numerical solution of the OCP

It is clear that the solution of the OCP is at the heart of the NMPC algorithm. The choice of the particular method for its solution will strongly affect the performance

and the reliability of the whole NMPC.

First, we briefly present different solution approaches for the OCP with particular focus on the direct methods and specifically on the direct multiple shooting method, which is a popular approach for transcribing the OCP into a Non-Linear Program (NLP), especially for real-time applications. Then we discuss the solution of the resulting NLP using a Sequential Quadratic Programming (SQP) algorithm.

2.2.1 Numerical solution approaches

There are three main classes of methods used for the numerical solution of the OCP.

The first class consists of the *Dynamic Programming* (DP) and the *Hamilton-Jacobi-Bellman* (HJB) [18] approaches. Both methods are based on the computation of the level sets of the OCP value function and require the solution of a partial differential equation. These methods suffer from the, so called, *course 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 require the optimality conditions of the considered problem and are based on the exploitation of Pontryagin's minimum principle [19]. These methods end up to the numerical solution of a multi-point boundary value problem [20]. Although they lead to highly accurate numerical solutions, indirect methods are not normally applied to on-line implementations, like in an NMPC [4].

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 [21]. The direct methods are in principle more popular for the on-line solution of an OCP, so in the rest of this section, we will focus on them.

The direct methods are characterized by the way in which they transcribe the OCP into an NLP. Their differences consist of the way in which the state and inputs are discretized, as well as of the parameters that will be chosen as decision variables in the resulting NLP. The three approaches are the *direct collocation*, the *direct single shooting* and the *direct multiple shooting*.

The direct collocation method aims at the discretization of both the states and the inputs. The resulting NLP is large and sparse and can be solved by an SQP or an Interior-Point (IP) algorithm [22].

The direct single shooting discretizes only the control inputs, which are then used as decision variables in the resulting NLP. As regards the state variables, given the initial state of the system they derive implicitly by integrating the system dynamics. This method leads to an NLP with a small number of decision variables, compared to the direct collocation method. However, it is sensitive to changes in the initial value of the state, which has a large influence on the state when the system is highly nonlinear or unstable [23].

Finally, the direct multiple shooting method [24], as the name suggests, practically divides the time interval $[t_k, t_k + T]$ into smaller subintervals and applies the shooting method to each one of them. This approach increases the decision variables in the resulting NLP, as now the states that correspond to the shooting nodes have to also be considered as decision variables. As a result, the NLP is large (not larger

however than the one of the collocation method) and sparse. Moreover, appropriate conditions have to be added for each shooting node to ensure the continuity of the resulting solution. Compared to the single shooting, the multiple shooting method can be parallelized, can take advantage of the state variables in order to initialize the optimization solvers and generally reports faster contraction rates of Newton-type iterations, especially for highly nonlinear and unstable systems [21]. Finally, is often preferable for on-line implementations.

In the following, we will present in more detail the direct multiple shooting method.

Direct multiple shooting method

For the discretization of the OCP we start from the partition of the prediction horizon $[t_k, t_k + T]$ using a time grid that divides the considered time interval into N subintervals

$$t_k < t_{k+1} < ... < t_{k+N} = t_k + T. \tag{2.4}$$

Although it is not necessary, we choose the resulting subintervals $[t_{k+i}, t_{k+i+1}]$ for $i = 0, ..., N-1$ to be equidistant and equal to the sampling time, i.e., $t_{k+i+1} - t_{k+i} = \delta$. Over the chosen grid, we discretize the control trajectory $\boldsymbol{u}(\cdot)$ using a piecewise-constant vector function

$$\boldsymbol{u}(t) = \boldsymbol{u}_{k|i}, \quad \forall t \in [t_{k+i}, t_{k+i+1}]$$

with $\boldsymbol{u}_{k|i} \in \mathbb{R}^{n_u}$. Note that alternative control parameterizations can be also considered. The interested reader is referred to [23] for a list of the most common ones.

Regarding the state trajectory, let us denote by $\boldsymbol{x}_{k|i}$ the predicted state of the system at time instant t_{k+i}, i.e., $\boldsymbol{x}_{k|i} = \boldsymbol{x}(t_{k+i})$. In the multiple shooting setup, the state trajectory is considered independently at each time subinterval $[t_k, t_{k+1}]$ and the vectors $\boldsymbol{x}_{k|i}$, for $i = 0, ..., N$ at the beginning of each subinterval are used as decision variables. To ensure continuity for the resulting trajectory, we consider that the state of the system over a subinterval $[t_{k+i}, t_{k+i+1}]$ is described by the solution of the Initial Value Problem (IVP)

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{\phi}(\boldsymbol{x}(t), \boldsymbol{u}_{k|i}), \quad \boldsymbol{x}(t_{k+i}) = \boldsymbol{x}_{k|i}, \quad t \in [t_{k+i}, t_{k+i+1}]. \tag{2.5}$$

If we denote by $\boldsymbol{\phi}_{\text{d-t}}(\cdot, \cdot)$ the discrete time dynamics of the system (2.1) obtained by a numerical integration method under the assumption of piecewise-constant control inputs and by $\tilde{\boldsymbol{x}}(t_{k+i+1}; \boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i})$ an approximation of the solution of the IVP (2.5) for $t = t_{k+i+1}$ such that

$$\tilde{\boldsymbol{x}}(t_{k+i+1}; \boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) \tag{2.6}$$

then to ensure continuity, the following equality constraint has to be considered

$$\boldsymbol{x}_{k|i+1} - \tilde{\boldsymbol{x}}(t_{k+i+1}; \boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0, ..., N-1$$

or equivalently

$$\boldsymbol{x}_{k|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0, ..., N-1. \tag{2.7}$$

For the discretization of the running cost function, one can evaluate it at the corresponding nodes of the grid (2.4) or at a possibly finer one. The same procedure can be followed for the considered state, inputs and path constraints.

So the infinite-dimensional OCP can be transformed into the finite-dimensional NLP of the form

$$\min_{\substack{\boldsymbol{x}_{k|0},\dots,\boldsymbol{x}_{k|N},\\ \boldsymbol{u}_{k|0},\dots,\boldsymbol{u}_{k|N-1}}} \sum_{i=0}^{N-1} V_{k|i}(\boldsymbol{x}_{k|i},\boldsymbol{u}_{k|i}) + V_{k|N}(\boldsymbol{x}_{k|N}) \tag{2.8a}$$

subject to:

$$\boldsymbol{x}_{k|0} - \bar{\boldsymbol{x}}_k = \boldsymbol{0} \tag{2.8b}$$

$$\boldsymbol{x}_{k|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i},\boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0,\dots,N-1 \tag{2.8c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_{k|i} \leq \boldsymbol{x}_{\max}, \quad i = 0,\dots,N \tag{2.8d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_{k|i} \leq \boldsymbol{u}_{\max}, \quad i = 0,\dots,N-1 \tag{2.8e}$$

$$\boldsymbol{h}(\boldsymbol{x}_{k|i},\boldsymbol{u}_{k|i}) \leq \boldsymbol{0}, \quad i = 0,\dots,N-1 \tag{2.8f}$$

$$\boldsymbol{r}(\boldsymbol{x}_{k|N}) \leq \boldsymbol{0} \tag{2.8g}$$

where $\bar{\boldsymbol{x}}_k$ is the estimated state of the system at time t_k, (2.8b) the initial value constraint, $V_{k|i}(\boldsymbol{x}_{k|i},\boldsymbol{u}_{k|i})$ the running cost evaluated at t_{k+i} and $V_{k|N}(\boldsymbol{x}_N)$ the terminal cost evaluated at t_{k+N}.

2.2.2 Nonlinear optimization

Preliminaries

Let us express the NLP of interest (2.8) in the form

$$\min_{\boldsymbol{w}} F(\boldsymbol{w}) \tag{2.9a}$$

subject to:

$$\boldsymbol{G}(\boldsymbol{w},\bar{\boldsymbol{x}}_k) = \boldsymbol{0} \tag{2.9b}$$

$$\boldsymbol{H}(\boldsymbol{w}) \leq \boldsymbol{0} \tag{2.9c}$$

collecting the decision variables in the vector

$$\boldsymbol{w} = (\boldsymbol{x}_{k|0},\boldsymbol{u}_{k|0},\dots,\boldsymbol{x}_{k|N-1},\boldsymbol{u}_{k|N-1},\boldsymbol{x}_{k|N}) \in \mathbb{R}^{n_w},$$

and expressing the n_e equality constraints (2.8b) and (2.8c) as $\boldsymbol{G}(\boldsymbol{w},\bar{\boldsymbol{x}}_k) = \boldsymbol{0}$, the n_i inequality constraints (2.8d - 2.8g) as $\boldsymbol{H}(\boldsymbol{w}) \leq \boldsymbol{0}$ and the cost functions as $F(\boldsymbol{w})$. Note that in $\boldsymbol{G}(\boldsymbol{w},\bar{\boldsymbol{x}}_k)$, the initial state $\bar{\boldsymbol{x}}_k$ enters linearly due to the initial value constraint (2.8b).

For the rest of this analysis, we assume that all the functions are at least twice differentiable.

Let us denote by $\mathcal{S}$ the set of all $\boldsymbol{w}$ that satisfy the constraints (2.9b) and (2.9c) defined as

$$\mathcal{S} := \{\boldsymbol{w} \in \mathbb{R}^{n_w} | \boldsymbol{G}(\boldsymbol{w},\bar{\boldsymbol{x}}_k) = \boldsymbol{0} \text{ and } \boldsymbol{H}(\boldsymbol{w}) \leq \boldsymbol{0}\}.$$

We will refer to $\mathcal{S}$ as *feasible set* and to each point in it, $\boldsymbol{w} \in \mathcal{S}$ as *feasible point.*

Consider the i-th element of $\boldsymbol{H}(\boldsymbol{w})$ that corresponds to the i-th inequality constraint $h_i(\boldsymbol{w})$. This constraint is called *active* if $h_i(\boldsymbol{w}) = 0$, while *inactive* if $h_i(\boldsymbol{w}) < 0$. We collect all the indices that correspond to active inequality constraint in $\mathcal{A}(\boldsymbol{w})$ defined as

$$\mathcal{A}(\boldsymbol{w}) := \{i \in \{1, ..., n_i\} | h_i(\boldsymbol{w}) = 0\}.$$

We say that at a point $\boldsymbol{w} \in \mathcal{S}$ the Linear Independence Constraints Qualification (LICQ) holds iff the constraint gradients $\nabla g_j(\boldsymbol{w})$ for $j \in \{1, ..., n_e\}$, where g_j the j-th element of $\boldsymbol{G}$, and $\nabla h_i(\boldsymbol{w})$ for $i \in \mathcal{A}(\boldsymbol{w})$ are linearly independent.

The *Lagrange function* of the optimization problem (2.9) is

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = F(\boldsymbol{w}) + \boldsymbol{\lambda}^T \boldsymbol{G}(\boldsymbol{w}, \bar{\boldsymbol{x}}_k) + \boldsymbol{\mu}^T \boldsymbol{H}(\boldsymbol{w}) \tag{2.10}$$

with $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_{n_e})$ and $\boldsymbol{\mu} = (\mu_1, ..., \mu_{n_i})$ the vectors of the Lagrange multipliers.

First-order necessary conditions or *Karush - Kuhn - Tacker* (KKT) conditions suggest that if $\boldsymbol{w}^*$ is a local minimizer of the optimization problem (2.9) and LICQ holds at $\boldsymbol{w}^*$ then there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \boldsymbol{0} \tag{2.11a}$$
$$\boldsymbol{G}(\boldsymbol{w}^*, \bar{\boldsymbol{x}}_k) = \boldsymbol{0} \tag{2.11b}$$
$$\boldsymbol{H}(\boldsymbol{w}^*) \leq \boldsymbol{0} \tag{2.11c}$$
$$\mu_i^* \geq 0, \quad i = 1, ..., n_i \tag{2.11d}$$
$$\mu_i^* h_i(\boldsymbol{w}^*) = 0, \quad i = 1, ..., n_i \tag{2.11e}$$

The point $(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ that satisfies the KKT conditions is called KKT point.

In principle, for a point at which LICQ holds, the KKT conditions are only necessary for optimality and it is required to argue about the second order derivatives for local optimality. However, in the special case of convex optimization problems, the KKT conditions are also sufficient for global optimality.

Most of the optimization solvers aim at finding an approximate solution to the KKT conditions (2.11). Representative examples are the Interior-Point (IP) [25] methods and the Sequential Quadratic Programming (SQP) [22]. The two methods mainly differ in the way in which (2.11d) and (2.11e) are treated.

In this work, we are going to solve the NMPC using the real-time iteration method that is based on the SQP algorithm. As a result in the rest we will focus on this.

Sequential quadratic programming

SQP is an iterative method for the solution of the NLP. At each iteration, SQP solves a quadratic approximation of the NLP until a convergence criterion is satisfied. Specifically, at the κ-th iteration, an inequality constrained Quadratic Program (QP) is obtained from the quadratic approximation of the NLP around the iterate $\boldsymbol{w}_\kappa$. Solution of the resulting QP gives a search direction $\Delta \boldsymbol{w}_\kappa$ based on which the next iterate is computed from

$$\boldsymbol{w}_{\kappa+1} = \boldsymbol{w}_\kappa + \alpha_\kappa \Delta \boldsymbol{w}_\kappa \tag{2.12}$$

where $\alpha_\kappa \in [0, 1]$ a step size computed by a globalization method [22].

The inequality constrained QP obtained by the quadratic approximation of the NLP (2.9) at the iterate $\boldsymbol{w}_\kappa$ is

$$\min_{\Delta \boldsymbol{w}_\kappa} \nabla F(\boldsymbol{w}_\kappa)^T \Delta \boldsymbol{w}_\kappa + \frac{1}{2} \Delta \boldsymbol{w}_\kappa{}^T \bar{\boldsymbol{B}}_\kappa \Delta \boldsymbol{w}_\kappa \tag{2.13a}$$

subject to:

$$\boldsymbol{G}(\boldsymbol{w}_\kappa, \bar{\boldsymbol{x}}_k) + \nabla \boldsymbol{G}(\boldsymbol{w}_\kappa)^T \Delta \boldsymbol{w}_\kappa = \boldsymbol{0} \tag{2.13b}$$

$$\boldsymbol{H}(\boldsymbol{w}_\kappa) + \nabla \boldsymbol{H}(\boldsymbol{w}_\kappa)^T \Delta \boldsymbol{w}_\kappa \leq \boldsymbol{0} \tag{2.13c}$$

where $\nabla \boldsymbol{G}(\boldsymbol{w}_\kappa)$ and $\nabla \boldsymbol{H}(\boldsymbol{w}_\kappa)$ are the Jacobians of the equality and inequality constraints, respectively, and $\nabla F(\boldsymbol{w}_\kappa)$ the gradient of the objective function, all evaluated at $\boldsymbol{w}_\kappa$. Note that since the initial value $\bar{\boldsymbol{x}}_k$ enters linearly in $\boldsymbol{G}(\boldsymbol{w}_\kappa, \bar{\boldsymbol{x}}_k)$, the Jacobian $\nabla \boldsymbol{G}(\boldsymbol{w}_\kappa)$ is independent of $\bar{\boldsymbol{x}}_k$. Such property will be particularly useful for the real-time implementation of the SQP method.

The matrix $\bar{\boldsymbol{B}}_\kappa$ in principle corresponds to the exact Hessian of the Lagrangian, i.e., $\nabla^2_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_\kappa, \boldsymbol{\lambda}, \boldsymbol{\mu})$. However, computation of the exact Hessian requires a considerable amount of time and so an approximation of it can be considered. In this work, we will consider the *generalized Gauss-Newton method* [26] which is appropriate for cost functions of least squares form. For a cost function of the form $F(\boldsymbol{w}) = 1/2 \left| c(\boldsymbol{w}) \right|^2$ the Gauss-Newton Hessian can be written as $\bar{\boldsymbol{B}} = \nabla c(\boldsymbol{w}) \nabla c(\boldsymbol{w})^T$, which is a positive semi-definite matrix. We can notice that the considered Hessian is Lagrange multiplier-free as it only depends on the primal variables $\boldsymbol{w}$.

The SQP algorithm is given in Algorithm 2. Note that the computationally intensive parts of this algorithm are: (i) the computation of the sensitivities in step 3 and (ii) the solution of the QP in step 4.

QP solution and condensing As it is shown in Algorithm 2, at each SQP iteration a QP is constructed from the quadratic approximation of the NLP (2.9) and solved. Since the NLP (2.9) results from the use of the multiple shooting method, the associated QP considers decision variables that correspond to both the states and the control inputs of the system and as a result, the problem has high dimensions and a sparse block structure. This structure can be exploited by using an appropriate QP solver (e.g., `FORCES` [7] or `qpDUNES` [9]). An alternative approach suggests the reduction of the decision variables by eliminating those that correspond to the system state, using the linearized system dynamics from (2.13b). This approach, referred as *condensing* [24], results to a smaller and dense QP that can be efficiently solved by appropriate solvers that use dense linear algebra (e.g., `qpOASES` [8]). Clearly, the use of condensing requires an additional step after the solution of the resulting QP (step 4 of Algorithm 2) in order to obtain the elements of $\boldsymbol{w}_\kappa$ that correspond to the state of the system using again the linearized dynamics. It is reported that the condensing method is appropriate for relatively short prediction horizons, while as the size of the horizon increases the use of structure exploiting solvers is preferable. For more details, the interested reader is referred to [27].

Algorithm 2: SQP algorithm using the generalized Gauss-Newton method

Input: the estimated state of the system at time t_k, $\bar{\boldsymbol{x}}_k$, and an initial guess $\tilde{\boldsymbol{w}}_k$

$\boldsymbol{w}_\kappa \leftarrow \tilde{\boldsymbol{w}}_k$;
while *convergence conditions not satisfied* **do**
  compute the vectors $\boldsymbol{G}(\boldsymbol{w}_\kappa, \bar{\boldsymbol{x}}_k)$, $\boldsymbol{H}(\boldsymbol{w}_\kappa)$ and the sensitivities $\nabla F(\boldsymbol{w}_\kappa)$, $\bar{\boldsymbol{B}}_\kappa$, $\nabla \boldsymbol{G}(\boldsymbol{w}_\kappa)$ and $\nabla \boldsymbol{H}(\boldsymbol{w}_\kappa)$;
  construct and solve the QP (2.13) to obtain the search direction $\Delta \boldsymbol{w}_\kappa$;
  compute step size α_κ;
  update the primal variables: $\boldsymbol{w}_\kappa \leftarrow \boldsymbol{w}_\kappa + \alpha_\kappa \Delta \boldsymbol{w}_\kappa$;
end
return $\boldsymbol{w}_\kappa$

2.3 Real-time implementation

At the beginning of this chapter, we implicitly assumed that at each time instant t_k a state estimate of the system is available and the control action from the NMPC is applied to the system. This would require an infinitesimally small computational amount of time that is obviously not realistic. On the contrary, the numerical solution of an OCP requires a significant amount of time, even comparable to the sampling time δ, while in the meantime the system evolves. Thus, by the time that the new control action is available, the system is at a different state than the estimated one. One can attack this problem by starting to solve the OCP before t_k using a prediction of the state that the system will have at t_k. Nevertheless, even if the solution will be available at t_k it will be based on outdated information, ignoring any disturbances that might affect the evolution of the system in the meantime.

In both cases, the so called *real-time dilemma* [12] is evident. It consists in deciding either to solve the NLP until convergence and apply an exact, however, based on outdated information about the system and its environment, solution or to apply an approximate solution but based on the most recent information available.

Fortunately, the recent advancement in the field permits not only the reduction of the computational time but also to maintain the approximation errors at acceptable levels.

Some ideas that can enable an NMPC algorithm to perform in real-time can be found in [28, 21]. Among them are:

- **offline preparation and code generation**: exploits the fact that the NLPs solved during each NMPC cycle are similar permitting the generation of an optimized code with vectors and matrices of fixed dimensions;
- **state prediction for delay compensation**: if one knows in advance an upper bound of the computational time needed for the solution of the NMPC, then it is advisable instead of using the current state estimate, to predict the state that the system will have at the time in which the solution is expected to be available.
- **separation into preparation and feedback phase**: here one can exploit

Algorithm 3: Real-time iteration algorithm

Input: solution of the previous NLP, $\boldsymbol{w}_{k-1}$

Preparation phase

1 prepare an initial guess $\tilde{\boldsymbol{w}}_k$ from the previous solution $\boldsymbol{w}_{k-1}$;

2 compute the vector $\boldsymbol{H}(\boldsymbol{w}_\kappa)$ and the sensitivities $\nabla F(\boldsymbol{w}_\kappa)$, $\bar{\boldsymbol{B}}_\kappa$, $\nabla \boldsymbol{G}(\boldsymbol{w}_\kappa)$ and $\nabla \boldsymbol{H}(\boldsymbol{w}_\kappa)$;

Input: state estimation $\bar{\boldsymbol{x}}_k$

Feedback phase

3 given the elements of step 2 construct and solve the QP (2.13) to obtain the search direction $\Delta \boldsymbol{w}_k$;

4 update the primal variables: $\boldsymbol{w}_k \leftarrow \tilde{\boldsymbol{w}}_k + \Delta \boldsymbol{w}_k$;

5 **return** $\boldsymbol{w}_k$

the fact that many of the expensive computations for the solution of the NLP can be performed without the knowledge of the estimated state $\bar{\boldsymbol{x}}_k$ and they can be performed in advance during a preparation phase. The rest of the computations that can be only performed as soon as the estimated state is available constitute the feedback phase.

- **use of warm-start**: exploiting the fact that in NMPC neighboring problems are consecutively solved, one can use the solution of the previous problem as an initial guess for the current one.

- **iteration along with the evolution of the problem**: based again on the idea that the NMPC solves similar neighboring problems, one can avoid solving each problem to convergence but instead settle with an approximate solution and move to the next optimization problem using the latest information available.

In the literature there exist several methods that use some of the ideas presented above like the *Newton-type controller* [29], the *continuation/GMRES method* [30] and the *advanced step controller* [31]. For a more detailed review of these methods, the reader is referred to [28, 12]. However, in this work, we will only focus on the Real-Time Iteration (RTI) method [32, 33] that will be presented in more detail in the following.

2.3.1 Real-time iteration scheme

As we already mentioned, within NMPC, iterating an SQP algorithm to convergence will require a significant (possibly prohibitive) amount of time, leading eventually to an outdated control action.

Providing a "good" initial guess, $\tilde{\boldsymbol{w}}_k$, to the SQP (see Algorithm 2) can be beneficial as [12]: (i) it minimizes the possibility to output an infeasible solution and (ii) it permits to take full Newton-steps, i.e., $\alpha_\kappa = 1$, increasing the convergence rate. However, providing a good initial guess to the SQP is not trivial. Luckily, in the context of NMPC this choice emanates naturally from the structure of the NMPC

itself. Specifically, NMPC requires the solution of consecutive similar OCPs (and consequently NLPs). As a result, their solution is also similar. For two consecutive NLPs, say the NLP that corresponds to time t_{k-1} and the NLP that corresponds to t_k, the solution of the one solved at t_{k-1} can be used as an initial guess for the solution of the one to be solved at t_k.

The Real-Time Iteration (RTI) scheme, originally presented in [32], exploits the fact that NMPC solves consecutive neighboring OCPs and instead of the solution of SQP until convergence, it suggests the execution of only one full Newton-step per NLP using as initial guess the solution of the previous one and always with the latest information available about the system. In order to further reduce the feedback time, the RTI scheme further exploits the structure of (2.13). Particularly it exploits the fact that $\boldsymbol{H}(\boldsymbol{w}_k)$, $\nabla F(\boldsymbol{w}_k)$, $\bar{\boldsymbol{B}}_k$, $\nabla \boldsymbol{G}(\boldsymbol{w}_k)$ and $\nabla \boldsymbol{H}(\boldsymbol{w}_k)$ whose computation is the most expensive step of the SQP iteration (see Algorithm 2) do not require the estimation of the system state $\bar{\boldsymbol{x}}_k$ for their computations. As a result, these computations can precede the estimation of the system state permitting the use of the most recent state information. These computations, that practically prepare the QP to be solved at the time that the estimated state will be available, constitute the *preparation phase*. After the state estimation is received the QP (2.13) can be solved in order to obtain the new Newton-step and consequently the solution of the NLP. This phase is the *feedback phase*. The Algorithm 3 presents an overview of the RTI scheme.

2.3.2 Efficient software

Apart from tailored algorithms, like the RTI, achieving real-time performance for the solution of the NMPC requires appropriate software. A list of embedded optimization software packages can be found in [34]. However, a review of those is out of the scope of this work. Instead, here we will briefly present the two packages that will be used in the chapters to follow, namely, the `ACADO Toolkit` and its successor `acados`.

The `ACADO Toolkit` [35, 17, 36] is a software environment consisting of a collection of algorithms for direct optimal control. Its efficient structure, that permits low computational times, in combination with the use of tailored algorithms, like RTI, make it suitable for real-time NMPC. Its main advantage is that it represents the functions symbolically using an operator-based-tree form. In this way, the specific structure of a function can be efficiently detected and exploited, permitting also numerical, symbolic and automatic differentiation and C-code generation.

Regarding the code generation tool of the `ACADO Toolkit`, it receives the NMPC formulation in order to prepare an optimized C-code. Specifically, it exports C-code with the elements of NMPC (the user-specified functions) and the associated derivatives with respect to the state and control inputs, which are symbolically simplified. Following, an integration routine is chosen and based on this a discretization algorithm is prepared using either a single or multiple shooting method together with appropriate algebra routines for condensing. For the resulting NLP, the RTI scheme is autogenerated, binded with an appropriate solver for the resulting dense QP that uses either an interior-point or an active-set method. The resulting C-code is self-contained, is based on hard-coded dimensions and makes use of only static memory. Thus it is efficient for real-time implementations.

Successor of the ACADO Toolkit is the acados [34]. Although the idea behind the two software is similar, there are notable differences. Instead of the use of expression-trees for the symbolic representation, acados is based on CasADi [37] that uses expression graphs that often lead to a smaller and faster code, as the resulting instruction sequences are reportedly shorter. Moreover, regarding the linear algebra, in ACADO Toolkit it is code-generated, which is an efficient approach when matrices of low dimensions are involved, while it keeps the generated code independent of external libraries. On the contrary, acados is based on the linear algebra package BLASFEO [38] that incorporates optimized linear algebra routines and is efficient for matrices of relatively higher dimensions (reportedly, acados outperforms ACADO on medium-scale problems [34]). Finally, a variety of SQP-like methods (including the RTI scheme) and a variety of numerical simulation routines and QP solvers give flexibility to the user.

2.4 Conclusions

This chapter offered some preliminaries for the NMPC schemes that will be used for real-time motion generation in the following chapters. Starting from a general infinite-dimensional OCP the procedure for its solution using the direct multiple shooting was described. After describing an SQP algorithm for the resulting NLP, the RTI scheme was presented for the solution of the NMPV in real-time. Finally, the two software packages that will be used in the following chapters for the formulation and solution of the NMPC were briefly presented.

Part I

Motion generation for mobile robots

Chapter 3

Mobile robot navigation

The first part of this thesis focuses on the mobile robot navigation in environments populated by static and/or moving obstacles.

Mobile robots are used in numerous applications, ranging from house cleaning, surveillance and parcel delivery to search and rescue operations or exploration of human-hostile environments. When the robot has to operate autonomously, navigation is at the heart of the considered operation. According to the definition given in [39], *navigation is the problem of finding a collision-free motion for the robot system from one configuration (or state) to another.*

The mobile robot navigation has been studied extensively in the past years, however, is still an active field of research. A comprehensive review of some of the most commonly used methods can be found in [40, 41]. Classic approaches are based on *graph search* like A* [42], *sampling-based techniques* like the Probabilistic Roadmap (PRM) [43] and the Rapidly-exploring Random Tree (RRT) [44, 45] or its optimal version RRT* [46]. In addition, the motion planning problem can be also formulated as a mathematical program that can be solved by a numerical optimization solver [47]. Note that these methods require full knowledge of the environment and a significant amount of time to output the resulting path. For this reason, they are characterized as *offline* methods. However, in most of the cases one cannot have complete knowledge of the environment, which might also be populated by moving obstacles (like other robots or humans) whose motion is not known in advance. As a result, a motion planning method has to be able to adapt to the environment changes when they occur. With this in mind *online* approaches were implemented. Among them, an extension of RRT with a real-time flavor [48], approaches like the *dynamic-window approach* [49], *velocity obstacles* [50], as well as the *artificial potential fields* [51].

Recently, NMPC has become a rather famous alternative for mobile robot navigation in environments populated by static and/or moving obstacles. Solving an OCP at each control cycle and by using the robot dynamic model as a prediction model along with appropriate state and input constraints, NMPC can generate kinodynamically feasible motions that adapt to the changes of the environments, while since the OCP is defined over a time horizon, the resulting motion have also a notion of look-ahead.

This chapter offers some preliminary material and an introductory discussion on

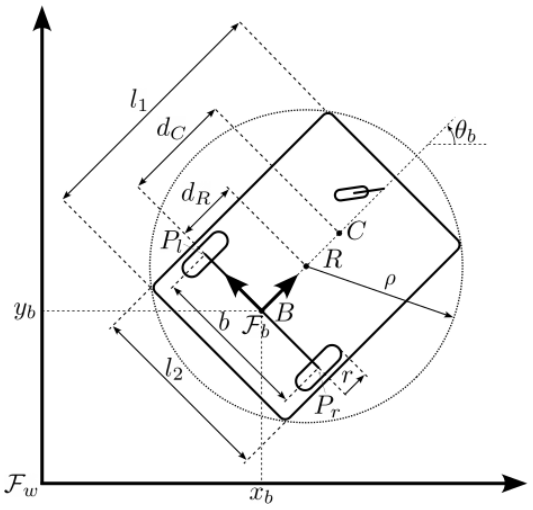

Figure 3.1. A general differential-drive mobile robot. The robot consists of two driving wheels and a caster wheel for mechanical balance.

the use of NMPC as a motion generation approach for mobile robot navigation. Its purpose is to pave the road to the methods proposed in Chapters 4 and 5 for the solution of the robot navigation problem. Note that, although some of the concepts presented in this and Chapters 4 and 5 are applicable to most mobile robots (i.e., ground, aerial, under-water), this thesis only focuses on wheeled mobile robots.

The chapter is organized as follows. In Sect. 3.1 we formulate a general navigation problem for a mobile robot that operates in environments populated by static and/or moving obstacles. Sect. 3.2 obtains the kinematic and dynamic model of a commonly used type of wheeled mobile robot, the *differential-drive robot*, that it is going to be used as a testing platform for the methods proposed in the following chapters. Sect. 3.3 presents an NMPC method for robot navigation, complete with a discussion about the collision avoidance constraint, which is at the heart of the robot navigation problem. We focus our attention on the way in which the structure of the collision avoidance constraint affects the robot safety within an NMPC setup. Finally, Sect. 3.4 offers some concluding remarks.

3.1 Problem formulation

In this section, we will formulate a general version of the mobile robot navigation problem. Note that the particular details of the problem can vary depending on the environment in which the robot moves, the robot itself[1], as well as the available information about the robot state and the state of the environment obstacles.

Consider the robot illustrated in Fig. 3.1 with configuration $\boldsymbol{q}$. The robot moves on horizontal ground, in a workspace $\mathcal{W}$ populated by static and/or moving obstacles.

[1]In many practical applications, the motions required by the mobile robot to perform are relatively slow, without the need of excessive accelerations. As a result, the use of the kinematic model of the robot is adequate. On the other hand, in cases where the robot performs aggressive motions and its dynamic characteristics (e.g., mass and moment of inertia) are significant, then the use of the dynamic model is required.

We denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$ and by $\mathcal{O}(t) \subset \mathcal{W}$ the volume occupied by the obstacles at time t.

A navigation task is assigned to the robot in terms of a vector $\boldsymbol{y} \in \mathcal{Y}$ which describes the position of a representative point C (see Fig. 3.1) on the robot and is related to the configuration via a forward kinematic map $\boldsymbol{y} = \boldsymbol{k}(\boldsymbol{q})$. The task is to drive C to a goal region $\mathcal{G}$. The problem consists in generating in real-time a motion that

1. drives the robot from its starting configuration to a configuration realizing the task, i.e., brings C in the goal region $\mathcal{G}$;
2. is consistent with the considered model of the robot and respects existing bounds on the robot velocity and the considered control inputs;
3. always avoids collisions between the robot and the obstacles, i.e., $\mathcal{R}(\boldsymbol{q}) \cap \mathcal{O}(t) = \emptyset$ for all t.

3.2 Mobile robot model

As it was already mentioned, this analysis focuses on a general differential-drive mobile robot as the one depicted in Fig. 3.1. The considered robot consists of two driving wheels and a caster wheel for mechanical balance. For our analysis we make the following assumptions:

- the robot bodies are rigid;
- the robot wheels are in *point contact* with the ground, while the contact points that correspond to the driving wheels do not change with respect to a reference frame attached to the vehicle;
- the robot maintains its balance, in the sense that its wheels remain always in contact with the ground (which is a reasonable assumption for most conventional wheeled mobile robots whose center of mass is close to the ground);
- the ground-wheel friction is adequate to prevent slippage.

Referring to Fig. 3.1, we denote by $\mathcal{F}_w$ the world reference frame, by $B(x_b, y_b)$ the midpoint of the line segment joining the two driving wheels with x_b and y_b its Cartesian coordinates, by $P_r(x_r, y_r)$ and $P_l(x_l, y_l)$ the contact points of the right and left wheel with the ground. With b we denote the distance between the contact points of the driving wheels. Let us also denote the reference frame $\mathcal{F}_b$ with its origin attached at B and its x-axis aligned with the sagittal axis of the robot.

The robot *generalized coordinates* are collected in the vector

$$\boldsymbol{q} = (x_b, y_b, \theta_b, \phi_r, \phi_l) \in \mathbb{R}^n$$

where (x_b, y_b, θ_b) define the pose of the mobile robot on the horizontal plane, with θ_b being the orientation of the vehicle (and consequently of the two driving wheels) with respect to the world frame $\mathcal{F}_w$, and ϕ_r and ϕ_l the joint angle of the right and left driving wheels respectively.

The position vector of P_r and P_l can be expressed in the world frame as

$$\boldsymbol{p}_r = \begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} x_b + b/2 \sin\theta_b \\ y_b - b/2 \cos\theta_b \end{pmatrix} \tag{3.1a}$$

$$\boldsymbol{p}_l = \begin{pmatrix} x_l \\ y_l \end{pmatrix} = \begin{pmatrix} x_b - b/2 \sin\theta_b \\ y_b + b/2 \cos\theta_b \end{pmatrix} \tag{3.1b}$$

while their velocity as

$$\dot{\boldsymbol{p}}_r = \begin{pmatrix} \dot{x}_r \\ \dot{y}_r \end{pmatrix} = \begin{pmatrix} \dot{x}_b + b/2\, \dot{\theta}_b \cos\theta_b \\ \dot{y}_b + b/2\, \dot{\theta}_b \sin\theta_b \end{pmatrix} \tag{3.2a}$$

$$\dot{\boldsymbol{p}}_l = \begin{pmatrix} \dot{x}_l \\ \dot{y}_l \end{pmatrix} = \begin{pmatrix} \dot{x}_b - b/2\, \dot{\theta}_b \cos\theta_b \\ \dot{y}_b - b/2\, \dot{\theta}_b \sin\theta_b \end{pmatrix} \tag{3.2b}$$

3.2.1 Kinematic constraints

The robot is in contact with the ground through its wheels. The forces exerted by the ground to the robot at the contact points of the wheels constrain the robot motion.

No lateral wheel motion The assumption for adequate wheel-ground friction suggests that the robot wheels are prevented from slipping sideways (i.e., to a direction orthogonal to the robot sagittal plane). So, for the two wheels we have the following constraints

$$-\dot{x}_r \sin\theta_b + \dot{y}_r \cos\theta_b = 0 \tag{3.3a}$$

$$-\dot{x}_l \sin\theta_b + \dot{y}_l \cos\theta_b = 0 \tag{3.3b}$$

After substituting (3.2) in (3.3) we get

$$\dot{x}_b \sin\theta_b - \dot{y}_b \cos\theta_b = 0 \tag{3.4a}$$

$$\dot{x}_b \sin\theta_b - \dot{y}_b \cos\theta_b = 0 \tag{3.4b}$$

We can notice that the no lateral motion condition for each wheel leads to the same constraint. This is due to the structure of the differential-drive robot.

Pure rolling condition The pure rolling condition suggests that the translational motion of the wheels is a result of its rotation around its axis only. Again this is a result of the assumption that the friction is adequate to prevent slippage. This leads to the following conditions that connect the wheel angular velocity around its axis to the wheel translational velocity (and consequently to the robot velocity) in the world frame.

$$\dot{x}_r \cos\theta_b + \dot{y}_r \sin\theta - r\dot{\phi}_r = 0 \tag{3.5a}$$

$$\dot{x}_l \cos\theta_b + \dot{y}_l \sin\theta - r\dot{\phi}_l = 0 \tag{3.5b}$$

by substituting (3.2) in (3.5) we get

$$\dot{x}_b \cos\theta_b + \dot{y}_b \sin\theta_b + \dot{\theta}_b b/2 - r\dot{\phi}_r = 0 \tag{3.6a}$$

$$\dot{x}_b \cos\theta_b + \dot{y}_b \sin\theta_b - \dot{\theta}_b b/2 - r\dot{\phi}_l = 0 \tag{3.6b}$$

The constraints (3.4), (3.6a) and (3.6b) constitute the $k = 3$ linearly independent kinematic constraints that can be written in *Pfaffian form* as

$$\underbrace{\begin{pmatrix} \sin\theta_b & -\cos\theta_b & 0 & 0 & 0 \\ \cos\theta_b & \sin\theta_b & b/2 & -r & 0 \\ \cos\theta_b & \sin\theta_b & -b/2 & 0 & -r \end{pmatrix}}_{\boldsymbol{A}^T(\boldsymbol{q}) \in \mathbb{R}^{k\times n}} \underbrace{\begin{pmatrix} \dot{x}_b \\ \dot{y}_b \\ \dot{\theta}_b \\ \dot{\phi}_r \\ \dot{\phi}_l \end{pmatrix}}_{\dot{\boldsymbol{q}}} = \mathbf{0} \tag{3.7}$$

3.2.2 Kinematic model

At a given configuration $\boldsymbol{q}$, the generalized velocities $\dot{\boldsymbol{q}}$ lie at the *null space* of matrix $\boldsymbol{A}^T(\boldsymbol{q})$ and can be expressed as

$$\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \tag{3.8}$$

where $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n\times m}$ a matrix whose columns span the null space of $\boldsymbol{A}^T(\boldsymbol{q})$, with $m = n - k$ and $\boldsymbol{\nu} \in \mathbb{R}^m$ a velocity input vector. We will refer to vector $\boldsymbol{\nu}$ as the vector of the robot *pseudovelocities*, in order to distinguish it from the generalized velocities $\dot{\boldsymbol{q}}$. Equation (3.8) is referred to as the *kinematic model* of the robot.

Note that the choice of $\boldsymbol{G}(\boldsymbol{q})$ is not unique and it can be chosen in such a way that the velocity input $\boldsymbol{\nu}$ has a specific physical meaning. Particularly, one can use as an input the angular velocity of the two wheels, $\boldsymbol{\nu} = (\dot{\phi}_r, \dot{\phi}_l)$ by choosing

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{pmatrix} r/2\cos\theta_b & r/2\cos\theta_b \\ r/2\sin\theta_b & r/2\sin\theta_b \\ r/b & -r/b \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{3.9}$$

Alternatively, by choosing $\boldsymbol{G}(\boldsymbol{q})$ as

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{pmatrix} \cos\theta_b & 0 \\ \sin\theta_b & 0 \\ 0 & 1 \\ 1/r & b/(2r) \\ 1/r & -b/(2r) \end{pmatrix} \tag{3.10}$$

one can use the input vector $\boldsymbol{\nu} = (\upsilon, \omega)$ where υ is the robot *driving velocity*[2] and ω its *steering velocity*[3]. Note that the driving and steering velocities are connected to the wheels angular velocities via

$$\upsilon = \frac{r(\dot{\phi}_r + \dot{\phi}_l)}{2}, \qquad \omega = \frac{r(\dot{\phi}_r - \dot{\phi}_l)}{b}.$$

[2]As driving velocity we define the velocity of the robot along its longitudinal axis.
[3]As steering velocity we define the angular velocity around the vertical axis at B.

Differentiation of (3.8) with respect to time gives a relationship between the generalized accelerations and the time derivative of the pseudovelocities that we will refer to them as *pseudo-accelerations*

$$\ddot{\boldsymbol{q}} = \dot{\boldsymbol{G}}(\boldsymbol{q})\boldsymbol{\nu} + \boldsymbol{G}(\boldsymbol{q})\dot{\boldsymbol{\nu}}. \tag{3.11}$$

Representative points on the robot In principle any point on the robot is related to the robot configuration $\boldsymbol{q}$ via a forward kinematic map. Referring to Fig. 3.1 consider the point C which will be used as a representative point for the navigation task. Its position can be described as

$$\boldsymbol{y} = \boldsymbol{p}_c = \begin{pmatrix} x_c \\ y_c \end{pmatrix} = \underbrace{\begin{pmatrix} x_b + d_c \cos\theta_b \\ y_b + d_c \sin\theta_b \end{pmatrix}}_{\boldsymbol{k}(\boldsymbol{q})} \tag{3.12}$$

while its velocity as

$$\dot{\boldsymbol{y}} = \dot{\boldsymbol{p}}_c = \frac{\partial}{\partial \boldsymbol{q}} \boldsymbol{k}(\boldsymbol{q}) \boldsymbol{G}(\boldsymbol{q}) \boldsymbol{\nu}$$

3.2.3 Dynamic model

The dynamic model of the robot provides a mapping between the generalized forces acting on the robot (i.e., actuator forces and external forces) and its motion. In this work, in order to derive the dynamic model of the robot we follow the *Lagrange formulation*. Note that the derivation of the dynamic model of the robot of Fig. 3.1 is relatively simple, as the kinetic energy can be computed even by hand. However, if the structure of the robot was more complex, a systematic approach for the derivation of the kinetic and potential energy, and in extension of the robot equations of motion, is required (see Appendix A).

The following analysis is based on [52]. Given the generalized coordinates $\boldsymbol{q}$, we define the *Lagrangian* of the mechanical system (robot) as the difference between its kinetic energy $\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and its potential energy $\mathcal{U}(\boldsymbol{q})$

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \mathcal{U}(\boldsymbol{q}) \tag{3.13}$$

The kinetic energy of the system is the sum of the kinetic energies of each individual robot body. In the particular case of the considered mobile robot (Fig. 3.1), its kinetic energy is the sum of the kinetic energy of the base (i.e., the main vehicle) and the two driving wheels[4]. Regarding the potential energy, our initial assumption that the robot moves on an horizontal ground suggests that $\mathcal{U}(\boldsymbol{q}) = 0$.

Having computed the Lagrangian as a function of the generalized coordinates and the generalized velocities, the Lagrange equations for the kinematically constrained mobile robot are

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}}\right)^T - \left(\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}}\right)^T = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{\tau} + \boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda} \tag{3.14}$$

[4]The kinetic energy of the caster wheels can be ignored by simply considering them as part of the main vehicle for a fixed configuration.

where $\boldsymbol{\tau} \in \mathbb{R}^{n_\tau}$ is the vector of the forces/moments applied by the n_τ robot actuators[5], $\boldsymbol{S}(\boldsymbol{q}) \in \mathbb{R}^{n \times n_\tau}$ the matrix that maps the actuator forces to forces performing work on the generalized coordinates, $\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}$ the vector of the forces[6] exerted to the robot by its contact with the ground at the generalized coordinates level, $\boldsymbol{\lambda} \in \mathbb{R}^k$ the vector of the *Lagrange multipliers*.

From (3.13) and (3.14) we get the dynamic model of the robot as

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{\tau} + \boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda} \tag{3.15a}$$

$$\boldsymbol{A}^T(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0} \tag{3.15b}$$

where $\boldsymbol{B}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ is the robot inertia matrix, while $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^n$ is the vector of velocity and gravitational terms for which holds

$$\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{B}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \boldsymbol{q}}\left(\dot{\boldsymbol{q}}^T \boldsymbol{B}(\boldsymbol{q})\dot{\boldsymbol{q}}\right)\right)^T + \left(\frac{\partial \mathcal{U}(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)^T. \tag{3.16}$$

Note that in the considered case the term that corresponds to gravitational forces is zero since $\mathcal{U}(\boldsymbol{q}) = 0$.

One can eliminate the Lagrange multipliers by left-multiplying both sides of (3.15a) by $\boldsymbol{G}^T(\boldsymbol{q})$, leading to the *reduced-dynamics model*

$$\boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{S}(\boldsymbol{q})\boldsymbol{\tau}. \tag{3.17}$$

By also substituting (3.11) in (3.17) we get

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu}) = \boldsymbol{E}(\boldsymbol{q})\boldsymbol{\tau} \tag{3.18}$$

where

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q}) &= \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q})\boldsymbol{G}(\boldsymbol{q}) \\ \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu}) &= \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q})\dot{\boldsymbol{G}}(\boldsymbol{q})\boldsymbol{\nu} + \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ \boldsymbol{E}(\boldsymbol{q}) &= \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{S}(\boldsymbol{q}). \end{aligned}$$

By substituting the kinematic constraints (3.15b) by the kinematic model (3.8) and since $\boldsymbol{M}(\boldsymbol{q})$ is positive definite, we can obtain the *state-space reduced model* as

$$\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \tag{3.19a}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}(\boldsymbol{q})\left(\boldsymbol{E}(\boldsymbol{q})\boldsymbol{\tau} - \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu})\right) \tag{3.19b}$$

3.3 Motion generation via NMPC

To generate the desired motion we will use an NMPC algorithm for real-time motion generation. The proposed NMPC that will be presented in this section is the base for the NMPC approaches for robot navigation that will be presented in Chapters 4 and 5.

[5] In this work we assume that the number of the actuators is equal to the number of the robot DOF, i.e., $n_\tau = m = n - k$.

[6] Note that these are the forces that enforce the kinematic constraints to the robot.

The considered robot model that will be used as a prediction model will be expressed as

$$\dot{\boldsymbol{x}} = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}) \tag{3.20}$$

where $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{\nu})$ is the robot state and $\boldsymbol{u}$ the robot control inputs. Note that the control inputs depend on the model that we use. Particularly, if we consider only the kinematic model (3.8) we can consider inputs at the pseudo-acceleration level, i.e., $\boldsymbol{u} = \dot{\boldsymbol{\nu}}$. Alternatively, if we consider the state-space reduced model (3.19) then the considered inputs correspond to the torques applied by the robot actuators, i.e., $\boldsymbol{u} = \boldsymbol{\tau}$.

Consider the prediction horizon T, the sampling time δ and the number of the subintervals $N = T/\delta$. At the generic time instant t_k, we want the NMPC to generate over the time interval $[t_k, t_k + T]$ a motion that drives the task error to zero with the least possible control effort, while maintaining the robot safety.

Denote by $\boldsymbol{x}_{k|i}$ and $\boldsymbol{u}_{k|i}$ the predicted robot state and control inputs at time t_{k+i}, by $\boldsymbol{y}_d$ the position of a representative point in $\mathcal{G}$ and by $\boldsymbol{y}_{k|i}$ and $\dot{\boldsymbol{y}}_{k|i}$ the predicted position and velocity of C at t_{k+i}. Denoting also by $\boldsymbol{e}_{k|i} = \boldsymbol{y}_d - \boldsymbol{y}_{k|i}$ the predicted task error at time t_{k+i}, the running and terminal costs can be expressed respectively as

$$V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{e}_{k|i}^T \boldsymbol{Q} \boldsymbol{e}_{k|i} + \dot{\boldsymbol{y}}_{k|i}^T \boldsymbol{S} \dot{\boldsymbol{y}}_{k|i} + \boldsymbol{u}_{k|i}^T \boldsymbol{R}\, \boldsymbol{u}_{k|i},$$

$$V_{k|N}(\boldsymbol{x}_{k|N}) = \boldsymbol{e}_{k|N}^T \boldsymbol{Q}_N \boldsymbol{e}_{k|N} + \dot{\boldsymbol{y}}_{k|N}^T \boldsymbol{S}_N \dot{\boldsymbol{y}}_{k|N},$$

where $\boldsymbol{Q}$, $\boldsymbol{S}$ and $\boldsymbol{R}$ are weighting matrices of appropriate dimensions for the predicted task error, the velocity of C and the control effort throughout the prediction horizon, while $\boldsymbol{Q}_N$ and $\boldsymbol{S}_N$ are the weighting matrices for the predicted task error and the velocity of C at the final time instant.

The resulting NLP to be solved at time t_k will be

$$\min_{\substack{\boldsymbol{x}_{k|0},\ldots,\boldsymbol{x}_{k|N},\\ \boldsymbol{u}_{k|0},\ldots,\boldsymbol{u}_{k|N-1}}} \sum_{i=0}^{N-1} V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) + V_{k|N}(\boldsymbol{x}_{k|N}) \tag{3.21a}$$

subject to:

$$\boldsymbol{x}_{k|0} - \boldsymbol{x}_k = \boldsymbol{0} \tag{3.21b}$$

$$\boldsymbol{x}_{k|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0, ..., N-1 \tag{3.21c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_{k|i} \leq \boldsymbol{x}_{\max}, \quad i = 0, ..., N \tag{3.21d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_{k|i} \leq \boldsymbol{u}_{\max}, \quad i = 0, ..., N-1 \tag{3.21e}$$

$$\textit{collision avoidance constraints at } t_k, ..., t_{k+N} \tag{3.21f}$$

where $\boldsymbol{x}_k$ the state of the robot at time t_k, $\boldsymbol{\phi}_{\text{d-t}}(\cdot, \cdot)$ the discrete-time model of the robot obtained via numerical integration of (3.20) under the assumption of piecewise-constant control inputs, while $\boldsymbol{x}_{\min}$,$\boldsymbol{x}_{\max}$ and $\boldsymbol{u}_{\min}$,$\boldsymbol{u}_{\max}$ are the lower and upper bounds of the considered robot state and control inputs respectively.

Clearly, in order for the robot to navigate safely in an environment occupied by obstacles, collision avoidance constraints have to be added to the NMPC scheme. In the following, we offer a discussion about the collision avoidance constraints.

3.3.1 The collision avoidance constraint

In order to ensure that the generated motion will be collision-free we have to include appropriate collision avoidance constraints. Let us consider the j-th obstacle whose volume is denoted by $\mathcal{O}_j(t) \subset \mathcal{O}(t)$. Typically a collision avoidance constraint aims at limiting the robot-obstacle relative motion in such a way that the volume $\mathcal{R}(\boldsymbol{q})$ of the robot and the volume of the obstacle do not overlap, i.e.,

$$\mathcal{R}(\boldsymbol{q}) \cap \mathcal{O}_j(t) = \emptyset \quad \forall j = 1, ..., n_o. \tag{3.22}$$

with n_o the number of obstacles. In principle the constraint (3.22) is non-convex and non-differentiable [53, 54]. As a result, (3.22) cannot be included directly in (3.21f) and so a reformulation or an approximation of this constraint is required.

Clearly, the particular structure of the environment plays an important role in the formulation of the collision avoidance constraint. In some *structured environments*[7] one can leverage the particular features and treat the problem as a corridor navigation problem after proper reformulation of the equations of motion, see [55, 56]. In such a way the collision avoidance constraint can be substituted by a state bound constraint of the form (3.21d) with the upper and lower bounds set at each control cycle in such a way that they form a "corridor" that does not contain any obstacle inside. However, in the general case this is not possible and one has to include explicit collision avoidance constraints. In [54] the authors proposed a reformulation of the generic collision avoidance constraint (3.22) as a smooth nonlinear version, appropriate for numerical optimization. In [57, 58] *signed distance fields* are used in order to generate collision-free trajectories. In [59], a soft collision avoidance constraint is introduced that considers obstacles described by general non-convex sets. Other approaches aim at the approximation of (3.22) by enveloping the robot and the obstacles with bounding spheres or ellipsoids [60, 61, 62, 63] and evaluating whether the two bodies collide using a quadratic inequality that corresponds to the inequality of the ellipsoid.

It should be noted that all the aforementioned collision avoidance constraints depend only on distance information between the robot and the obstacles, without any consideration about the rest of the robot state. We will refer to this kind of methods as *distance-based* methods. In an NMPC, a distance-based constraint is active, and as a result affects the generated motion, only if the obstacle is reachable by the prediction horizon. In that case, the NMPC can "see" the obstacle and generate a motion that avoids the collision with it. So, one can notice that the effectiveness of such methods depends on the length of the prediction horizon since the longer the prediction horizon, the earlier an imminent collision can be detected and averted without significant (sometimes prohibitive) actuation effort.

On the other hand, real-time performance obviously does not allow arbitrarily long prediction horizons. In practice, high-speed navigation requires the use of the robot dynamic model and high control frequency. Moreover, the presence of moving obstacles with unpredictable motion also requires fast control cycles in order to use the latest information about the environment. As a result, the maximum achievable

[7] With the term *structured* here, we refer to environments that contain an underlying structure (i.e., a city road or the corridor of a building).

prediction horizon on typical robot processing platforms ends up being relatively short. Consequently, the use of a purely distance-based collision avoidance constraint may jeopardize the robot safety, since the danger of collision may be detected at a time when the robot does not have the necessary actuation power to prevent it.

In an attempt to increase the area in which the collision avoidance constraint affects the solution of the NMPC, authors exploited methods in which the volume of the bounding geometries used in order to envelop the robot and the obstacles for the collision avoidance constraint, increases throughout the prediction horizon [62].

Alternatively, other approaches attack the problem of collision avoidance via appropriate constraints on Control Barrier Functions (CBFs). CBFs are a tool for attaining forward-invariance of a set via an appropriate constraint. The interested reader is referred to the works [64, 65, 66] for an overview on CBFs. In the context of mobile robot navigation, one can define a *safe-set* that contains robot states in which the robot is not in collision with an environment obstacle. If this safe-set is forward-invariant, collision avoidance is guaranteed. So, one can use such constraints on appropriate CBFs in order to enforce collision avoidance. Note that these constraints involve not only the distance between the robot and the obstacle but also its rate of change. Thus, using them in an NMPC setup can be beneficial as the presence of an obstacle will affect the generated motion, even if it is not in the range of the prediction horizon. Examples of CBF-based collision avoidance constraints in an NMPC setup for mobile robot navigation can be found in [67, 68, 69].

The influence of the collision avoidance constraint can be increased if a look-ahead capability is given, by considering both the whole robot state with respect to the obstacles and the robot actuation capabilities. This idea has been already exploited in some classical motion planning methods like the *dynamic window approach* [49] or the *velocity obstacles* [50]. The importance of considering the whole robot state in order to guarantee safety is also stressed in [70] where the concept of *Inevitable Collision States* (ICS) is first introduced. In [71], the idea of ICS is exploited by an anytime motion planning approach based on an RRT. In [72], workspace obstacles are represented as velocity obstacles over a predefined time horizon, whose length depends on the robot state and actuation capabilities and is such that the robot can always avoid collision with the obstacle by a predefined maneuver. Finally, in [73] a forbidden velocity map is defined as the set of the robot prohibited velocities associated with each obstacle based on the state of the robot and its braking capabilities. Nevertheless, these methods require a search in the input space which is not consistent with the nature of the NMPC. A collision avoidance constraint that is based on the robot-obstacle relative state and the worst-case stopping time of the robot joints is introduced in [74] and applied to a linear MPC in [75]. However, the constraint becomes stricter throughout the prediction horizon considering a worst-case scenario in which the obstacle and the robot are moving toward each other with maximum velocity.

3.4 Conclusions

In this chapter, we formulated a general navigation problem in the presence of obstacles for a differential-drive mobile robot, while for the considered robot we

derived its kinematic and dynamic model. For the solution of the considered problem, we proposed an NMPC scheme for real-time motion generation. Special attention was given to the collision avoidance constraint that has to be included in the NMPC scheme in order to enforce safety. We argued that a purely distance-based collision avoidance constraint can jeopardize the robot safety, if in view of real-time performance the prediction horizon is relatively short. On the contrary, collision avoidance constraints that consider additional information regarding the robot-obstacle relative velocity and the robot actuation capabilities are preferable for enforcing safety.

In the following chapters, the proposed NMPC scheme for robot navigation will be used in two navigation problems. In Chap. 4 we consider robot navigation in environments crowded by humans. To enforce safety we use in the NMPC scheme collision avoidance constraints built upon appropriate CBFs. In Chap. 5 we consider a robot navigating in an environment occupied by static and moving obstacles and we introduce a novel collision avoidance constraint to be used in the NMPC, that considers both the robot-obstacle relative state as well as the robot actuation capabilities.

Chapter 4

Crowd navigation using NMPC and control barrier functions

In this chapter, we consider the problem of safe robot navigation in an environment populated by a human crowd. Navigation in a human crowded environment is a common task for robots used in service applications (e.g., parcel delivery, patrolling, vacuum cleaning). As in every navigation task, collision avoidance is at the heart of the problem. However, in the context of human crowd navigation, a collision might not only harm the robot but also the humans populating the environment, whose safety should be the first priority of any robotic application.

This chapter is based on our work, originally presented in [1], for safe robot navigation in a human crowd using NMPC. As we already mentioned in Chap. 3, a purely distance-based collision avoidance constraint within the NMPC can be insufficient due to the short prediction horizons dictated by the real-time performance, risking to jeopardize not only the robot but also the human safety. For this purpose, we consider collision avoidance constraints built upon discrete-time CBFs. The proposed NMPC serves as the motion generation module, that along with a crowd prediction module, constitute a sensor-based scheme for safe robot navigation in a crowd. Regarding the crowd prediction module, it relies on a simple, yet effective, technique based on Kalman Filters (KFs), that predicts the future motion of the humans based on the information acquired by an on-board sensor.

The novelty of this work lies in the use of a real-time NMPC with CBF-based collision avoidance constraint as part of a complete framework for safe robot navigation in a crowd. Extensive simulations show the effectiveness of the proposed method while comparison with a version of the NMPC that uses a collision avoidance constraint based on purely distance information shows the superior performance of CBFs as collision avoidance constraints in an NMPC.

This chapter is organized as follows. Sect. 4.1 formulates the considered crowd navigation problem, while Sect. 4.2 gives an overview of the proposed framework. The crowd prediction module is presented in Sect. 4.3 followed by the motion generation module which is presented in Sect. 4.4. In Sect. 4.5 we present simulation results and in Sect. 4.6 we offer some concluding remarks.

Figure 4.1. An instance of the considered problem. The robot must safely navigate in the crowd of moving humans to reach the goal region (in yellow).

4.1 Problem formulation

Consider the mobile robot of Fig. 3.1 with configuration $\boldsymbol{q}$ moving in a workspace $\mathcal{W}$ populated by a crowd of humans (see Fig. 4.1). The robot motion can be described by the kinematic model (3.8). Denote by $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{\nu}) \in \mathcal{D}$ the robot state, with the pseudovelicities $\boldsymbol{\nu}$ comprised by the robot driving and steering velocity, i.e., $\boldsymbol{\nu} = (\upsilon, \omega)$. If we consider the robot being controlled at the wheel acceleration level, i.e., $\boldsymbol{u} = (\ddot{\phi}_r, \ddot{\phi}_l)$, then the considered robot model can be expressed as

$$\dot{\boldsymbol{x}} = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}) = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \\ \frac{r}{2}(\ddot{\phi}_r + \ddot{\phi}_l) \\ \frac{r}{b}(\ddot{\phi}_r - \ddot{\phi}_l) \end{pmatrix}. \tag{4.1}$$

We assume that the robot is always aware of its own state $\boldsymbol{x}$.

Denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$ and by $\mathcal{H}(t) \subset \mathcal{W}$ the volume occupied by the humans at time t. The robot is equipped with an on-board sensor – in particular, a laser rangefinder – through which it continuously acquires information about its surrounding. This information, typically in the form of a range-bearing pair, can be readily transcribed into a set $\mathcal{S}$ of the measured relative position of points on the humans detected within the sensor field-of-view (FOV).

The robot is assigned a navigation task in terms of a vector $\boldsymbol{y} \in \mathcal{Y}$ which describes the position of the representative point C of the robot (see Fig. 3.1). The task is to drive C to a goal region $\mathcal{G}$. The problem of safe navigation in a crowd consists in generating in real-time a motion, that based on the available sensory information $\mathcal{S}$,

1. drives the robot from an initial configuration to a configuration that realizes the task, i.e., the representative point C reaches the goal region $\mathcal{G}$;

2. is consistent with model (4.1) and respects existing limitations on both state and input variables;

3. always avoids collisions with humans, i.e., $\mathcal{R}(\boldsymbol{q}(t)) \cap \mathcal{H}(t) = \emptyset$ at all time instants t.

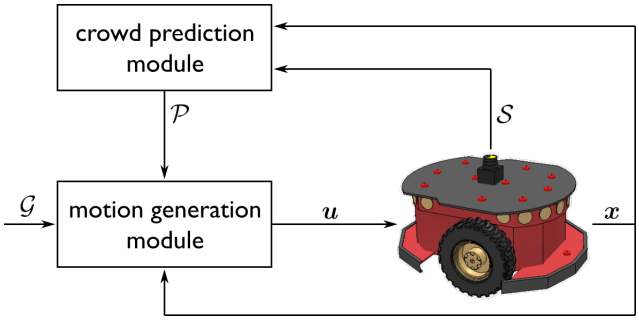

Figure 4.2. Block scheme of the proposed safe navigation framework.

We emphasize that for safe navigation knowledge about the humans future motion is essential. However, in the considered context, this knowledge is not available and has thus to be predicted based on the available sensory information. This aspect is therefore part of the considered problem.

4.2 Proposed framework

To solve the considered problem, we propose the safe navigation framework outlined in the block scheme of Fig. 4.2. It works in a digital fashion over sampling intervals of duration δ and is constituted by two main modules, i.e., the *crowd prediction* and *motion generation* module.

At the generic time instant $t_k = k\delta$, the available sensory information $\mathcal{S}_k$ and the current robot state $\boldsymbol{x}_k$ are passed to the crowd prediction module. Based on this information, the crowd prediction module predicts the motion of a number of surrounding humans over the time interval $[t_k, t_k + T]$, with $T = N\delta$ its duration and N the number of sampling intervals within it. In particular, denote by $\boldsymbol{p}_k^j$ the absolute position of the closest to the robot point of a generic human at time t_k and collect in a vector $\boldsymbol{P}_k^j$ the predicted path of this point over the time interval $[t_k, t_k + T]$

$$\boldsymbol{P}_k^j = (\boldsymbol{p}_{k|0}^j, \dots, \boldsymbol{p}_{k|N}^j),$$

where $\boldsymbol{p}_{k|i}^j$ is the predicted absolute position of the closest to the robot point of the human at $t_{k+i} = (k+i)\delta$ and $\boldsymbol{p}_{k|0}^j = \boldsymbol{p}_k^j$. At each time t_k the crowd prediction module collects the predicted path for M humans in the tuple

$$\mathcal{P}_k = \{\boldsymbol{P}_k^1, \dots, \boldsymbol{P}_k^M\},$$

where M represents the number of humans that will be considered for collision avoidance and is a byproduct of the crowd prediction module. $\mathcal{P}_k$ is the output of the crowd prediction module.

The predicted motion of the M humans, collected in $\mathcal{P}_k$, is fed to the motion generation module, together with the goal region $\mathcal{G}$ and the current robot state $\boldsymbol{x}_k$. The motion generation module relies on a real-time NMPC algorithm to produce wheel acceleration commands $\boldsymbol{u} = (\ddot{\phi}_r, \ddot{\phi}_l)$ for the robot. It uses a prediction horizon T (over which $\mathcal{P}_k$ is defined). In order to avoid collisions with the surrounding

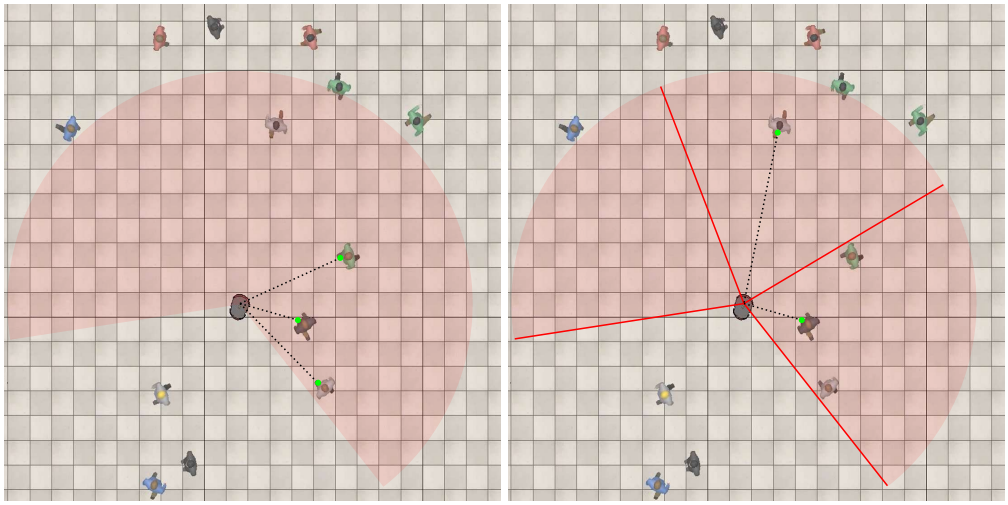

Figure 4.3. Humans selected for collision avoidance by the *K-Neighbors* (left) and *K-Cones* (right) strategy for $K = 3$. Green dots indicate the closest point of the selected humans.

humans the NMPC uses a CBF-based collision avoidance constraint for each of the M humans accounted in $\mathcal{P}_k$.

The focus of this thesis is on the motion generation module and particularly on the CBF-based collision avoidance constraint used within the NMPC. However, the crowd prediction module is an important element of the proposed safe robot navigation framework that also affects design choices within the NMPC. As a result in addition to the motion generation module that is presented in Sect. 4.4, the crowd prediction module will be briefly presented in Sect. 4.3.

4.3 Crowd prediction module

This module receives in input the sensory information $\mathcal{S}_k$ available at t_k and the current robot state $\boldsymbol{x}_k$. It outputs the tuple $\mathcal{P}_k$ with the predicted motion of M humans over $[t_k, t_k + T]$.

To obtain this, it is essential to first choose which humans in the crowd will be considered for collision avoidance, whose future motion must therefore be predicted. We propose two possible strategies to make this choice, which are described in the following and illustrated in Fig. 4.3. Let K be a user-specified maximum number of humans to be considered.

- *K-Neighbors.* This strategy considers for collision avoidance the K closest to the robot humans within the sensor FOV.
- *K-Cones.* This strategy conceptually divides the sensor FOV in K cones of equal detection angle and considers the closest to the robot human within each of them.

The state of the selected K humans at time instant t_k is estimated by means of an array of K KFs, called KF-1, ..., KF-K. In particular, consider the generic l-th human and collect in a vector $\boldsymbol{\chi}_k^l = (\boldsymbol{p}_k^l, \dot{\boldsymbol{p}}_k^l)$ his state at t_k, with $\boldsymbol{p}_k^l$ and $\dot{\boldsymbol{p}}_k^l$ the

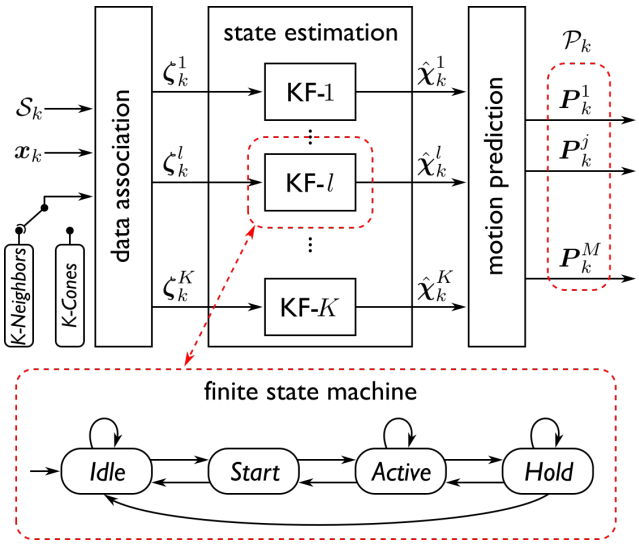

Figure 4.4. Conceptual scheme of the crowd prediction module.

position and velocity of his closest to the robot point, which will be estimated by the associated to the l-th human KF, i.e., KF-l. Assume this closest to the robot point moves omnidirectionally with constant velocity over the sampling intervals; moreover, notice that the measurements include only position information. Then, the state-transition and output models can be represented by the discrete-time stochastic system

$$\boldsymbol{\chi}_{k+1}^l = \underbrace{\begin{pmatrix} \boldsymbol{I}_{2\times 2} & \delta \boldsymbol{I}_{2\times 2} \\ \boldsymbol{0}_{2\times 2} & \boldsymbol{I}_{2\times 2} \end{pmatrix}}_{\boldsymbol{A}} \boldsymbol{\chi}_k^l + \boldsymbol{v}_k \tag{4.2}$$

$$\boldsymbol{\zeta}_k^l = \underbrace{\begin{pmatrix} \boldsymbol{I}_{2\times 2} & \boldsymbol{0}_{2\times 2} \end{pmatrix}}_{\boldsymbol{C}} \boldsymbol{\chi}_k^l + \boldsymbol{w}_k \tag{4.3}$$

where $\boldsymbol{v}_k$, $\boldsymbol{w}_k$ are white Gaussian noises with zero mean and covariance matrices $\boldsymbol{V}_k$, $\boldsymbol{W}_k$, respectively; $\boldsymbol{\zeta}_k^l$ is the measured closest to the robot point of the human which needs to be carefully extracted from the sensory information $\mathcal{S}_k$. Note that, since the available sensory information is non-specific (i.e., it does not embed the notion of human identity), the same KF can, in principle, estimate the state of different humans at different time instants.

The full crowd prediction procedure, outlined in Fig. 4.4, consists of the three stages described in the next subsections.

4.3.1 Data association

The adopted selection strategy (i.e., *K-Neighbors* or *K-Cones*) determines the way in which the measurements extracted from $\mathcal{S}_k$ are associated to the KFs.

When adopting the *K-Neighbors* strategy, an iterative procedure is involved to extract from $\mathcal{S}_k$ a subset $\mathcal{Z}_k$ containing the measured absolute position of the closest to the robot points on the K closest to the robot humans. Such procedure

relies on the assumption that the planar projection of the occupancy volume of any human in the crowd is always contained in a bounding circle of radius ρ_H. At the l-th iteration, the measured relative position of the point that is the closest to the robot is extracted from $\mathcal{S}_k$, expressed in absolute coordinates (using $\boldsymbol{x}_k$), and added to $\mathcal{Z}_k$; then, the center of the bounding circle of the l-th human is computed via simple geometrical considerations and all positions of points lying within it are removed from $\mathcal{S}_k$. The procedure terminates as soon as the number of elements in $\mathcal{Z}_k$ reaches K ($|\mathcal{Z}_k| = K$) or $\mathcal{S}_k$ becomes empty ($\mathcal{S}_k = \emptyset$). Finally, the extracted $|\mathcal{Z}_k|$ measurements are associated to the KFs of the array using the *maximum likelihood technique* (see [76]), while if $|\mathcal{Z}_k| < K$ a void measurement $\boldsymbol{\zeta}_k = \emptyset$ is associated to each of the remaining $K - |\mathcal{Z}_k|$ KFs.

When adopting the *K-Cones* strategy, each cone is separately considered. Denote by $\mathcal{S}_k^l$ the subset of $\mathcal{S}_k$ containing the measured relative position of human points lying within the l-th cone. If this is not empty ($\mathcal{S}_k^l \neq \emptyset$), the measured relative position of the point that is the closest to the robot is extracted; then, expressing it in absolute coordinates (using $\boldsymbol{x}_k$) produces the measurement $\boldsymbol{\zeta}_k^l$ to be associated to KF-l. Otherwise, a void measurement $\boldsymbol{\zeta}_k^l = \emptyset$ is associated to KF-l.

4.3.2 State estimation

The l-th KF receives the associated measurement $\boldsymbol{\zeta}_k^j$ at t_k and has memory of the previous state estimate $\hat{\boldsymbol{\chi}}_{k-1}^l$, the last valid measurement $\bar{\boldsymbol{\zeta}}^l$ that it received and the time instant $\bar{t}^l$ at which this was received. Each of $\boldsymbol{\zeta}_k^j$ and $\hat{\boldsymbol{\chi}}_{k-1}^l$ can be either valid ($\boldsymbol{\zeta}_k^j \neq \emptyset$, $\hat{\boldsymbol{\chi}}_{k-1}^l \neq \emptyset$) or void ($\boldsymbol{\zeta}_k^j = \emptyset$, $\hat{\boldsymbol{\chi}}_{k-1}^l = \emptyset$). Moreover, KF-$l$ has an associated finite state machine (FSM), depicted in Fig. 4.4, that governs its operation. It consists of four states, i.e., *Idle*, *Start*, *Active*, *Hold*, that are described in the following, together with the actions to be taken according to the received measurement and the KF memory in order to produce the state estimate $\hat{\boldsymbol{\chi}}_k^l$.

- *Idle*. The KF is inactive, i.e., $\hat{\boldsymbol{\chi}}_{k-1}^l = \emptyset$.
  - If $\boldsymbol{\zeta}_k^l \neq \emptyset$, $\hat{\boldsymbol{\chi}}_k^l$ is initialized as $\hat{\boldsymbol{\chi}}_k^l = (\boldsymbol{\zeta}_k^l, \boldsymbol{0})$ and the FSM state becomes *Start*.
  - Otherwise, $\hat{\boldsymbol{\chi}}_k^l = \emptyset$ and the FSM state remains *Idle*.
- *Start*. The KF is initializing the state estimate.
  - If $\boldsymbol{\zeta}_k^l \neq \emptyset$, $\hat{\boldsymbol{\chi}}_k^l$ is set as $\hat{\boldsymbol{\chi}}_k^l = (\boldsymbol{\zeta}_k^l, \frac{1}{\delta}(\boldsymbol{\zeta}_k^l - \hat{\boldsymbol{p}}_{k-1}^l))$ and the FSM state becomes *Active*.
  - Otherwise, $\hat{\boldsymbol{\chi}}_k^l = \emptyset$ and the FSM state becomes *Idle*.
- *Active*. The KF is actively generating the state estimate based on valid measurements.
  - If $\boldsymbol{\zeta}_k^l \neq \emptyset$, $\hat{\boldsymbol{\chi}}_k^l$ is generated applying the standard prediction-correction procedure. First, the state prediction and covariance matrix are generated as

$$\begin{aligned} \hat{\boldsymbol{\chi}}_{k|k-1}^l &= \boldsymbol{A}\hat{\boldsymbol{\chi}}_{k-1}^l \\ \boldsymbol{\Sigma}_{k|k-1}^l &= \boldsymbol{A}\boldsymbol{\Sigma}_{k-1}^l\boldsymbol{A}^T + \boldsymbol{V}_k. \end{aligned} \tag{4.4}$$

Then, the innovation $\boldsymbol{z}_k^l$, i.e., the difference between the measured and the predicted output, is computed as

$$\boldsymbol{z}_k^l = \boldsymbol{\zeta}_k^l - \boldsymbol{C}\hat{\boldsymbol{\chi}}_{k|k-1}^l \tag{4.5}$$

and it is checked whether its norm does not exceed a predefined threshold $\bar{z}$.

* If $\left\|\boldsymbol{z}_k^l\right\| < \bar{z}$, the state estimate and covariance matrix are corrected as

$$\begin{aligned} \hat{\boldsymbol{\chi}}_k^l &= \hat{\boldsymbol{\chi}}_{k|k-1}^l + \boldsymbol{\Gamma}_k^l \boldsymbol{z}_k^l \\ \boldsymbol{\Sigma}_k^l &= \boldsymbol{\Sigma}_{k|k-1}^l - \boldsymbol{\Gamma}_k^l \boldsymbol{C} \boldsymbol{\Sigma}_{k|k-1}^l, \end{aligned} \tag{4.6}$$

  with $\boldsymbol{\Gamma}_k^l$ the *Kalman gain matrix* computed as

$$\boldsymbol{\Gamma}_k^l = \boldsymbol{\Sigma}_{k|k-1}^l \boldsymbol{C}^T \left(\boldsymbol{C}\boldsymbol{\Sigma}_{k|k-1}^l \boldsymbol{C}^T + \boldsymbol{W}_k\right)^{-1}$$

  and the FSM state remains *Active*.

* Otherwise, $\hat{\boldsymbol{\chi}}_k^l$ is reset as $\hat{\boldsymbol{\chi}}_k^l = (\boldsymbol{\zeta}_k^l, \hat{\boldsymbol{p}}_{k-1}^l)$ and the FSM state becomes *Start*.

– Otherwise, $\hat{\boldsymbol{\chi}}_k^l$ is generated using (4.4–4.6) by setting $\boldsymbol{\zeta}_k^l = \bar{\boldsymbol{\zeta}}^l$ in (4.5) and the FSM state becomes *Hold*.

- *Hold*. The KF is temporarily generating the state estimate based on the last valid measurement $\bar{\boldsymbol{\zeta}}^l$.

  – If $\boldsymbol{\zeta}_k^l \neq \emptyset$, $\hat{\boldsymbol{\chi}}_k^l$ is generated using (4.4–4.6) and the FSM state becomes *Active*.

  – Otherwise:

    * if $t_k \leq \bar{t}^l + \bar{T}$, with $\bar{T}$ a predefined hold period, $\hat{\boldsymbol{\chi}}_k^l$ is generated using (4.4–4.6) by setting $\boldsymbol{\zeta}_k^l = \bar{\boldsymbol{\zeta}}^l$ in (4.5) and the FSM state remains *Hold*;
    * else, $\hat{\boldsymbol{\chi}}_k^l = \emptyset$ and the FSM state becomes *Idle*.

4.3.3 Motion prediction

Let $\hat{\mathcal{X}}_k = \{\hat{\boldsymbol{\chi}}_k^1, \ldots, \hat{\boldsymbol{\chi}}_k^M\}$ be the set of the M valid state estimates, i.e., $\hat{\boldsymbol{\chi}}_k^j \neq \emptyset$ ($j = 1, \ldots, M$), among the K produced by the KFs. Then, the tuple $\mathcal{P}_k = \{\boldsymbol{P}_k^1, \ldots, \boldsymbol{P}_k^M\}$ is generated by computing each vector $\boldsymbol{P}_k^j = (\boldsymbol{p}_{k|0}^j, \ldots, \boldsymbol{p}_{k|N}^j)$ under the assumption that the corresponding human closest point will move with constant velocity $\hat{\dot{\boldsymbol{p}}}_k^j$ over the time interval $[t_k, t_k + T]$, i.e., $\boldsymbol{p}_{k|i}^j = \hat{\boldsymbol{p}}_k^j + i\delta\hat{\dot{\boldsymbol{p}}}_k^j$ ($i = 0, \ldots, N$).

4.4 Motion generation via NMPC

At the heart of the proposed safe navigation framework is the motion generation module. It is constituted by a real-time NMPC algorithm that at each control cycle receives as an input the predicted humans motion $\mathcal{P}_k$ produced by the crowd

prediction module, the goal region $\mathcal{G}$ and the current robot state $\boldsymbol{x}_k$ and computes the control inputs $\boldsymbol{u}$ to be applied to the robot. In order to ensure that the resulting motion will be collision-free the NMPC uses a discrete-time CBF-based collision avoidance constraint throughout the prediction horizon T for each of the humans accounted in $\mathcal{P}_k$.

In the following, we first present the NMPC algorithm and then describe the adopted collision avoidance constraints.

4.4.1 NMPC algorithm

Consider the NLP to be solved at time t_k. The task is to drive the representative point C of the robot inside the goal region $\mathcal{G}$, possibly with minimum control effort, while maintaining the robot and humans safety. For this purpose, we can use the NLP presented in (3.21), restated here for completeness.

Let $\boldsymbol{x}_{k|i}$ and $\boldsymbol{u}_{k|i}$ be the predicted robot state and control inputs at time t_{k+i}, $\boldsymbol{y}_{k|i}$ and $\dot{\boldsymbol{y}}_{k|i}$ be respectively the predicted position and velocity of C at t_{k+i}, $\boldsymbol{y}_d$ be the position of a representative point of $\mathcal{G}$ and $\boldsymbol{e}_{k|i} = \boldsymbol{y}_d - \boldsymbol{y}_{k|i}$ be the predicted task error at time t_{k+i}.

The NLP to be solved at time t_k is

$$\min_{\substack{\boldsymbol{x}_{k|0},\dots,\boldsymbol{x}_{k|N},\\ \boldsymbol{u}_{k|0},\dots,\boldsymbol{u}_{k|N-1}}} \sum_{i=0}^{N-1} V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) + V_{k|N}(\boldsymbol{x}_{k|N}) \tag{4.7a}$$

subject to:

$$\boldsymbol{x}_{k|0} - \boldsymbol{x}_k = \boldsymbol{0} \tag{4.7b}$$

$$\boldsymbol{x}_{k|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0, ..., N-1 \tag{4.7c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_{k|i} \leq \boldsymbol{x}_{\max}, \quad i = 0, ..., N \tag{4.7d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_{k|i} \leq \boldsymbol{u}_{\max}, \quad i = 0, ..., N-1 \tag{4.7e}$$

$$\textit{collision avoidance constraints at } t_k, ..., t_{k+N} \tag{4.7f}$$

where $\boldsymbol{x}_k$ the robot state at time t_k, $\boldsymbol{\phi}(\cdot,\cdot)$ the discrete-time dynamics of the robot obtained via numerical integration of (4.1) under the assumption of piecewise-constant control inputs, while $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$ and $\boldsymbol{u}_{\min}$, $\boldsymbol{u}_{\max}$ are the lower/upper bounds on the state variables and control inputs. The running cost $V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i})$ and the terminal cost $V_{k|N}(\boldsymbol{x}_{k|N})$ are expressed respectively as

$$V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{e}_{k|i}^T \boldsymbol{Q} \boldsymbol{e}_{k|i} + \dot{\boldsymbol{y}}_{k|i}^T \boldsymbol{S} \dot{\boldsymbol{y}}_{k|i} + \boldsymbol{u}_{k|i}^T \boldsymbol{R}\, \boldsymbol{u}_{k|i},$$

$$V_{k|N}(\boldsymbol{x}_{k|N}) = \boldsymbol{e}_{k|N}^T \boldsymbol{Q}_N \boldsymbol{e}_{k|N} + \dot{\boldsymbol{y}}_{k|N}^T \boldsymbol{S}_N \dot{\boldsymbol{y}}_{k|N},$$

with $\boldsymbol{Q}$, $\boldsymbol{S}$ and $\boldsymbol{R}$ weighting matrices of appropriate dimensions for the predicted task error, the velocity of C and the control effort throughout the prediction horizon, and $\boldsymbol{Q}_N$ and $\boldsymbol{S}_N$ weighting matrices for the predicted task error and the velocity of C at the final time instant.

As regards the collision avoidance constraints (4.7f), they will be presented in full detail in the following.

4.4.2 CBF-based collision avoidance

To guarantee collision avoidance, the proposed NMPC incorporates constraints formulated via discrete-time CBFs.

Consider the smallest circle bounding the planar projection of the robot occupancy volume $\mathcal{R}(\boldsymbol{q})$ at any configuration $\boldsymbol{q}$ (see Fig. 3.1). Let ρ be the radius of this circle and R its center, whose position vector is denoted by $\boldsymbol{r}$ and depends on the robot configuration $\boldsymbol{q}$, and consequently to its state $\boldsymbol{x}$, via a forward kinematic map $\boldsymbol{r} = \boldsymbol{\sigma}(\boldsymbol{q})$.

We say that a robot state $\boldsymbol{x}$ is safe in terms of collision with a generic human, if the robot being at this state maintains a safety clearance $d_s > 0$ with the closest point of this human, i.e., $\|\boldsymbol{r} - \boldsymbol{p}\| \geq \rho + d_s$, where $\boldsymbol{p}$ the position of the closest to the robot point of the generic human. Accordingly, we define the safe-set of robot states with respect to the generic human as

$$\mathcal{C} = \{\boldsymbol{x} \in \mathcal{D} : h(\boldsymbol{x}) \geq 0\}$$

where

$$h(\boldsymbol{x}) = \|\boldsymbol{r} - \boldsymbol{p}\|^2 - (\rho + d_s)^2.$$

It can be shown [69] that the function $h(\boldsymbol{x})$ can serve as a CBF for the discrete-time system (4.7c), while the condition

$$\Delta h(\boldsymbol{x}_k, \boldsymbol{u}_k) \geq -\gamma h(\boldsymbol{x}_k) \tag{4.8}$$

with $\Delta h(\boldsymbol{x}_k, \boldsymbol{u}_k) = h(\boldsymbol{x}_{k+1}) - h(\boldsymbol{x}_k)$ and $0 < \gamma \leq 1$, attains forward-invariance of the safe-set $\mathcal{C}$, enforcing collision avoidance.

By imposing condition (4.8) throughout the prediction horizon for all M humans accounted in $\mathcal{P}_k$, the CBF-based collision avoidance constraints in (4.7f) can be explicitly written as

$$\begin{aligned} \Delta h^j(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) \geq -\gamma h^j(\boldsymbol{x}_{k|i}), \quad & i = 0, \ldots, N-1, \\ & j = 1, \ldots, M, \end{aligned} \tag{4.9}$$

where

$$h^j(\boldsymbol{x}_{k|i}) = \|\boldsymbol{r}_{k|i} - \boldsymbol{p}^j_{k|i}\|^2 - (\rho + d_s)^2,$$

with $\boldsymbol{r}_{k|i} = \boldsymbol{\sigma}(\boldsymbol{q}_{k|i})$ and $\boldsymbol{p}^j_{k|i}$ extracted from element $\boldsymbol{P}^j_k$ of $\mathcal{P}_k$.

Note that the identity of the closest point of a certain human might change along the prediction horizon. Such aspect is not explicitly considered by our collision avoidance constraints which, instead, involve only information about the predicted future position of the point that is the closest at time t_k. However, as we will show via simulation results in Sect. 4.5, this is not an issue for the proposed scheme thanks to (i) the real-time capabilities of our implementation that allows high control frequency and (ii) the use of a relatively small value for γ that improves the ability to promptly react to sudden changes in the robot surroundings.

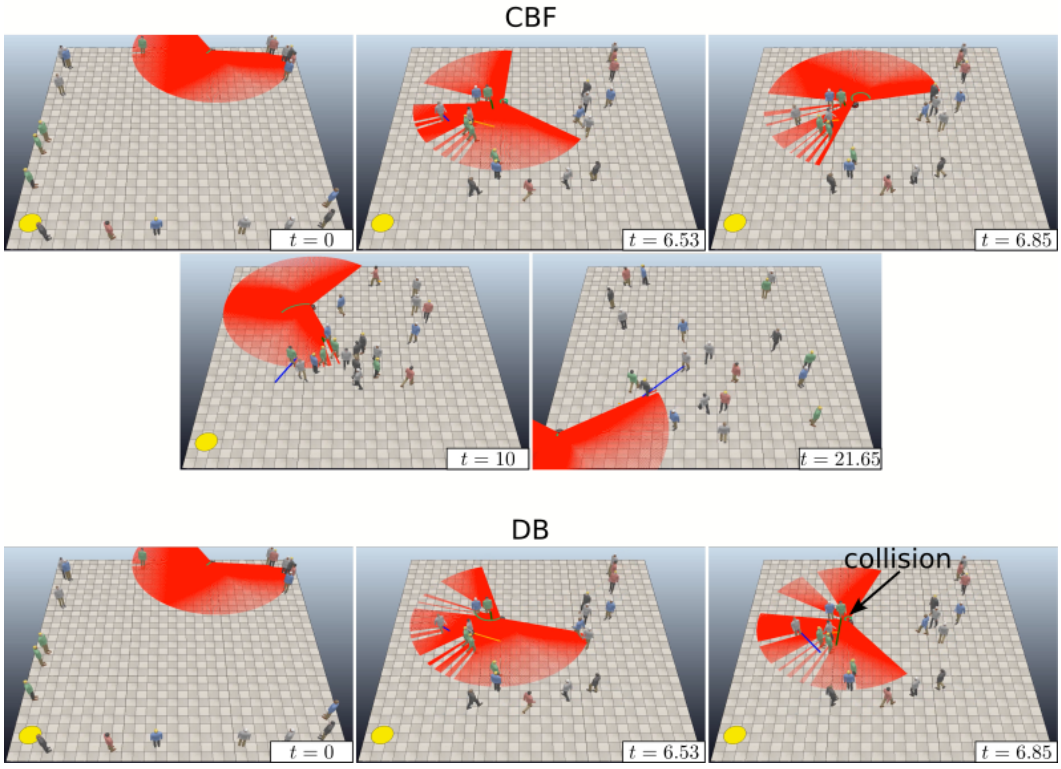

Figure 4.5. Snapshots of the motion generated by the proposed method (top) and the compared method (bottom) in a robot-unfriendly crowd of 20 humans using the *K-Neighbors* strategy. At time $t = 6.85\,\mathrm{s}$ the DB method runs into collision with one of the humans while the proposed method navigates safely towards the goal.

4.5 Simulations

The proposed safe navigation method was implemented in the CoppeliaSim environment as a C++ plugin. The latter wraps a library implementing the NMPC algorithm generated using the Python interface of `acados` [77]. For the NMPC the RTI scheme was used.

The targeted platform is the Pioneer 3-DX mobile robot for with $r = 9.75$ cm and $b = 38.1$ cm. It is equipped with an Hokuyo URG-04LX laser rangefinder whose detection area is aligned with the robot heading direction and has a range of 5 m and an angle of 240°, thus leaving a 120° blind zone behind the robot. According to the robot characteristics, its driving and steering velocities are bounded as, respectively, $0 \leq v \leq 1.2$ m[1] and $|\omega| \leq 5.24$ rad/s, while for the angular accelerations of the wheels $|\ddot{\phi}_r|, |\ddot{\phi}_l| \leq 70$ rad/s^2.

To assess the performance of the proposed framework, we conducted a series of simulations in an environment populated by a crowd of humans. To simulate the motion of the crowd we assumed that each human moves along a sequence of randomly generated viapoints (locations of the environment). The human moves in a unicycle-like fashion [52] and under the action of an artificial potential field, such

[1]We chose to not use a negative value for the minimum driving velocity in order to prevent the robot from moving back in its blind zone.

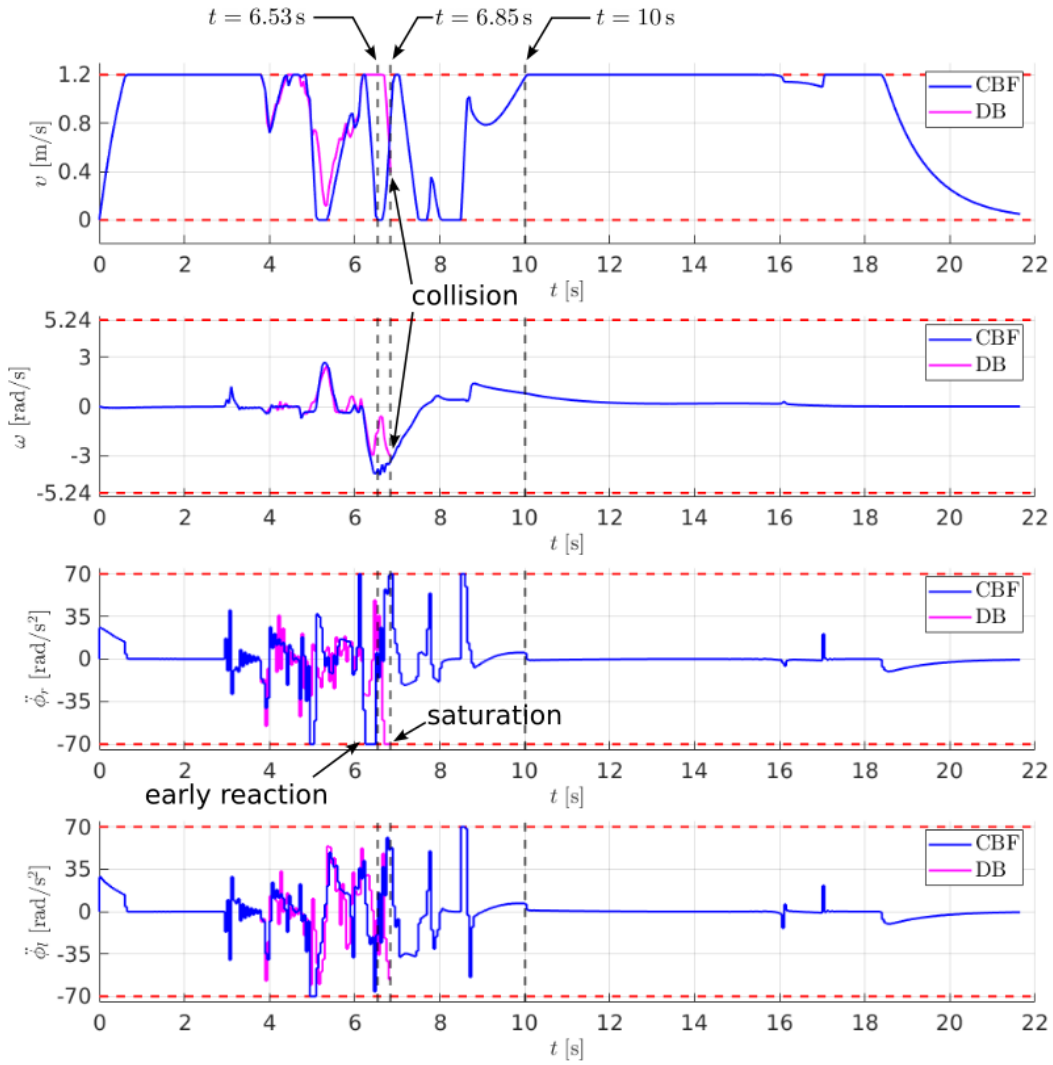

Figure 4.6. The driving and steering velocity profiles and the control inputs (driving wheels acceleration) generated by the two methods in the robot-unfriendly crowd of 20 humans using the *K-Neighbors* strategy.

that an attractive field drives him towards his currently targeted viapoint, while repulsive fields push him away from the other humans. Every time a human reaches the assigned viapoint, remains still for a certain pause period before moving to the next viapoint. Each human is assigned with a maximum linear velocity that cannot exceed during his motion.

The conducted simulations are divided into two campaigns. In the first one, the crowd is *robot-friendly* meaning that the humans are aware of the presence of the robot (and try to avoid collisions with it). To achieve this behavior, an additional repulsive field is added to each human, pushing him away from the robot. In the second campaign, a *robot-unfriendly* crowd is considered, which means that the humans are not aware of the robot presence, so it is up to the robot only to avoid collisions with them.

It is emphasized that the robot is not aware of such crowd behavior, but only relies on the information obtained by the on-board sensor.

Apart from the overall evaluation of the performance of the proposed framework, our intention is to investigate how this performance is affected by the particular choice of (*i*) the selection strategy in the crowd prediction module (i.e., *K-Neighbors* and *K-Cones*) and (*ii*) the collision avoidance constraint in the motion generation module. In order to investigate the effect of the collision avoidance constraint, the proposed CBF-based method will be compared with a purely distance-based (DB) approach that is obtained by using

$$h^j(\boldsymbol{x}_{k|i}) \geq 0, \quad i = 0, \ldots, N, \quad j = 1, \ldots, M$$

in (4.7f). To this purpose, for each campaign, we considered 12 simulation scenarios obtained by varying (*i*) the number (5, 10 or 20) of humans in a 15×15 m environment, (*ii*) the adopted selection strategy (*K-Neighbors* or *K-Cones*), and (*iii*) the formulation of the collision avoidance constraints (CBF or DB). For each of these scenarios, we run 50 different simulations, each time randomly generating the initial configuration of the robot, goal region $\mathcal{G}$, sequence of viapoints, associated pause periods and maximum linear velocity of each human.

All the simulations were performed on an Intel Core i7-8700K CPU running at 3.7 GHz. In all of them, the whole safe navigation scheme worked with a sampling time $\delta = 0.05$ s, while the prediction horizon was set to $T = 2$ s; also, we used $K = 3$, $d_C = 0.15$ m, $\rho_H = 0.8$ m and $\gamma = 0.3$.

The results of the two simulation campaigns are reported in Tables 4.1–4.2, respectively, where we show the success rate (we recorded a failure whenever a collision occurred) and maximum computation time (including both crowd prediction and motion generation modules) averaged over the 50 runs in each simulation scenario. Video clips showing examples of generated motions are available at the link `https://youtu.be/iDdM6Ud9I4c`.

In the case of robot-friendly crowd (see Table 4.1), as expected, for both our and DB method the success rate decreases as the number of humans populating the environment increases. However, our method maintains a higher success rate even in the most challenging environment (20 humans), where we achieve a success rate higher than 85%, independently of the adopted selection strategy, while the DB method fails in more than half of cases when using the *K-Cones* strategy. Moreover, the table confirms the real-time capabilities of the proposed scheme, as the maximum computational time is always lower than the sampling time. We also report that, in our observations, most of the computational time is spent by the motion generation module, while only a negligible portion is used by the crowd prediction module, which also highlights the efficiency of the latter.

The case of robot-unfriendly crowd (see Table 4.2) is clearly more challenging. In environments containing 10 and 20 humans, the success rate is significantly reduced compared to the case of robot-friendly crowd. This is reasonable if we consider that in this case a human can either approach the robot from its blind zone and collide with it, without any possibility for the robot to react, or suddenly turn towards it without giving the robot the chance to react in time. However, the table reveals that, even in a robot-unfriendly crowd, our method is clearly more effective than the

# of humans	selection strategy	collision avoidance constraint	success rate (%)	maximum computation time (ms)
5	*K-Neighbors*	CBF	100	31
		DB	92	28
	K-Cones	CBF	98	34
		DB	90	29
10	*K-Neighbors*	CBF	96	30
		DB	86	33
	K-Cones	CBF	98	29
		DB	72	34
20	*K-Neighbors*	CBF	88	36
		DB	64	36
	K-Cones	CBF	86	35
		DB	48	41

Table 4.1. Performance data in the case of robot-friendly crowd

# of humans	selection strategy	collision avoidance constraint	success rate (%)	maximum computation time (ms)
5	*K-Neighbors*	CBF	92	30
		DB	90	31
	K-Cones	CBF	92	32
		DB	84	27
10	*K-Neighbors*	CBF	74	28
		DB	62	28
	K-Cones	CBF	80	27
		DB	68	31
20	*K-Neighbors*	CBF	60	32
		DB	38	30
	K-Cones	CBF	58	29
		DB	40	31

Table 4.2. Performance data in the case of robot-unfriendly crowd

DB approach. Also in this case the real-time requirement is respected, as proven by the reported maximum computation times.

From the simulation results we can see a clear advantage of the proposed method (CBF) against the compared method (DB). To illustrate the different behavior of the two methods, in Fig. 4.5 we provide snapshots of the robot motion under the two methods in a robot-unfriendly crowd of 20 humans using the *K-Neighbors* strategy, while in Fig. 4.6 we illustrate the driving and steering velocity profiles and the control inputs throughout the robot motion. The driving velocity profiles reveal a more conservative motion generated by the proposed method compared to the one generated by the DB method. One can notice that while the DB method

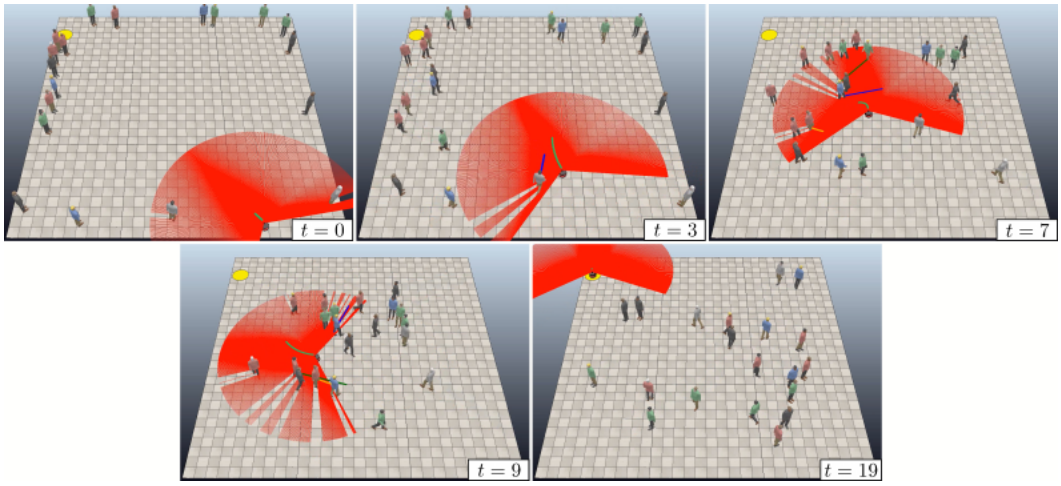

Figure 4.7. Safe navigation in a robot-unfriendly crowd of 20 humans using the *K-Neighbors* strategy. Snapshots from a simulation.

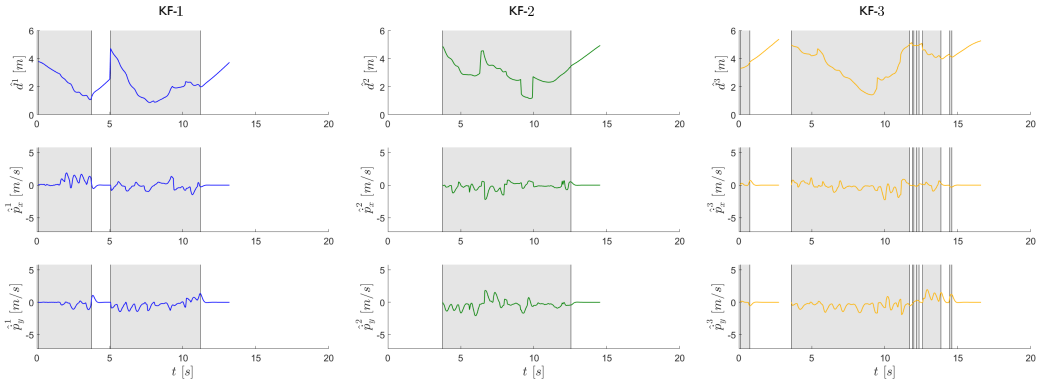

Figure 4.8. Safe navigation in a robot-unfriendly crowd of 20 humans using the *K-Neighbors* strategy. Plots of $\hat{d}^l$, $\hat{p}^l_x$, $\hat{p}^l_y$ $(l = 1, 2, 3)$, i.e., distance and x,y-components of the velocity of a human estimated by KF-l. Gray zones indicate time intervals in which the corresponding KF was in the *Active* state.

moves at high speed at time $t = 6.53\,\mathrm{s}$, the proposed method, reacting earlier to the presence of the closest human, has reduced the driving velocity to zero performing simultaneously a fast reorientation. With this collision avoidance maneuver the imminent collision with the human is effectively avoided and the robot manages to reach the goal safely. On the contrary, due to the late reaction of the DB method to the presence of the human, it was impossible for the robot to perform successfully the required collision avoidance maneuver and it collided at time $t = 6.85\,\mathrm{s}$. Note that although the robot tried to perform a reorientation to the right (see the acceleration at the right wheel being saturated before the collision in Fig. 4.6), due to its late reaction, the collision was inevitable.

Regarding the different selection strategies, Figures 4.7 and 4.9 show snapshots from two simulations obtained using the proposed method with the *K-Neighbors* and *K-Cones* strategy, respectively, in a robot-unfriendly crowd of 20 humans. Figures 4.8 and 4.10 show associated plots. Clips are included in the video. Comparing the

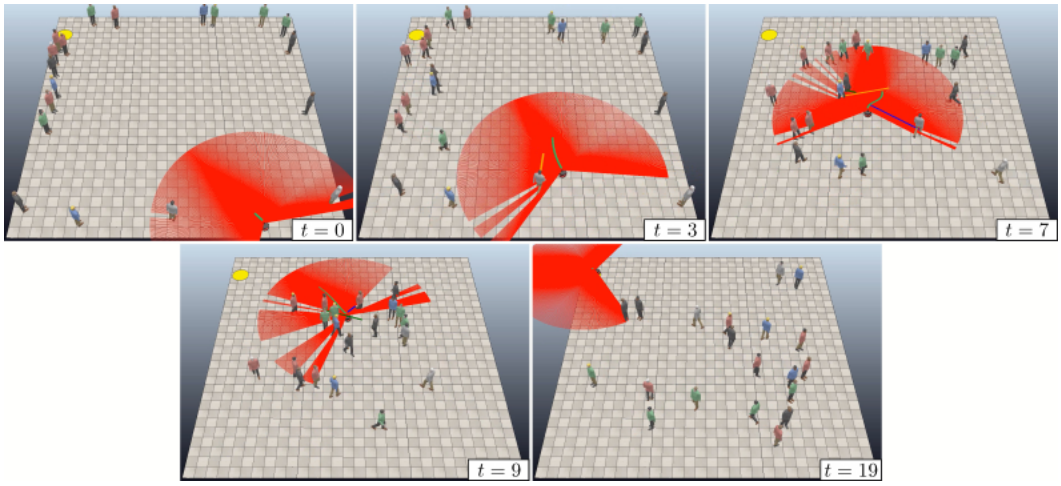

Figure 4.9. Safe navigation in a robot-unfriendly crowd of 20 humans using the *K-Cones* strategy. Snapshots from a simulation.

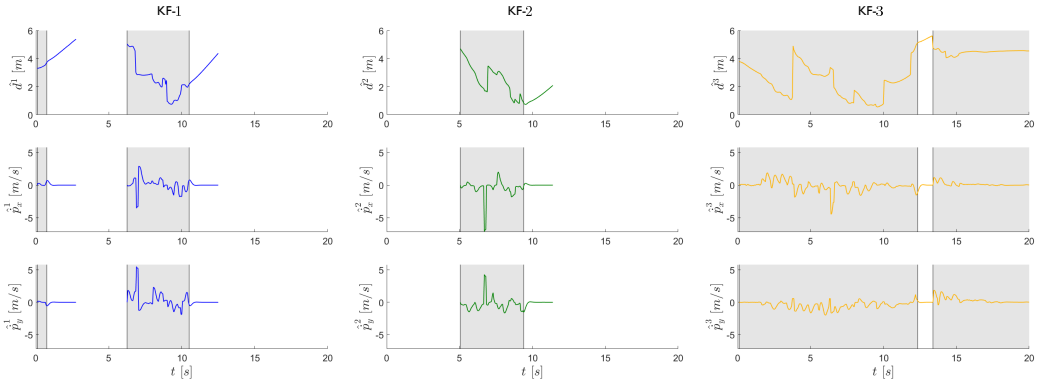

Figure 4.10. Safe navigation in a robot-unfriendly crowd of 20 humans using the *K-Cones* strategy. Plots of $\hat{d}^l$, $\hat{p}^l_x$, $\hat{p}^l_y$ ($l = 1, 2, 3$), i.e., distance and x,y-components of the velocity of a human estimated by KF-l. Gray zones indicate time intervals in which the corresponding KF was in the *Active* state.

snapshots, taken at the same time instants, it is possible to appreciate that although the crowd is identically placed at each considered time instant, the robot follows different paths due to the influence of the chosen selection strategy.

Although in terms of both success rate and computation time we found no significant differences between the two selection strategies, in practice the *K-Cones* strategy (as can be seen from the plots corresponding to KF-1 and KF-2 in Fig. 4.10) might consider for collision avoidance a number of humans smaller than K even if K humans are actually present in the sensor detection area, which could obviously be disadvantageous. On the other hand, the *K-Cones* strategy is a valid option when dealing with multiple sensors that have a reduced detection angle. In this case, each cone will correspond to a certain sensor and the presented scheme can be applied without any significant modification.

Note that, even if we presented simulation results obtained using $K = 3$, our approach can work with generic values of K provided that this still guarantees

real-time performance. The video also shows simulations obtained for $K = 6$.

4.6 Conclusions

In this chapter, we presented a sensor-based scheme for safe robot navigation in a crowd of moving humans. At each control cycle, the crowd prediction module foresees the future motion of the humans in the robot surroundings using the information obtained by the robot on-board sensors. Then, the motion generation module produces feasible commands for the robot to safely drive it among the humans until a desired goal region of the workspace is reached. For crowd prediction we employ a simple, yet effective, technique based on Kalman filters, while our motion generation strategy combines NMPC and collision avoidance constraints formulated via discrete-time CBFs.

Simulation results show that this combination can produce sensible results in environments crowded by a varying number of moving humans. Moreover, we demonstrated that our approach considering collision avoidance constraints built upon appropriate CBFs outperforms the typical schemes that consider collision avoidance constraints based on purely distance information.

Future work will address

- the investigation of feasibility properties of the NMPC and design of recovery strategies in case of possible infeasibilities;
- the extension of our method to environments containing obstacles of various shapes;
- an in-depth performance analysis of the effect of the various parameters (e.g., K, T, γ).

Chapter 5

A dynamics-aware NMPC method for robot navigation

In many navigation problems, a relatively heavy robot (with respect to its actuation capabilities) may be called to move at high-speed in environments populated by static and/or moving obstacles (e.g., load transportation robot in a warehouse). In a motion like this, the effect of the robot dynamics becomes significant. As a result, in order to generate the desired motion via NMPC, one has to consider the robot full dynamic model as well as small sampling times. Clearly, real-time performance does not allow arbitrarily long prediction horizons, while on typical robot processing platforms, this horizon ends up being relatively short. In cases like this, the use of a purely distance-based collision avoidance constraint may jeopardize the robot safety, since the danger of collision may be detected at a time when the robot does not have the necessary actuation capabilities to prevent it.

In Chap. 4, we showed that a CBF-based collision avoidance constraint that practically considers in addition to the robot-obstacle distance, the rate of change of this distance can significantly improve the performance of the NMPC compared to a purely distance-based collision avoidance constraint. Nevertheless, in that work, only the robot kinematics were considered.

In this chapter, we consider a scenario in which a relatively heavy robot has to navigate at high speed in an environment occupied by static and moving obstacles. To solve this problem we use the motion generation approach which was originally proposed in our work [2]. Specifically, taking inspiration from the ICS concept, we define the notion of *Avoidable Collision State* (ACS), i.e., a state from which the robot can avoid collision with a certain obstacle. Building on this, we propose an NMPC method for real-time motion generation which enforces collision avoidance through a constraint that essentially requires the robot to be in an ACS at all times. Extensive simulations on a differential-drive robot that navigates in both static and dynamic environments clearly show the superiority of the proposed NMPC method with respect to distance-based formulations, especially when the robot must navigate at high-speed in environments cluttered with moving obstacles. In particular, it is shown that thanks to its dynamics-aware nature, the novel constraint works well with relatively short prediction horizons, making real-time performance achievable even on low-cost computational platforms.

The chapter is organized as follows. The navigation problem is formulated in Sect. 5.1. In Sect. 5.2 we outline the proposed NMPC approach, while in Sect. 5.3 we formally define the concept of ACS and derive the associated collision avoidance constraint. Simulation results for a differential-drive robot are presented in Sect. 5.4. Finally, some concluding remarks are offered in Sect. 5.5.

5.1 Problem formulation

Consider again the differential-drive mobile robot of Fig. 3.1 with configuration $\boldsymbol{q}$ whose motion is described by the state-space reduced model (3.19), rewritten here as

$$\dot{\boldsymbol{x}} = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}) = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{u})) \end{pmatrix}, \tag{5.1}$$

with the robot state defined as $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{\nu})$, with the pseudovelocities vector $\boldsymbol{\nu}$ comprised by the robot driving and steering velocity, i.e., $\boldsymbol{\nu} = (\upsilon, \omega)$, and the control inputs being at the wheel torque level, i.e., $\boldsymbol{u} = \boldsymbol{\tau} = (\tau_r, \tau_l)$.

The robot moves in a workspace $\mathcal{W}$, populated by static and/or moving obstacles. We shall denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$, and by $\mathcal{O}(t) \subset \mathcal{W}$ the volume occupied by the environment obstacles at time t.

A navigation task is assigned to the robot in terms of a vector $\boldsymbol{y} \in \mathcal{Y}$, which describes the position of a representative point on the robot, here the point C, and is related to the configuration via a forward kinematic map $\boldsymbol{y} = \boldsymbol{k}(\boldsymbol{q})$. The task is to drive C in the goal region $\mathcal{G}$.

The problem consists in generating in real-time a motion that:

1. drives the robot from its starting configuration to a configuration realizing the task, i.e., the representative point C lies inside the goal region $\mathcal{G}$;

2. is kinodynamically feasible, in the sense that it is consistent with model (5.1) and respects existing constraints on both $\boldsymbol{x}$ (i.e., velocity limits) and $\boldsymbol{u}$ (i.e., torque bounds);

3. avoids collisions between the robot and the obstacles, i.e., $\mathcal{R}(\boldsymbol{q}) \cap \mathcal{O}(t) = \emptyset$, at all time instants t.

As for the information available for solving the problem, it is assumed that the robot is always aware of its own state and that an on-board sensor provides the position and velocity of all obstacles located inside the sensor field-of-view.

5.2 Proposed NMPC approach

To generate the desired motion we will use an NMPC algorithm for real-time motion generation. Consider the NLP to be solved at time t_k. Our objective is to drive the representative point C of the robot inside the goal region, ideally with the minimum control effort, maintaining also the robot and environment safety. So, we can use the NLP proposed in (3.21), which is restated here for completeness.

Denote by T the prediction horizon, by δ the sampling time and by $N = T/\delta$ the number of control intervals in the prediction horizon. Let $\boldsymbol{x}_{k|i}$ and $\boldsymbol{u}_{k|i}$ be the predicted robot state and control inputs at time t_{k+i}, $\boldsymbol{y}_{k|i}$ and $\dot{\boldsymbol{y}}_{k|i}$ be respectively the predicted position and velocity of C at t_{k+i}, $\boldsymbol{y}_d$ be the position of a representative point in $\mathcal{G}$ and $\boldsymbol{e}_{k|i} = \boldsymbol{y}_d - \boldsymbol{y}_{k|i}$ be the predicted task error at time t_{k+i}. The NLP to be solved at time t_k is

$$\min_{\substack{\boldsymbol{x}_{k|0},\dots,\boldsymbol{x}_{k|N},\\ \boldsymbol{u}_{k|0},\dots,\boldsymbol{u}_{k|N-1}}} \sum_{i=0}^{N-1} V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) + V_{k|N}(\boldsymbol{x}_{k|N}) \tag{5.2a}$$

subject to:

$$\boldsymbol{x}_{k|0} - \boldsymbol{x}_k = \boldsymbol{0} \tag{5.2b}$$

$$\boldsymbol{x}_{k|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0, ..., N-1 \tag{5.2c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_{k|i} \leq \boldsymbol{x}_{\max}, \quad i = 0, ..., N \tag{5.2d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_{k|i} \leq \boldsymbol{u}_{\max}, \quad i = 0, ..., N-1 \tag{5.2e}$$

$$\textit{collision avoidance constraints at } t_k, ..., t_{k+N} \tag{5.2f}$$

where $\boldsymbol{x}_k$ the robot state at time t_k, $\boldsymbol{\phi}_{\text{d-t}}(\cdot,\cdot)$ the discrete-time system dynamics obtained via numerical integration of (5.1) under the assumption of piecewise-constant control inputs, while $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$ and $\boldsymbol{u}_{\min}$, $\boldsymbol{u}_{\max}$ are the lower/upper bounds on the state variables and control inputs. The running cost $V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i})$ and the terminal cost $V_{k|N}(\boldsymbol{x}_{k|N})$ are expressed, respectively, as

$$V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{e}_{k|i}^T \boldsymbol{Q} \boldsymbol{e}_{k|i} + \dot{\boldsymbol{y}}_{k|i}^T \boldsymbol{S} \dot{\boldsymbol{y}}_{k|i} + \boldsymbol{u}_{k|i}^T \boldsymbol{R}\, \boldsymbol{u}_{k|i},$$

$$V_{k|N}(\boldsymbol{x}_{k|N}) = \boldsymbol{e}_{k|N}^T \boldsymbol{Q}_N \boldsymbol{e}_{k|N} + \dot{\boldsymbol{y}}_{k|N}^T \boldsymbol{S}_N \dot{\boldsymbol{y}}_{k|N},$$

with $\boldsymbol{Q}$, $\boldsymbol{S}$ and $\boldsymbol{R}$ weighting matrices of appropriate dimensions for the predicted task error, the velocity of C and the control effort throughout the prediction horizon, and $\boldsymbol{Q}_N$ and $\boldsymbol{S}_N$ weighting matrices for the predicted task error and the velocity of C at the final time instant.

As for the collision avoidance constraint (5.2f), which is the main contribution of this work, it is discussed in full detail in the next section.

5.3 Dynamics-aware collision avoidance

The proposed collision avoidance constraint hinges upon the notion of Avoidable Collision State (ACS), i.e., a robot state from which it is possible to avoid collisions. In this section, we will first give a formal definition of what an ACS is, and then derive the corresponding collision avoidance constraint.

5.3.1 Preliminaries

The notion of ACS is obstacle-specific, in the sense that it characterizes the possibility for the robot to avoid a certain obstacle. In view of this, we first need to identify the obstacles for which there is an actual danger of collision given the current state of the robot.

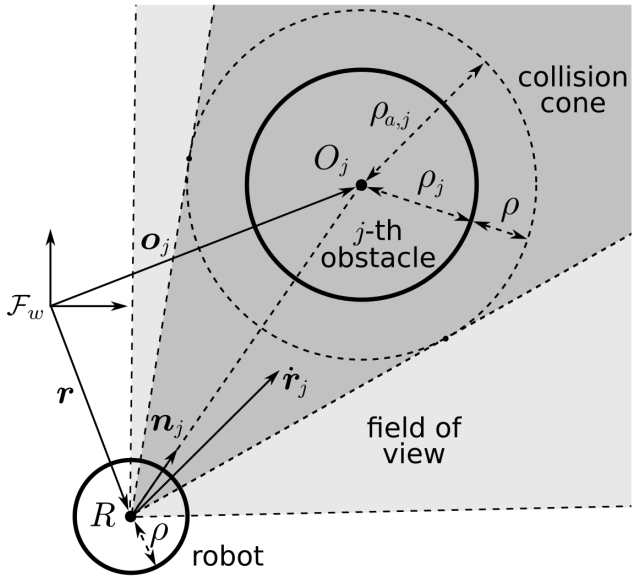

Figure 5.1. Obstacle $\mathcal{O}_j$ is considered dangerous for the robot at a certain time if the relative velocity $\dot{\boldsymbol{r}}_j$ of the robot with respect to the obstacle lies inside the collision cone.

We are going to use bounding circles to envelope the planar projection of the occupancy volume of both the robot and the obstacles. In particular, we take the smallest circle that contains the planar projection of the robot volume $\mathcal{R}$, and denote its radius by ρ and its center by R. The position vector of R in the world frame is denoted by $\boldsymbol{r}$ and is related to the robot configuration via a forward kinematic map $\boldsymbol{r} = \boldsymbol{\sigma}(\boldsymbol{q})$. For its velocity we have: $\dot{\boldsymbol{r}} = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{\nu}$ where $\boldsymbol{J}(\boldsymbol{q}) = \partial\boldsymbol{\sigma}(\boldsymbol{q})/\partial\boldsymbol{q}\boldsymbol{G}(\boldsymbol{q})$. Similarly, we use a bounding circle of radius ρ_j to envelop the planar projection of the volume of the generic obstacle $\mathcal{O}_j(t) \subset \mathcal{O}(t)$, denoting its center by O_j, the corresponding position vector by $\boldsymbol{o}_j$ and its velocity by $\dot{\boldsymbol{o}}_j$. Finally, let $\boldsymbol{n}_j$ be the unit vector pointing from R to O_j, and $\boldsymbol{r}_j = \boldsymbol{r} - \boldsymbol{o}_j$ be the relative position of R with respect to O_j, so that the corresponding relative velocity is $\dot{\boldsymbol{r}}_j$. Refer to Fig. 5.1 for illustration.

Now, augment the obstacle circle by the radius of the robot circle, denoting by $\rho_{a,j} = \rho_j + \rho$ the total radius. The *collision cone* is the cone defined by R and the tangents from R to the augmented obstacle. Obstacle $\mathcal{O}_j$ is *dangerous* at time t if $\dot{\boldsymbol{r}}_j(t)$ lies inside the collision cone (see Fig. 5.1). Simple geometrical arguments lead to the following condition for an obstacle to be dangerous:

$$h(\boldsymbol{x}, \boldsymbol{\xi}_j) = \boldsymbol{n}_j^T \frac{\dot{\boldsymbol{r}}_j}{\|\dot{\boldsymbol{r}}_j\|} - \frac{\sqrt{\|\boldsymbol{r}_j\|^2 - \rho_{a,j}^2}}{\|\boldsymbol{r}_j\|} \geq 0 \tag{5.3}$$

where $\boldsymbol{\xi}_j = (\boldsymbol{o}_j, \dot{\boldsymbol{o}}_j)$ is the state of the j-th obstacle.

5.3.2 Avoidable collision states

By definition, to avoid an obstacle which is non-dangerous at time t the robot simply needs to keep its course. Therefore, we only need to characterize the possibility of avoiding obstacles that are dangerous at t. In particular, we will say that the

robot is in an *Avoidable Collision State* (ACS) with respect to a dangerous obstacle if there exists at least one trajectory that originates from the current state, is kinodynamically feasible and avoids collision with the obstacle.

In principle, to conclude that a state is an ACS we should check all feasible trajectories emanating from it until we find at least one that avoids collision. However, such a potentially exhaustive study is incompatible with a real-time application. We shall therefore look at one specific motion and ask ourselves if the robot can avoid collision during that motion. If the answer is positive, then it can be concluded that the current state is certainly an ACS.

In particular, in the presence of an obstacle $\mathcal{O}_j$ which moves along a given direction with constant speed and is dangerous at time t, we consider the robot moving in such a way that at each time instant $t' \geq t$ its relative velocity with respect to the obstacle, $\dot{\boldsymbol{r}}_j(t')$, projected on the original direction of collision, $\boldsymbol{n}_j(t)$, decreases with a constant rate α.

During this motion there will be eventually a time instant, denoted by t_v, in which the projection of the robot relative velocity on the original direction of collision will be zero, i.e., $\boldsymbol{n}_j^T(t)\dot{\boldsymbol{r}}_j(t_v) = 0$. A sufficient condition for this motion to be collision-free is:

$$\boldsymbol{n}_j^T(t)(\boldsymbol{o}_j(t') - \boldsymbol{r}(t')) \geq \rho_{a,j}, \quad \forall t' \in [t, t_v].$$

Denoting by $\gamma(t)$ the robot-obstacle clearance and considering that at time t the relation $\boldsymbol{n}_j^T(t)(\boldsymbol{o}_j(t) - \boldsymbol{r}(t)) = \gamma(t) + \rho_{a,j}$ holds, after simple substitution of $\rho_{a,j}$, the condition for collision-free motion becomes:

$$\boldsymbol{n}_j^T(t)(\boldsymbol{r}_j(t') - \boldsymbol{r}_j(t)) \leq \gamma(t), \quad \forall t' \in [t, t_v]. \tag{5.4}$$

Note that inequality (5.4) defines an admissible half-plane for the relative position of R with respect to O_j in which the robot has to remain for all $t' \in [t, t_v]$ (see Fig. 5.2).

In order to ensure that the considered motion can be implemented by the robot we will investigate whether the required deceleration of R, i.e. $\ddot{\boldsymbol{r}} = \boldsymbol{n}_j(t)\alpha$, projected on the robot actuation space lies within the actuation limits.

Throughout the considered motion, the position of R, projected on the original direction of collision $\boldsymbol{n}_j(t)$ is described at time $t' \geq t$ as

$$\boldsymbol{n}_j^T(t)\boldsymbol{r}(t') = \boldsymbol{n}_j^T(t)\boldsymbol{r}(t) + \boldsymbol{n}_j^T(t)\dot{\boldsymbol{r}}(t)(t' - t) + \frac{1}{2}\alpha(t' - t)^2 \tag{5.5}$$

and its projected velocity as

$$\boldsymbol{n}_j^T(t)\dot{\boldsymbol{r}}(t') = \boldsymbol{n}_j^T(t)\dot{\boldsymbol{r}}(t) + \alpha(t' - t). \tag{5.6}$$

As for the obstacle, the position of O_j at $t' \geq t$ projected on the original direction of collision is

$$\boldsymbol{n}_j^T(t)\boldsymbol{o}_j(t') = \boldsymbol{n}_j^T(t)\boldsymbol{o}_j(t) + \boldsymbol{n}_j^T(t)\dot{\boldsymbol{o}}_j(t' - t). \tag{5.7}$$

By solving the system of equations (5.4) for the equality, (5.5), (5.6) and (5.7), with $t' = t_v$, we obtain the minimum required deceleration of the robot in the original direction of collision:

$$\alpha = -\frac{1}{2}\frac{\left(\boldsymbol{n}_j^T(\dot{\boldsymbol{o}}_j - \dot{\boldsymbol{r}}(t))\right)^2}{\gamma(t)}.$$

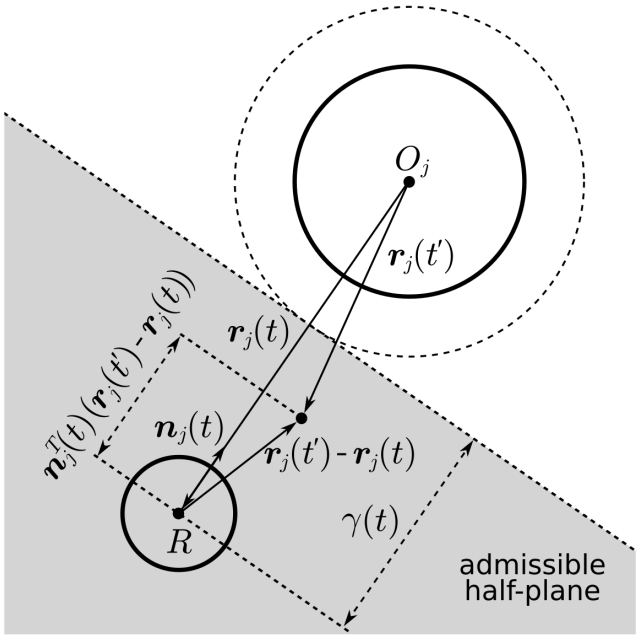

Figure 5.2. The admissible half-plane for the relative position of robot R with respect to the obstacle O_j at time t', $t' \in [t, t_v]$.

Note that α depends on both the robot and the j-th obstacle state. Considering that the acceleration of R is:

$$\ddot{\boldsymbol{r}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{\boldsymbol{J}}(\boldsymbol{q})\boldsymbol{\nu} \tag{5.8}$$

and that from (5.1) we can express $\dot{\boldsymbol{\nu}}$ as:

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu})), \tag{5.9}$$

we can obtain the required control inputs for this deceleration by substituting $\ddot{\boldsymbol{r}} = \boldsymbol{n}_j(t)\alpha$ and (5.9) in (5.8). Using the Moore-Penrose pseudoinverse, we get the minimum norm control inputs needed in order to apply the deceleration $\ddot{\boldsymbol{r}} = \boldsymbol{n}_j(t)\alpha$ to R:

$$\boldsymbol{u}_\alpha(\boldsymbol{x}, \boldsymbol{\xi}_j) = \Big(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q})\Big)^\dagger \boldsymbol{\beta}(\boldsymbol{x}, \boldsymbol{\xi}_j), \tag{5.10}$$

where

$$\boldsymbol{\beta}(\boldsymbol{x}, \boldsymbol{\xi}_j) = \boldsymbol{n}_j(t)\alpha - \dot{\boldsymbol{J}}(\boldsymbol{q})\boldsymbol{\nu} + \boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu}).$$

So the motion is kinodynamically feasible if the following condition is satisfied:

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_\alpha(\boldsymbol{x}, \boldsymbol{\xi}_j) \leq \boldsymbol{u}_{\max}. \tag{5.11}$$

This condition can be used as a constraint in order to guarantee that the robot is always at an ACS.

Note that the ACS property for a state is strongly related to its being *safe* according to [70], i.e., not being an Inevitable Collision State (ICS). However, the two properties differ in two aspects:

- The number of obstacles considered by the property. An ICS is defined with respect to all the obstacles (or at least all the obstacles visible by the robot), a practice that obviously increases the computational time, while the ACS property is defined with respect to a specific (dangerous) obstacle;
- The way in which the property is established. To prove that a state is not ICS, in principle, one has to search the whole control input set (or a finite subset [71]) to find a collision-free motion. On the other hand, to characterize a state as ACS we only look at the relative velocity of the robot with respect to the dangerous obstacle and specifically investigate whether it is possible to stop motion along the robot-obstacle direction before collision.

5.3.3 Use of the ACS condition in the NLP

Note that constraint (5.11) is suitable for enforcing collision avoidance since every robot motion that leads to collision will violate the constraint before the collision occurs. So we will use (5.11) in the proposed NLP applying it for each considered obstacle, for each time instant throughout the prediction horizon. In order to ensure that the constraint is inactive if non-dangerous obstacles are considered and since we cannot use *if* statements, we multiply $\boldsymbol{u}_\alpha(\boldsymbol{x}, \boldsymbol{\xi}_j)$ as given by (5.10) by the sigmoid function $g(h(\boldsymbol{x}, \boldsymbol{\xi}_j)) = 1/(1 + e^{-\kappa h(\boldsymbol{x}, \boldsymbol{\xi}_j)})$, with κ being a constant value that tunes the steepness of the sigmoid function, getting:

$$\boldsymbol{u}_b(\boldsymbol{x}, \boldsymbol{\xi}_j) = g(h(\boldsymbol{x}, \boldsymbol{\xi}_j)) \left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q})\right)^\dagger \boldsymbol{\beta}(\boldsymbol{x}, \boldsymbol{\xi}_j).$$

So the constraint that will be applied in the NLP is:

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_b(\boldsymbol{x}_{k|i}, \boldsymbol{\xi}_{j,k|i}) \leq \boldsymbol{u}_{\max}, \quad i = 0, \ldots, N, \quad j = 1, \ldots, n_o, \tag{5.12}$$

where $\boldsymbol{\xi}_{j,k|i}$ is the state of the j-th obstacle at t_{k+i} and n_o is the number of the considered obstacles in the NMPC.

5.4 Simulations

In order to show the effectiveness of the proposed method, we conducted a series of simulations in various static and dynamic environments. All the simulations were implemented in MATLAB on an Intel Core i9-9900K CPU at 3.60 GHz. For the solution of the NMPC, we used the RTI scheme implemented within the `ACADO Toolkit`.

As a robotic platform for our simulations, we used a differential-drive robot (see Fig. 3.1 for reference) with $l_1 = 60$ cm, $l_2 = b = 30$ cm and $r = 10$ cm. The Center of Mass (CoM) is located at the geometric center of the vehicle, R, located at a distance $d_R = 25$ cm from the driving wheels. We use the same point as the center of the robot bounding circle. The robot bounding circle radius is $\rho = \sqrt{l_1^2/4 + l_2^2/4}$. The vehicle mass and moment of inertia are respectively 50.0 kg and 1.41 kg·m^2, making this a rather heavy vehicle. The torque bounds for the wheel actuators are set to 2.5 Nm.

The robot representative point for the navigation task C is also placed at the geometric center of the robot.

We wish to compare the proposed NMPC method, which relies upon the dynamics-aware (DA) collision avoidance constraint, with an NMPC that uses a purely distance-based (DB) collision avoidance constraint. In both methods, the NLP formulation is the same, apart from the collision avoidance constraint (5.2f) which in DB takes the form:

$$\left\| \boldsymbol{r}_{k|i} - \boldsymbol{o}_{j,k|i} \right\| \geq \rho_{a,j}, \quad i = 0, \ldots, N, \quad j = 1, \ldots, n_o,$$

where $\boldsymbol{r}_{k|i}$ and $\boldsymbol{o}_{j,k|i}$ are the position of R and O_j at t_{k+i}, respectively. In both NMPCs we consider the $n_o = 5$ closest visible obstacles. The sampling time for real-time control is $\delta = 31$ ms in both cases.

The two methods have been tested in 50 different environments (25 static and 25 dynamic), using 3 different values for the maximum driving velocity $v_{\max}$. The maximum steering velocity is always set at $\omega_{\max} = 20/3\, v_{\max}$ rad/s. The prediction horizons were chosen as large as possible to allow real-time performance for the two methods. In particular, for the DB method the prediction horizon is $T = 0.992\,\mathrm{s}$ while for the DA method it is $T = 0.93\,\mathrm{s}$; in fact, the DB method requires slightly less computations than the DA method.

We assess the performance of the two methods by taking into account the success rate as well as the quality of the motion according to the following criteria: (1) time t_g needed for the robot to reach the goal, (2) control effort $J_\tau = \int_0^{t_g} \|\boldsymbol{\tau}\|^2 dt$, (3) length l_p of the resulting path, (4) duration of the longest iteration $\delta_{\max}$ and (5) average iteration time $\bar{\delta}$.

Video clips of representative simulation results are available at `https://youtu.be/LS57E8jGoJk`.

5.4.1 Static environments

In the first group of simulations, the robot moves in 25 environments, each occupied by 10 static obstacles (e.g., see Fig. 5.3). The robot starts from an initial position $(x_b, y_b) = (2, 2)$ at the Cartesian space with orientation $\theta_b = \pi/3$ rad and must reach $\boldsymbol{y}_g = (16, 15)$. Table 5.1 (top) summarizes the results of the DA and DB methods averaged over the 25 environments, for increasing values of the maximum driving velocity $v_{\max}$.

For lower $v_{\max}$ the two methods behave similarly having also the same success rate. An illustrative example of the robot motion with $v_{\max} = 0.9$ m/s is given in simulation 1 of the accompanying video. However, when $v_{\max} = 1.2$ m/s the success rate of the DB method drops to 88% while the DA method is hardly affected by the increase in the maximum velocity. Note that in this case, the stopping time[1] t_s is significantly larger than the prediction horizon. This shows that under these conditions, unlike DA, a robot navigating with the DB method would require either a longer prediction horizon or greater braking capabilities in order to navigate safely. A representative example of the behavior of the two methods is shown in Fig. 5.3 while in Fig. 5.4 we offer the resulting velocity and control input profiles throughout

[1]By stopping time t_s we denote the minimum time that a robot traveling on a straight line with speed $v_{\max}$ needs in order to stop.

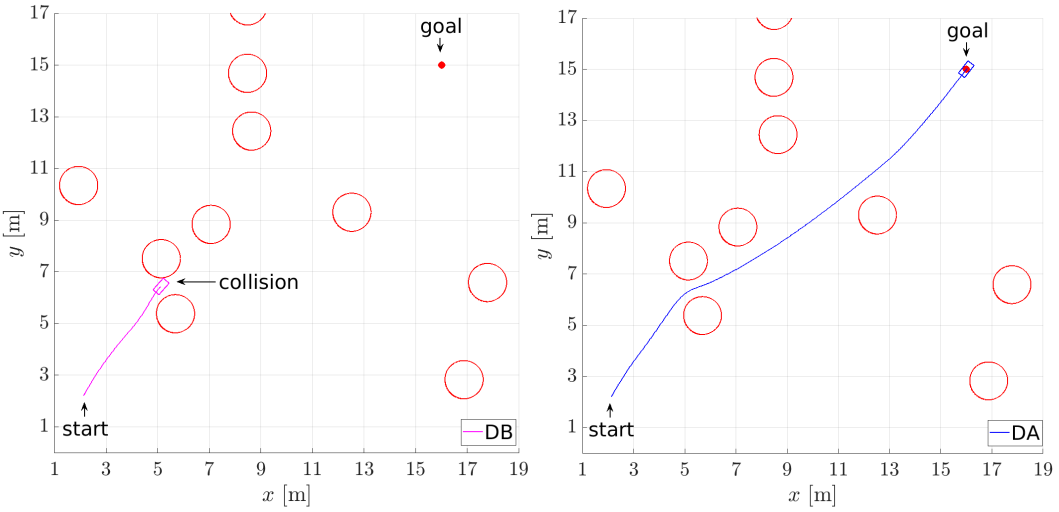

Figure 5.3. Motions generated by the two methods in one of the static environments, with $v_{\max} = 1.2$ m/s. The DB method (left) cannot avoid collision with an obstacle, whereas DA (right) goes safely through the narrow passage and successfully reaches the goal. See also simulation 2 in the accompanying video.

the robot motion (also see simulation 2 in the accompanying video). In particular, Fig. 5.3 shows the trajectories generated by the two methods in one of the static environments with $v_{\max} = 1.2$ m/s. The DB method (left) runs into a collision while passing through a narrow passage because the robot has accumulated too much speed in the initial part of the motion and avoiding the imminent collision would require more than the available torque (this corresponds to an unfeasibility for the NMPC). On the other hand, the DA method (right) starts the avoidance maneuver earlier thanks to the dynamics-aware constraint, which can detect a violation before the distance-based constraint does; therefore, DA is able to go through the narrow passage and reach the goal safely. This early reaction of the DA method to the presence of the obstacles is better illustrated in Fig. 5.4. Here, one can notice how the control inputs generated by the two methods start to differ after $t \approx 2$ s with DA starting diverging from the collision path almost 0.5 s earlier than the DB method. This late reaction from the DB method led later to torque saturation in the vicinity of the obstacle and eventually to collision.

Looking at the other performance criteria of Table 5.1, one can observe that on average the DA method produces trajectories that are slightly shorter and less energy-consuming than the DB method. As a counterpart, the DB method shows a slightly reduced duration of the longest iteration mainly due to the simplicity of the collision avoidance constraint.

5.4.2 Dynamic environments

The second group of simulations take place in 25 dynamic environments, each occupied by 10 static and 10 moving obstacles. The moving obstacles are zigzagging at a speed $v_o = v_{\max}/2$, while their direction changes after traveling a distance of 2.45 m by turning $\pi/3$ rad toward the robot. The results are presented in

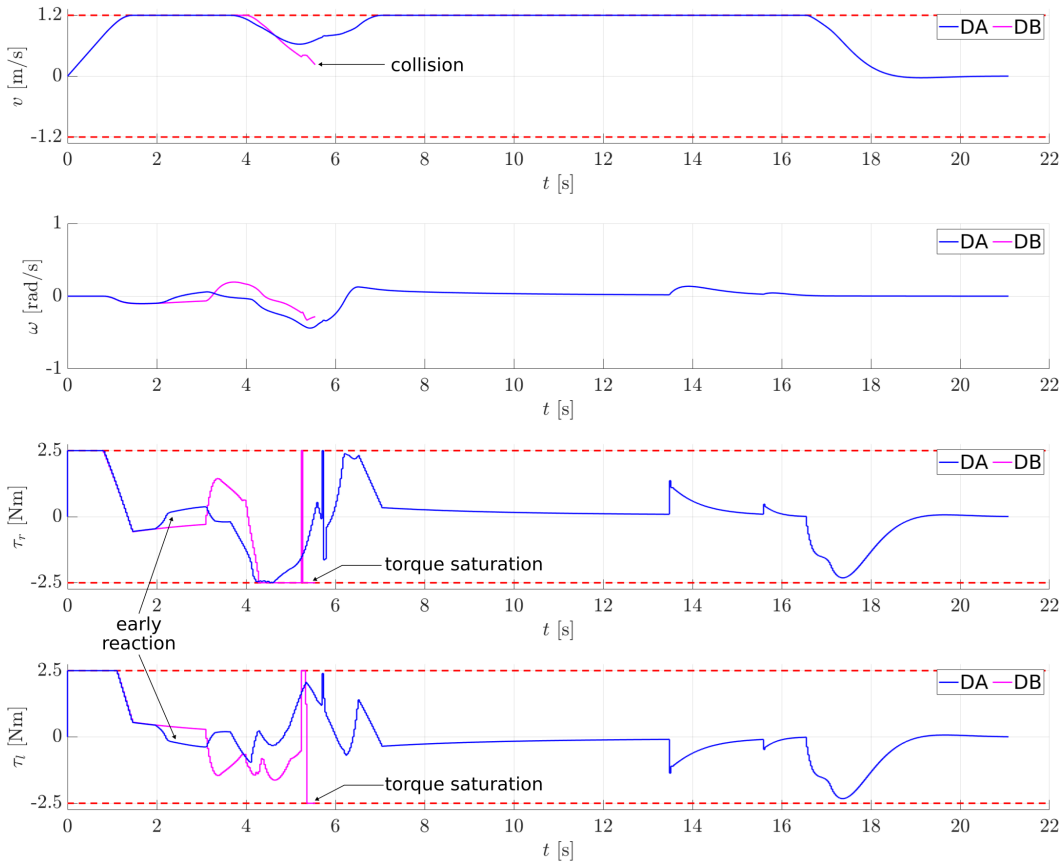

Figure 5.4. The driving and steering velocity profiles and the control inputs generated by the two methods in the static environment, with $v_{\max} = 1.2$ m/s. See also simulation 2 in the accompanying video.

Table 5.1 (bottom).

The success rate of both methods is affected by the presence of the moving obstacles. However, the DA method is much more effective than the DB method, with success rates over 80% while DB goes as low as 40%.

Once again, for the lowest $v_{\max}$ the DB method has its higher success rate, yet lower than the DA method. In simulation 3 of the accompanying video, we show the robot moving in a dynamic environment with $v_{\max} = 0.9$ m/s. Nevertheless, the success rate of the DB method decreases as $v_{\max}$ increases and plummets to 40% for $v_{\max} = 1.2$ m/s, revealing that the presence of moving obstacles in combination with fast robot motion affects significantly its performance. This is due to the fact that the maneuvers needed for collision avoidance become now more demanding, since a worst-case scenario may require the robot not only to stop, but also to accelerate in the opposite direction in order to avoid an imminent collision. Thus, by the time an imminent collision is detected in the DB case, there may not be enough actuation capability left to avoid the obstacle explaining why the DB method has such a low performance even in the case of the lowest maximum velocity. On the other hand, the DA method guarantees that the robot is at all times in an ACS. As a result, the robot is able to veer off collision paths earlier.

Static Environments						
	DB			DA		
$v_{\max}$ [m/s]	0.9	1.1	1.2	0.9	1.1	1.2
t_s [s]	0.93	1.116	1.209	0.93	1.116	1.209
success rate (%)	96	96	88	96	96	92
t_g [s]	27.16	24.29	25.07	26.63	24.76	23.98
J_τ [10^4 N^{2}m^{2}s]	1164.63	1495.56	1763.30	1046.58	1397.92	1601.14
l_p [m]	19.56	20.12	20.85	19.51	19.96	20.48
$\delta_{\max}$ [ms]	25.21	25.15	25.46	26.84	27.55	27.11
$\bar{\delta}$ [ms]	15.31	14.26	13.67	16.40	15.96	15.50

Dynamic Environments						
	DB			DA		
$v_{\max}$ [m/s]	0.9	1.1	1.2	0.9	1.1	1.2
t_s [s]	0.93	1.116	1.209	0.93	1.116	1.209
v_o [m/s]	0.45	0.55	0.6	0.45	0.55	0.6
success rate (%)	72	64	40	80	84	80
t_g [s]	29.65	25.52	24.38	30.61	29.78	27.19
J_τ [10^4 N^{2}m^{2}s]	1766.29	2001.42	2009.93	1625.94	2171.06	2510.07
l_p [m]	20.37	20.69	20.77	20.62	21.83	21.31
$\delta_{\max}$ [ms]	26.16	25.65	26.14	28.18	28.56	28.85
$\bar{\delta}$ [ms]	15.47	14.63	14.19	16.87	16.64	16.40

Table 5.1. Averaged results over 25 environments for the proposed dynamics-aware (DA) method vs the distance-based (DB) method. Top: static environments, bottom: dynamic environments.

For illustration, Fig. 5.5 (see also simulation 4 in the accompanying video) shows the behavior of the two methods for $v_{\max} = 1.2$ m/s in one of the dynamic environments through a series of snapshots. Fig. 5.6 reports the corresponding velocity profiles and control inputs for the same scenario.

As in the static environment case, the DB method is not aware of the robot dynamic state and therefore cannot prevent the collision with the moving obstacle. On the other hand, the DA method effectively reaches the goal avoiding nearby obstacles. Note in particular how the early reaction of the DA method in the presence of a moving obstacle (see also the corresponding control inputs in Fig. 5.6) averted the impending collision, while at the same time, the late reaction of the DB method lead to torque saturation at the vicinity of the obstacle, which was, however, not enough to prevent the collision. One can also appreciate the elaborate avoidance maneuver performed by the DA method in front of a moving obstacle combining a

reverse motion and a quick reorientation.

In an attempt to further assess the effectiveness of the proposed method, we tested the performance of the two methods in a more cluttered version of the dynamic environment, increasing the number of moving obstacles to 15. The obstacles are zigzagging at a speed of 0.4 m/s, with direction changes of $\pi/3$ (toward the robot) after traveling a distance of 2.45 m. The robot maximum velocity is $v_{\max} = 1.2$ m/s. This simulation, whose results are only shown in simulation 5 of the accompanying video for compactness, confirms that the robot controlled by the dynamics-aware NMPC method safely reaches its goal even in this challenging environment, while the distance-based NMPC fails.

5.5 Conclusions

In this chapter, we presented a novel real-time NMPC for robot navigation in environments populated by static and/or moving obstacles. Inspired by the concept of inevitable collision state, we defined the notion of ACS (avoidable collision state) and, based on this, formulated a hard constraint on the robot state guaranteeing that it can execute a collision avoidance trajectory in the presence of a dangerous obstacle. The method was compared with a version of the proposed NMPC that considers a purely distance-based collision avoidance constraint.

The comparative simulations showed that the proposed method is generally more effective than the considered distance-based alternative, especially when the robot navigates at high speed in cluttered dynamic environments. They also confirmed our claim that within an NMPC for robot navigation, the use of a collision avoidance constraint that considers information about the whole relative state of the robot with respect to the obstacle and the robot actuation capabilities can lead to an earlier reaction in the presence of an obstacle compared to a collision avoidance constraint that is based on purely distance information.

Future work will be aimed at

- comparing the proposed dynamics-aware collision avoidance constraint with a CBF-based collision avoidance constraint like the one proposed in Chap. 4;
- testing the proposed method in flying robots which are known to perform much more agile motions than a wheeled mobile robot;
- extending the proposed method on multi-body mobile robots (namely, mobile manipulators).

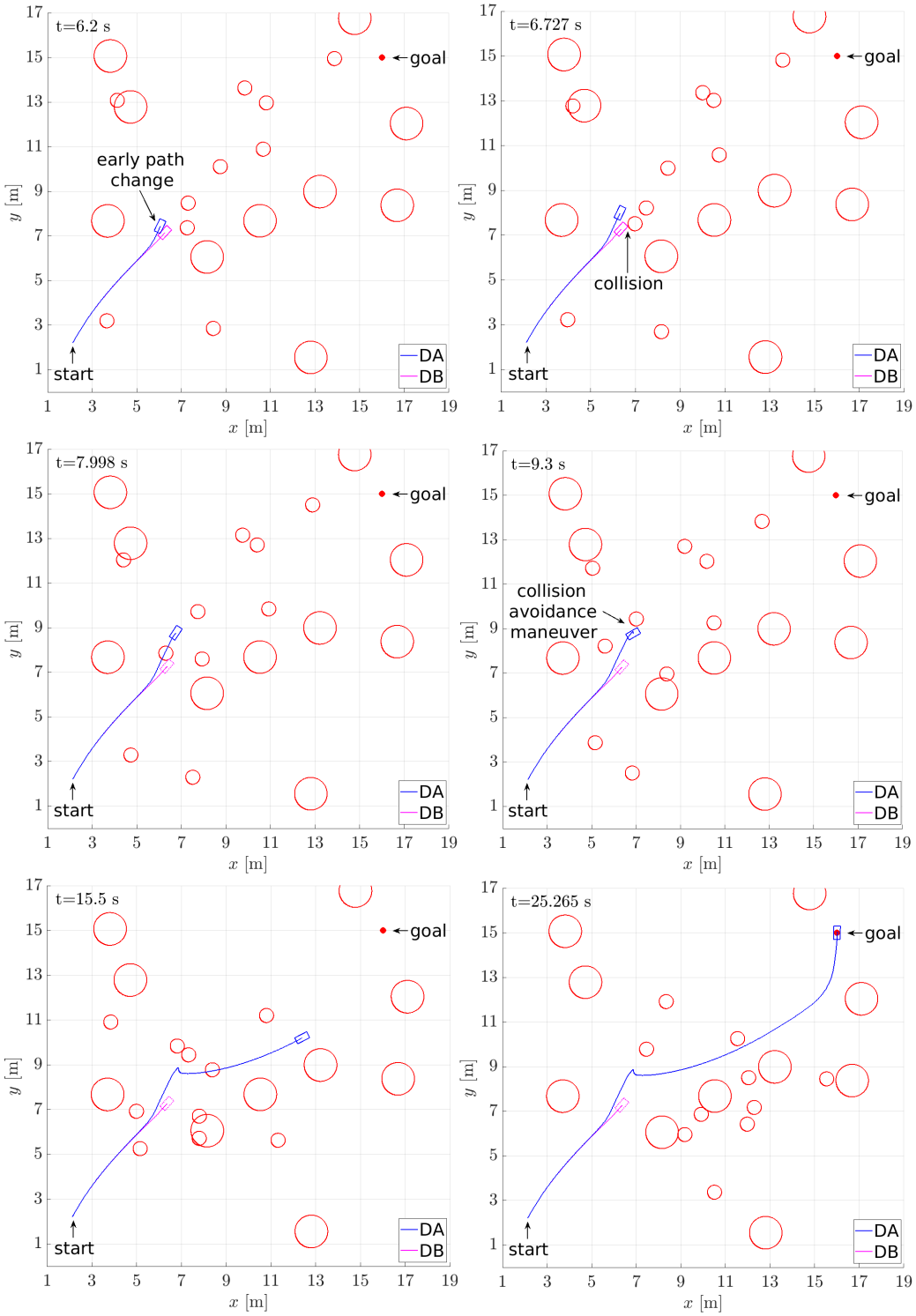

Figure 5.5. Snapshots of the motion generated by the two methods in one of the dynamic environments, with $v_{\max} = 1.2$ m/s. As in the static environment, the DB method cannot avoid collision with an obstacle, whereas the DA method safely navigates to the goal. See also simulation 4 in the accompanying video.

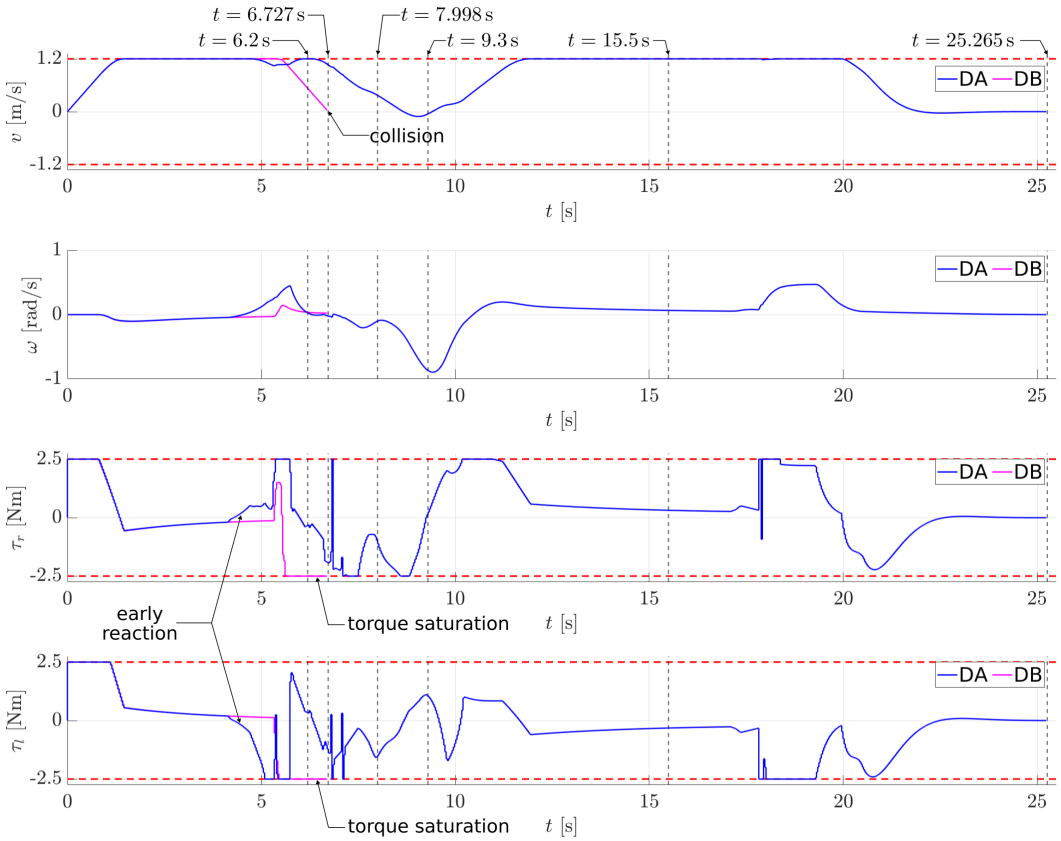

Figure 5.6. The driving and steering velocity profiles and the control inputs generated by the two methods in the dynamic environment, with $v_{\max} = 1.2$ m/s. Each vertical dashed gray line corresponds to one of the time instants in which the snapshots of Fig. 5.5 were taken. See also simulation 4 in the accompanying video.

Part II

Motion generation for mobile manipulators

Chapter 6

Ensuring balance for mobile manipulators via NMPC

This chapter concerns the problem of real-time generation of safe motions for Mobile Manipulators (MM) called to execute tasks that require aggressive motions. The proposed approach is based on a real-time NMPC algorithm that considers the robot full dynamics and appropriate constraints to ensure the robot safety.

6.1 Introduction

The MMs are robotic platforms consisting of a mobile base with one or more manipulators mounted on it. They have a large domain of application spanning from operations in inhospitable environments to household assistance. Their popularity is mainly a result of their inherent versatility, as they combine the mobile base mobility and the manipulator dexterity. Depending on the application, their structure and size vary, having examples of ground, underwater, aerial, or even space mobile manipulators, and are able to perform dexterous tasks in environments where human beings cannot reach or their operation in such environments could expose them to danger. This thesis only focuses on ground and particularly on wheeled mobile manipulators studying the problem of motion generation with particular focus on the safety of the resulting motion.

However complicated the assigned task is, preserving the robot safety and in extension the safety of the environment is of utmost importance. During its motion the robot runs the risk of: (*i*) exceeding its hardware limitations, (*ii*) colliding with the environment and/or itself (self-collision) and (*iii*) losing its balance. Note that, these safety risks for the robot can constitute a safety risk for its surrounding environment too, as a collision with the environment might cause material damage or even harm a human operating close to the robot. Clearly, in order to generate motions for a MM, it is required to take into consideration the aforementioned risks.

In the literature, there are several methods that generate task-consistent motion for MMs. Many approaches taking advantage of the MM structure treat each subsystem (i.e., mobile base and manipulator) separately by assigning sub-tasks to each one of them and solving two separate motion planning problems. A comprehensive survey of methods like this can be found in [78]. However, this approach

does not permit the exploitation of the full potential of a MM and in principle leads to suboptimal motions. This can be avoided by treating the system as a whole. This approach comes with the disadvantage that the high dimensions of the resulting motion planning problem make it computationally intensive. The problem of whole-body motion planning can always be treated as an OCP. In [79] the authors formulate and solve the motion planning problem among obstacles for nonholonomic MM as an OCP. Alternative approaches are based on sampling-based methods (e.g., [80, 81, 82, 83, 84, 85]). However, the aforementioned methods are in principle computationally intensive and can only be applied offline. Clearly, offline solutions are not effective when the robot operates in a dynamically changing environment. So online solutions have to be employed. In order to reduce the required computational time, the authors in [86] attack the motion planning problem for a nonholonomic MM using a discontinuous feedback controller built upon navigation functions that account for collision and singularity avoidance. The authors in [87] also use potential fields in order to drive the robot along a collision-free tunnel built in the workspace of the robot, while repulsive fields push the robot away from the environment obstacles.

Recently, NMPC became an attractive approach for real-time motion generation, applicable also to robots with high DOF, like MMs, thanks to the development of tailored algorithms for its solution, like the RTI scheme [32] or Differential Dynamic Programming (DDP)-like approaches like the Sequential Linear Quadratic (SLQ) algorithm [88], as well as to the development of appropriate software (to mention a few, `ACADO Toolkit` [17], `acados` [77], `Crocoddyl` [89]). In NMPC the safety risks that the robot runs can be considered explicitly by the use of the robot full dynamic model, state and input bounds as well as with appropriate collision avoidance and balance constraints.

Note that in cases where the robot has to execute tasks that permit low speeds without excessive accelerations, the use of the robot full dynamic model is not necessary and its kinematic model can be used instead. An example can be found in [90] where the kinematic model of an MM moving under non-holonomic constraints is considered as prediction model in an NMPC. However, the authors did not include inequality constraints like control input limits and collision avoidance constraints. In order to ensure that the resulting motion will be collision-free the methods in [75, 91, 92, 93] consider appropriate collision avoidance constraints in the proposed MPC schemes.

Clearly, the kinematic model of the robot is essential in order to generate motions that are compatible with a task assigned in terms of a representative point of the robot and in order to respect geometric constraints (e.g., collision avoidance constraints). However, as the motion of the robot becomes faster, one has to consider the robot dynamics in order to enforce kinodynamic feasibility and reason about the robot balance. Although approximations of balance constraints exist that ignore the robot dynamics effect, under the assumption of relatively slow motions (see [92]), these approximations can be dangerous when the effect of the dynamics becomes significant.

In the literature, there are many attempts to evaluate and prevent the MM loss of balance. The motion planning approach in [94] maintains the MM balance when the robot base is stationary. The *Force-Angle* measure [95, 96] predicts and prevents tip-over instabilities when the whole robot is in motion. The extensively

used in humanoids Zero-Moment Point (ZMP) [97] is applied to MMs in [98, 99, 100] neglecting though the moment of inertia of the robot bodies. Clearly, the consideration of a detailed model of the robot dynamics can be computationally intensive. In order to reduce this computational effort, in [101] the authors use approximations of the ZMP position gradient and Hessian matrix, while in [102] such approximations are avoided using appropriate recursive algorithms. Alternative but equivalent to the ZMP methods have also been introduced for the evaluation of the robot balance. In [103, 104] the Stability Twist Constraint (STC) is used in an inverse kinematics solver implemented as a QP. STC constrains the support wrench on the robot base to produce non-positive power with an imaginary twist along each axis of the robot support polygon. Similarly, [105] introduces an algorithm that adjusts the robot motion in order to recover from a loss of balance, considering the moment around the support polygon edges. Note, however, that the approaches of [101, 102, 103, 104, 105] have to be paired with a separate motion planner.

This chapter addresses the problem of motion generation for MMs called to perform aggressive motions. For the solution of the considered problem, we will use the NMPC approach proposed in our work [3]. Specifically, to enforce kinodynamic feasibility to the resulting motion we consider the robot full dynamic model and appropriate state and input bounds while the resulting motion is collision-free thanks to appropriate collision and self-collision avoidance constraints. To ensure that the robot will not lose its balance we adopt the STC constraint, which requires essentially that the exerted by the robot moments around the support polygon edges remain non-negative. To facilitate the real-time solution of the NMPC, the inherently nonlinear balance constraint is linearized using the solution of the previous NMPC iteration.

The chapter is organized as follows. The considered problem is formulated in Sect. 6.2. In Sect. 6.3 we obtain the kinematic and dynamic model of a general wheeled MM. In Sect. 6.4 we outline the proposed NMPC scheme for the generation of the desired motion. In Sect. 6.5 we present the collision and self-collision avoidance constraints to be included in the NMPC, while Sect. 6.6 is dedicated to the considered balance constraint. The simulation results for the considered MM are presented in Sect. 6.7, while some concluding remarks are offered in Sect. 6.8.

6.2 Problem formulation

Consider a general wheeled MM with configuration $\boldsymbol{q}$ taking values in an n-dimensional space. The robot operates in a 3-dimensional workspace $\mathcal{W}$ moving on an horizontal ground. The considered workspace is populated by static and/or moving obstacles. Denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$ and by $\mathcal{O}(t) \subset \mathcal{W}$ the volume occupied by the obstacles at time t.

The robot is called to execute a task in terms of a vector $\boldsymbol{y} \in \mathcal{Y}$ which describes the position of its end-effector and is assigned as a desired trajectory $\boldsymbol{y}_d(t)$, with defined derivatives and $t \in [0, t_f]$ where t_f the duration of the assigned task.

The problem consists in generating in real-time a motion that:

1. starts from the robot initial configuration and follows as close as possible the desired end-effector trajectory $\boldsymbol{y}_d$;

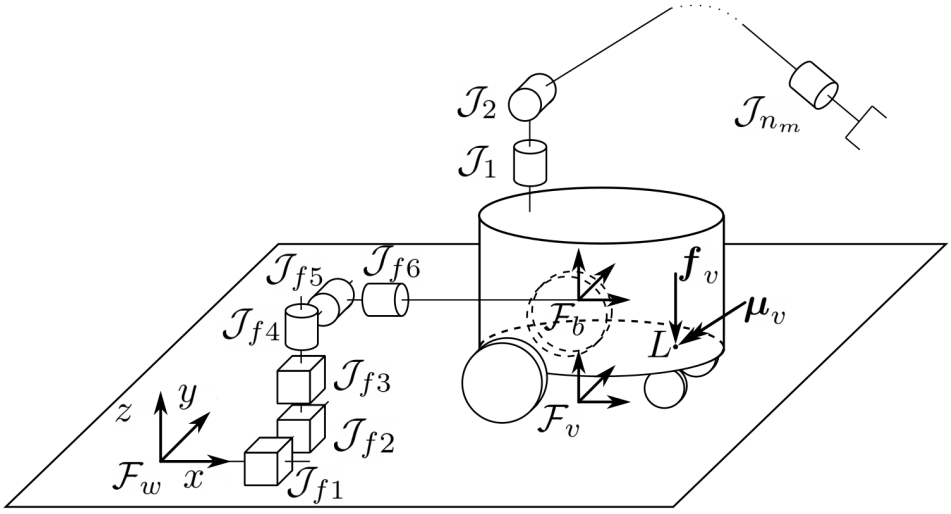

Figure 6.1. A general mobile manipulator consisting of a wheeled mobile base carrying a n_m joint manipulator.

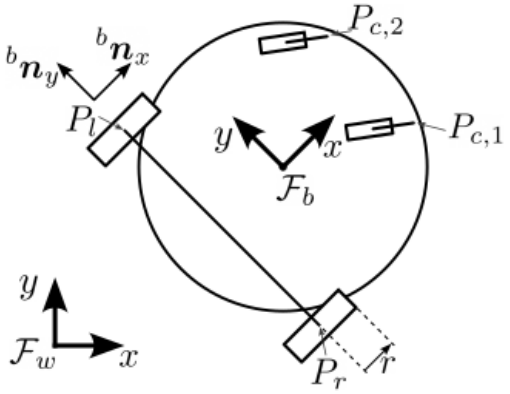

Figure 6.2. Top view of the mobile base of the considered mobile manipulator.

2. is kinodynamically feasible, in the sense that it is consistent with the robot dynamics and respects joint and velocity limits and the joint torque bounds;
3. maintains the robot balance, in the sense that the robot wheels are always in contact with the ground;
4. is collision-free, avoiding not only collisions with the environment obstacles but also self-collisions.

We assume that the robot is always aware of its own state as well as the position and velocity of the obstacles populating its environment.

6.3 Mobile manipulator model

In this work, we will consider a general wheeled mobile manipulator (see Fig. 6.1). The robot consists of a differential-drive mobile base with n_w wheels of which the 2

are actuated and the rest $n_w - 2$ are caster wheels placed for mechanical balance. The mobile base carries a manipulator of n_m joints, equipped with an end-effector.

For our analysis we make the following assumptions:

- the robot bodies are rigid;
- the robot wheels are in *point contact* with the ground, while the contact points that correspond to the driving wheels do not change with respect to a reference frame attached to the vehicle;
- the ground-wheel friction is adequate to prevent slippage.

Referring to Fig. 6.1, denote by $\mathcal{F}_w$ the world reference frame and by $\mathcal{F}_b$ a frame attached to the mobile base. In order to represent the pose of the mobile base we consider 6 fictitious joints that connect the world frame $\mathcal{F}_w$ with the mobile base frame $\mathcal{F}_b$. The joints are arranged as follows: 3 prismatic joints along the x, y and z axes of $\mathcal{F}_w$, followed by 3 revolute joints with axes parallel to the *yaw*, *pitch* and *roll* axes of the mobile base. Note, however, that this arrangement is not unique.

The generalized coordinates of the base are collected in the vector

$$\boldsymbol{q}_b = (x_b, y_b, z_b, \theta_z, \theta_y, \theta_x, \phi_r, \phi_l) \in I\!R^{6+2}$$

where the first six elements are the variables of the fictitious joints and represent the pose of the mobile base, while ϕ_r and ϕ_l are the joint angles of the right and left driving wheels, respectively.

The generalized coordinates of the manipulator are expressed in the vector

$$\boldsymbol{q}_m = (\theta_1, ..., \theta_{n_m}) \in I\!R^{n_m}$$

consisting of the variables of the n_m manipulator joints.

The generalized coordinates of the mobile base and of the manipulator constitute the generalized coordinates of the whole MM that we collect in the vector

$$\boldsymbol{q} = (\boldsymbol{q}_b, \boldsymbol{q}_m) \in I\!R^{6+2+n_m}.$$

6.3.1 Kinematic constraints

The considered robot moves on an horizontal ground. The robot is in contact with the ground through its wheels, while the forces exerted by the ground to the robot at the contact points, constrain the robot motion. Note that in order to obtain the robot model that is going to be used as prediction model in the NMPC, we consider its motion under normal conditions, i.e., the robot maintains its contact with the ground and in extension its balance. Clearly, in order to ensure this, an appropriate constraint has to be added in the NMPC in order to maintain the robot balance as we will do in a following section.

Having as a reference Fig. 6.2, let us denote by $P_r(x_r, y_r, z_r)$ and $P_l(x_l, y_l, z_l)$ the centroids of the driving wheels (we assume cylindrical wheels) and assume that the contact points that corresponds to the driving wheels coincide with the projections of P_r and P_l on the ground. Denote also by $P_{c,i}(x_{c,i}, y_{c,i}, z_{c,i})$ the point on the mobile base at which the i-th caster wheel is connected, with $i = 1, ..., n_w - 2$. Note that

the projection of $P_{c,i}$ on the ground does not coincide with the contact point of the i-th caster wheel. However, it is assumed that their distance and, as a consequence, the resulting moment applied by the caster wheel to the mobile base is negligible. Denote by $\boldsymbol{p}_r$ and $\boldsymbol{p}_l$ the position vectors of points P_r and P_r, respectively, while by $\boldsymbol{p}_{c,i}$ the position vector of $P_{c,i}$, all expressed in the world frame $\mathcal{F}_w$. The vectors $\boldsymbol{p}_r$, $\boldsymbol{p}_l$ and $\boldsymbol{p}_{c,i}$ are related to the mobile base configuration $\boldsymbol{q}_b$ via forward kinematic maps

$$\boldsymbol{p}_r = \boldsymbol{\sigma}_r(\boldsymbol{q}_b) \tag{6.1a}$$

$$\boldsymbol{p}_l = \boldsymbol{\sigma}_l(\boldsymbol{q}_b) \tag{6.1b}$$

$$\boldsymbol{p}_{c,i} = \boldsymbol{\sigma}_{c,i}(\boldsymbol{q}_b), \quad i = 1, ..., n_w - 2. \tag{6.1c}$$

Their velocity vectors are expressed as

$$\dot{\boldsymbol{p}}_r = \frac{\partial \boldsymbol{\sigma}_r(\boldsymbol{q}_b)}{\partial \boldsymbol{q}_b}\dot{\boldsymbol{q}}_b \tag{6.2a}$$

$$\dot{\boldsymbol{p}}_l = \frac{\partial \boldsymbol{\sigma}_l(\boldsymbol{q}_b)}{\partial \boldsymbol{q}_b}\dot{\boldsymbol{q}}_b \tag{6.2b}$$

$$\dot{\boldsymbol{p}}_{c,i} = \frac{\partial \boldsymbol{\sigma}_{c,i}(\boldsymbol{q}_b)}{\partial \boldsymbol{q}_b}\dot{\boldsymbol{q}}_b, \quad i = 1, ..., n_w - 2. \tag{6.2c}$$

No orthogonal to the ground motion In order for the robot to maintain its contact with the ground, the velocity of the contact points along the orthogonal to the ground axis has to be zero. So the following conditions for the velocity of points P_r, P_l and $P_{c,i}$ hold

$$\dot{p}_r^z = \boldsymbol{n}^T\dot{\boldsymbol{p}}_r = \boldsymbol{n}^T\frac{\partial \boldsymbol{\sigma}_r(\boldsymbol{q}_b)}{\partial \boldsymbol{q}_b}\dot{\boldsymbol{q}}_b = 0 \tag{6.3a}$$

$$\dot{p}_l^z = \boldsymbol{n}^T\dot{\boldsymbol{p}}_l = \boldsymbol{n}^T\frac{\partial \boldsymbol{\sigma}_l(\boldsymbol{q}_b)}{\partial \boldsymbol{q}_b}\dot{\boldsymbol{q}}_b = 0 \tag{6.3b}$$

$$\dot{p}_{c,i}^z = \boldsymbol{n}^T\dot{\boldsymbol{p}}_{c,i} = \boldsymbol{n}^T\frac{\partial \boldsymbol{\sigma}_{c,i}(\boldsymbol{q}_b)}{\partial \boldsymbol{q}_b}\dot{\boldsymbol{q}}_b = 0, \quad i = 1, ..., n_w - 2. \tag{6.3c}$$

where $\boldsymbol{n}$ is a unit vector orthogonal to the ground.

The conditions (6.3) also suggest that the components of vectors $\boldsymbol{p}_r$, $\boldsymbol{p}_l$ and $\boldsymbol{p}_{c,i}$ projected on the orthogonal to the ground axis are constant throughout the robot motion

$$p_r^z = \boldsymbol{n}^T\boldsymbol{p}_r = const. \tag{6.4a}$$

$$p_l^z = \boldsymbol{n}^T\boldsymbol{p}_l = const. \tag{6.4b}$$

$$p_{c,i}^z = \boldsymbol{n}^T\boldsymbol{p}_{c,i} = const., \quad i = 1, ..., n_w - 2. \tag{6.4c}$$

where the values of p_r^z, p_l^z and $p_{c,i}^z$ can be determined by the robot dimensions.

Intuitively, one can also realize that the fictitious joints $\mathcal{J}_{f3}$, $\mathcal{J}_{f5}$ and $\mathcal{J}_{f6}$ remain fixed during the robot motion and consequently the corresponding generalized coordinates maintain a constant value

$$z_b, \theta_y, \theta_x = const.$$

This value can be determined by solving the system of equations (6.4).

No lateral wheel motion and the pure rolling condition The assumption of adequate wheel-ground friction suggests that the robot wheels are prevented from slipping sideways (i.e., to a direction orthogonal to the mobile base sagittal plane). The same assumption enforces also pure rolling conditions, i.e., the translational motion of the wheels is a result of its rotation around its axis. Denote by ${}^b\boldsymbol{n}_x$ a unit vector along the x-axis of the base frame $\mathcal{F}_b$ and ${}^b\boldsymbol{n}_y$ a unit vector along the y-axis (see Fig. 6.2). Denote also by ${}^w\boldsymbol{R}_b(\boldsymbol{q}_b)$ the rotation matrix of frame $\mathcal{F}_b$ with respect to $\mathcal{F}_w$. The no-slipping condition can be written for the right and left driving wheels, respectively, as

$$ {}^b\boldsymbol{n}_y^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\dot{\boldsymbol{p}}_r = 0 \tag{6.5a} $$

$$ {}^b\boldsymbol{n}_y^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\dot{\boldsymbol{p}}_l = 0. \tag{6.5b} $$

By substituting (6.2a) and (6.2b) in (6.5) we get the no-lateral-slipping condition for the right and left wheels as

$$ {}^b\boldsymbol{n}_y^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\frac{\partial\boldsymbol{\sigma}_r(\boldsymbol{q}_b)}{\partial\boldsymbol{q}_b}\dot{\boldsymbol{q}}_b = 0 \tag{6.6a} $$

$$ {}^b\boldsymbol{n}_y^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\frac{\partial\boldsymbol{\sigma}_l(\boldsymbol{q}_b)}{\partial\boldsymbol{q}_b}\dot{\boldsymbol{q}}_b = 0 \tag{6.6b} $$

Regarding the pure rolling conditions for the right and left driving wheels, respectively, we have

$$ {}^b\boldsymbol{n}_x^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\dot{\boldsymbol{p}}_r = r\dot{\phi}_r \tag{6.7a} $$

$$ {}^b\boldsymbol{n}_x^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\dot{\boldsymbol{p}}_l = r\dot{\phi}_l. \tag{6.7b} $$

By substituting (6.2a) and (6.2b) in (6.7) we get the pure rolling conditions in the form

$$ {}^b\boldsymbol{n}_x^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\frac{\partial\boldsymbol{\sigma}_r(\boldsymbol{q}_b)}{\partial\boldsymbol{q}_b}\dot{\boldsymbol{q}}_b - r\dot{\phi}_r = 0 \tag{6.8a} $$

$$ {}^b\boldsymbol{n}_x^{Tw}\boldsymbol{R}_b^T(\boldsymbol{q}_b)\frac{\partial\boldsymbol{\sigma}_l(\boldsymbol{q}_b)}{\partial\boldsymbol{q}_b}\dot{\boldsymbol{q}}_b - r\dot{\phi}_r = 0 \tag{6.8b} $$

The kinematic constraints Overall the kinematic constraints that restrict the motion of the MM on the horizontal ground are (6.3), (6.6) and (6.8), which can be also expressed in Pfaffian form as $\boldsymbol{A}^T(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$. Note that the elements of $\boldsymbol{A}(\boldsymbol{q})$ that correspond to the manipulator are set to zero.

6.3.2 Kinematic model

As it was mentioned above, due to the robot motion on the horizontal ground the robot configuration $\boldsymbol{q}$ consists of generalized coordinates that evolve during the robot motion, denoted by the vector $\boldsymbol{q}_e = (x_b, y_b, \theta_z, \phi_r, \phi_l, \boldsymbol{q}_m)$, and of those that correspond to the fictitious joints $\mathcal{J}_{f3}$, $\mathcal{J}_{f5}$ and $\mathcal{J}_{f6}$, denoted by the vector $\boldsymbol{q}_f = (z_b, \theta_y, \theta_x)$, which remain constant throughout the robot motion if the robot maintains

its balance. These two vectors can be extracted from the robot configuration $\boldsymbol{q}$ using the selection matrices $\boldsymbol{P}_e$ and $\boldsymbol{P}_f$ such that

$$\boldsymbol{q}_e = \boldsymbol{P}_e \boldsymbol{q},$$

$$\boldsymbol{q}_f = \boldsymbol{P}_f \boldsymbol{q}.$$

If we denote by $\boldsymbol{\nu} = (\boldsymbol{\nu}_b, \dot{\boldsymbol{q}}_m)$ the robot velocity vector, where $\boldsymbol{\nu}_b = (\dot{\phi}_r, \dot{\phi}_l)$ is the vector of the mobile base velocities[1], the robot kinematic model can be written as

$$\dot{\boldsymbol{q}}_e = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \tag{6.9}$$

where $\boldsymbol{G}(\boldsymbol{q})$ is a matrix whose columns span the null space of $\boldsymbol{A}^T(\boldsymbol{q})\boldsymbol{P}_e^T$. Differentiation of (6.9) with respect to time gives

$$\ddot{\boldsymbol{q}}_e = \boldsymbol{G}(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{\boldsymbol{G}}(\boldsymbol{q})\boldsymbol{\nu}. \tag{6.10}$$

Regarding the end-effector position $\boldsymbol{y}$, in terms of which the end-effector task is assigned, it is related to the robot configuration via a forward kinematics map

$$\boldsymbol{y} = \boldsymbol{k}(\boldsymbol{q}). \tag{6.11}$$

The expressions for its velocity and acceleration are

$$\dot{\boldsymbol{y}} = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{\nu} \tag{6.12}$$

$$\ddot{\boldsymbol{y}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{\boldsymbol{J}}(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu} \tag{6.13}$$

where $\boldsymbol{J}(\boldsymbol{q}) = \partial \boldsymbol{k}(\boldsymbol{q})/\partial \boldsymbol{q}\boldsymbol{G}(\boldsymbol{q})$ the robot Jacobian and $\dot{\boldsymbol{J}}(\boldsymbol{q}, \boldsymbol{\nu})$ its time derivative.

6.3.3 Dynamic model

For the dynamic model of the MM we use the Lagrange formulation. For this purpose, the kinetic and potential energy of the considered robot has to be computed in order to determine the robot equations of motion. The process is briefly presented in Appendix A. The resulting dynamic model of the robot can be written in the form

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}, \tag{6.14}$$

being $\boldsymbol{B}(\boldsymbol{q}) \in I\!R^{n \times n}$ the inertia matrix, $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in I\!R^n$ the vector of velocity and gravitational terms, $\boldsymbol{u} \in I\!R^{n_u}$ the vector of the forces/torques applied by the n_u robot actuators, $\boldsymbol{S}(\boldsymbol{q}) \in I\!R^{n \times n_u}$ the matrix that maps the actuator forces to forces performing work on the generalized coordinates, and $\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}$ the vector of the forces exerted to the robot by its contact with the ground expressed at the generalized coordinates level, with $\boldsymbol{\lambda}$ being the vector of the contact forces.

By left multiplying (6.14) by $\boldsymbol{P}_e$ and after simple manipulation we get

$$\boldsymbol{P}_e\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_e^T\ddot{\boldsymbol{q}}_e + \boldsymbol{P}_e\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{P}_e\boldsymbol{S}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{P}_e\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}. \tag{6.15}$$

[1]Note that as we mentioned at Chap. 3 one can choose a different pseudovelocity vector $\boldsymbol{\nu}_b$ comprised by the driving and steering velocity of the mobile base.

If we define the robot state as the vector $\boldsymbol{x} = (\boldsymbol{q}_e, \boldsymbol{\nu}) \in \mathcal{X}$ we can express the robot state-space reduced model [52] as

$$\dot{\boldsymbol{x}} = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}) = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})\left(\boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu})\right) \end{pmatrix} \tag{6.16}$$

with

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q}) &= \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{P}_e\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_e^T\boldsymbol{G}(\boldsymbol{q}) \\ \boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu}) &= \boldsymbol{G}^T(\boldsymbol{q})\left(\boldsymbol{P}_e\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_e^T\dot{\boldsymbol{G}}(\boldsymbol{q})\boldsymbol{\nu} + \boldsymbol{P}_e\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})\right) \\ \boldsymbol{E}(\boldsymbol{q}) &= \boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{P}_e\boldsymbol{S}(\boldsymbol{q}). \end{aligned}$$

6.3.4 Contact forces

We already mentioned that $\boldsymbol{\lambda}$ is the vector of the contact forces exerted by the ground to the robot. As we already assumed, the ground-wheel friction conditions can balance the parallel to the ground forces exerted by the robot. In the following, we will only look at the orthogonal to the ground elements of $\boldsymbol{\lambda}$. Their determination, however, is not trivial as the balance problem is hyperstatic when the contact points are more than 3. Nevertheless, our intention is to argue about the robot balance, so it is sufficient to determine the sum of the orthogonal to the ground forces and the parallel to the ground moments exerted by the robot to the ground.

Note that the vector $\boldsymbol{P}_f\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}$ represents the contact forces at the generalized coordinates level that constrain the base motion along the axes of the joints $\mathcal{J}_{f3}$, $\mathcal{J}_{f5}$ and $\mathcal{J}_{f6}$. If we left multiply (6.14) by $\boldsymbol{P}_f$, we get the expression

$$\boldsymbol{P}_f\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_e^T\ddot{\boldsymbol{q}}_e + \boldsymbol{P}_f\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{P}_f\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}. \tag{6.17}$$

Note that in (6.17), $\boldsymbol{P}_f\boldsymbol{S}(\boldsymbol{q})\boldsymbol{u} = \boldsymbol{0}$ since the robot actuators do not perform work on the generalized coordinates $\boldsymbol{q}_f$. If we denote by f_{f3} the force that the robot exerts along the joint $\mathcal{J}_{f3}$ and by μ_{f5} and μ_{f6} the moments that the robot exerts around the joints $\mathcal{J}_{f5}$ and $\mathcal{J}_{f6}$ respectively, we have

$$\begin{pmatrix} f_{f3} \\ \mu_{f5} \\ \mu_{f6} \end{pmatrix} = -\boldsymbol{P}_f\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}. \tag{6.18}$$

Substituting (6.17), (6.10) and (6.16) in (6.18) we get these forces and moments as a function of the robot state and control inputs

$$\begin{aligned} \begin{pmatrix} f_{f3} \\ \mu_{f5} \\ \mu_{f6} \end{pmatrix} = &-\boldsymbol{P}_f\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_r^T\boldsymbol{G}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{P}_f\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_r^T\boldsymbol{G}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{m}(\boldsymbol{q}, \boldsymbol{\nu}) \\ &- \boldsymbol{P}_f\boldsymbol{B}(\boldsymbol{q})\boldsymbol{P}_r^T\dot{\boldsymbol{G}}(\boldsymbol{q})\boldsymbol{\nu} - \boldsymbol{P}_f\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{aligned} \tag{6.19}$$

Denote now by $\mathcal{F}_v$ a frame that has axes parallel to $\mathcal{F}_b$ and lies on the projection of $\mathcal{F}_b$ on the ground (see Fig.6.1). The forces orthogonal to the ground and the

moments parallel to the ground that the robot exerts at a point L of the ground can be expressed in $\mathcal{F}_v$ as:

$$\boldsymbol{f}_v = {}^w\boldsymbol{R}_v{}^T(\boldsymbol{q}) f_{f3} \hat{\boldsymbol{z}}^w_{f3}, \tag{6.20}$$

$$\boldsymbol{\mu}_v = {}^w\boldsymbol{R}_v{}^T(\boldsymbol{q}) \left(\mu_{f5} \hat{\boldsymbol{z}}^w_{f5} + \mu_{f6} \hat{\boldsymbol{z}}^w_{f6} + [\boldsymbol{c}]_\times f_{f3} \hat{\boldsymbol{z}}^w_{f3} \right) \tag{6.21}$$

where ${}^w\boldsymbol{R}_v(\boldsymbol{q})$ is the rotation matrix of $\mathcal{F}_v$ with respect to $\mathcal{F}_w$, $\hat{\boldsymbol{z}}^w_{f3}$, $\hat{\boldsymbol{z}}^w_{f5}$ and $\hat{\boldsymbol{z}}^w_{f6}$ are the unit vectors of the joints $\mathcal{J}_{f3}$, $\mathcal{J}_{f5}$ and $\mathcal{J}_{f6}$ axes in the world frame respectively, $\boldsymbol{c}$ is the position vector that starts from the point of application of $f_{f3}\hat{\boldsymbol{z}}^w_{f3}$ and ends at L expressed in the world frame and $[\boldsymbol{c}]_\times$ is the cross product operator which represents a skew-symmetric matrix built from the elements of $\boldsymbol{c}$.

6.4 Proposed NMPC approach

The considered real-time motion generation problem will be solved using an appropriate NMPC algorithm. Consider the NLP of the form (2.8) to be solved at the generic time t_k. Denote by T the prediction horizon, by δ the sampling time and by $N = T/\delta$ the number of control intervals in the prediction horizon. Denote also by $\boldsymbol{x}_{k|i}$ and $\boldsymbol{u}_{k|i}$ the predicted robot state and control inputs at time instants t_{k+i}, while by $\boldsymbol{y}_{k|i}$ and $\dot{\boldsymbol{y}}_{k|i}$ denote the predicted position and velocity of the end-effector at the same time instant. We define as $\boldsymbol{e}_{k|i} = \boldsymbol{y}_d(t_{k+i}) - \boldsymbol{y}_{k|i}$ the predicted task error at t_{k+i} and as $\dot{\boldsymbol{e}}_{k|i} = \dot{\boldsymbol{y}}_d(t_{k+i}) - \dot{\boldsymbol{y}}_{k|i}$ its derivative at the same time instant, where $\boldsymbol{y}_d(t_{k+i})$ and $\dot{\boldsymbol{y}}_d(t_{k+i})$ are the desired end-effector position and velocity at time t_{k+i}, respectively.

Our objective is to make the task error as small as possible while guaranteeing the safety requirements, possibly using the minimum control effort. So, we can express the running and terminal costs, respectively, as:

$$V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{e}^T_{k|i}\boldsymbol{Q}_p\boldsymbol{e}_{k|i} + \dot{\boldsymbol{e}}^T_{k|i}\boldsymbol{Q}_d\dot{\boldsymbol{e}}_{k|i} + \boldsymbol{\nu}^T_{k|i}\boldsymbol{S}\boldsymbol{\nu}_{k|i} + \boldsymbol{u}^T_{k|i}\boldsymbol{R}_u\boldsymbol{u}_{k|i} \tag{6.22}$$

$$V_{k|N}(\boldsymbol{x}_{k|N}) = \boldsymbol{e}^T_{k|N}\boldsymbol{Q}_{p,N}\boldsymbol{e}_{k|N} + \dot{\boldsymbol{e}}^T_{k|N}\boldsymbol{Q}_{d,N}\dot{\boldsymbol{e}}_{k|N} + \boldsymbol{\nu}^T_{k|N}\boldsymbol{S}_N\boldsymbol{\nu}_{k|N}. \tag{6.23}$$

where $\boldsymbol{Q}_p$, $\boldsymbol{Q}_d$, $\boldsymbol{S}$ and $\boldsymbol{R}_u$ are the weighting matrices for the task error, its derivative, the robot velocity and the control effort throughout the prediction horizon, while $\boldsymbol{Q}_{p,N}$, $\boldsymbol{Q}_{d,N}$ and $\boldsymbol{S}_N$ are the weighting matrices for the task error, its derivative and the robot velocity at the final time instant.

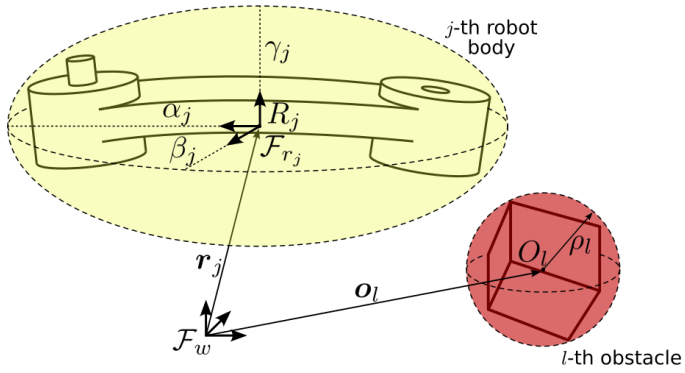

Figure 6.3. The bounding geometries used for the collision avoidance constraints between the robot and the environment obstacles. A bounding ellipsoid is used to envelop the j-th robot body and a bounding sphere is used to envelop the l-th obstacle.

The resulting NLP that will be solved at time instant t_k is:

$$\min_{\substack{\boldsymbol{u}_{k|0},...,\boldsymbol{u}_{k|N-1},\\ \boldsymbol{x}_{k|0},...,\boldsymbol{x}_{k|N}}} \sum_{i=0}^{N-1} V_{k|i}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) + V_{k|N}(\boldsymbol{x}_{k|N}) \tag{6.24a}$$

subject to:

$$\boldsymbol{x}_{k|0} - \boldsymbol{x}_k = \boldsymbol{0} \tag{6.24b}$$

$$\boldsymbol{x}_{k|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{k|i}, \boldsymbol{u}_{k|i}) = \boldsymbol{0}, \quad i = 0, ..., N-1 \tag{6.24c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_{k|i} \leq \boldsymbol{x}_{\max}, \quad i = 0, ..., N \tag{6.24d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_{k|i} \leq \boldsymbol{u}_{\max}, \quad i = 0, ..., N-1 \tag{6.24e}$$

$$\textit{collision avoidance constraints at } t_k, ..., t_{k+N} \tag{6.24f}$$

$$\textit{self-collision avoidance constraints at } t_k, ..., t_{k+N} \tag{6.24g}$$

$$\textit{balance constraints at } t_k, ..., t_{k+N-1} \tag{6.24h}$$

where $\boldsymbol{x}_k$ represents the current robot state, $\boldsymbol{\phi}_{\text{d-t}}(\cdot,\cdot)$ represents the discrete-time dynamics of the robot obtained via numerical integration of (6.16) under the assumption of piecewise-constant control inputs, while $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$ and $\boldsymbol{u}_{\min}$, $\boldsymbol{u}_{\max}$ are respectively the lower/upper bounds on the state variables and on the control inputs.

Regarding the collision avoidance and the balance constraints, they will be presented in full detail in Sections 6.5 and 6.6 respectively.

6.5 Collision avoidance

In order to design collision avoidance constraints between the robot and the environment obstacles we are going to consider bounding ellipsoids for the former and bounding spheres for the latter. In particular, for the j-th robot body, where $j \in \{1, ..., n_b\}$ with n_b the number of robot bodies, we consider a bounding ellipsoid that completely contains it (see Fig. 6.3). The centroid of the ellipsoid is denoted

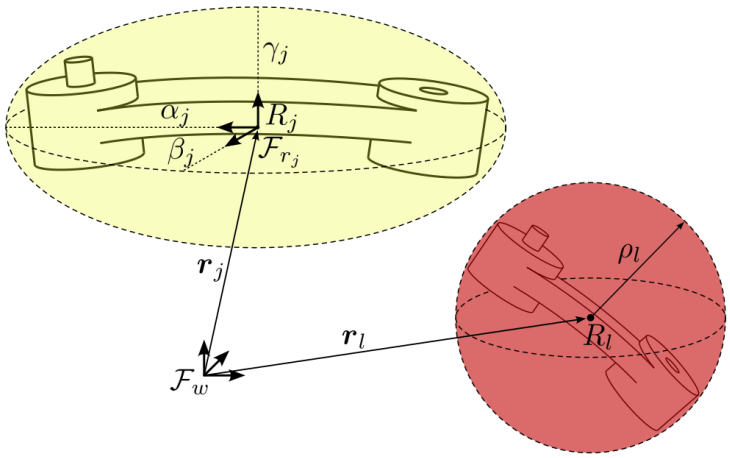

Figure 6.4. The bounding geometries used for the collision avoidance constraints between two robot bodies susceptible to collision (self-collision). A bounding ellipsoid is used for the larger body while a bounding sphere is used for the smaller one.

by R_j and by α_j, β_j and γ_j we denote the length of the principal semi-axes of the ellipsoid. By $\boldsymbol{r}_j$ denote the position of R_j in the world frame, which is related to the robot configuration via a forward kinematics map $\boldsymbol{r}_j = \boldsymbol{\sigma}_j(\boldsymbol{q})$. Finally, we define a reference frame $\mathcal{F}_{r_j}$ with origin at R_j and axes aligned with the principal axes of the ellipsoid. Its rotation matrix with respect to the world frame $\mathcal{F}_w$ is denoted by $\boldsymbol{R}_{r_j}(\boldsymbol{q})$.

Regarding the l-th obstacle $\mathcal{O}_l \subset \mathcal{O}$, where $l \in \{1, ..., n_o\}$ with n_o the number of the obstacles, we consider a bounding sphere with center placed at O_l and radius ρ_l, large enough in order to envelop the volume of the obstacle. By $\boldsymbol{o}_l$ we denote the position vector of O_l.

To ensure that the j-th robot body and the l-th obstacle do not collide, we can use as in [106] the inequality constraint

$$h_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l) = (\boldsymbol{r}_j - \boldsymbol{o}_l)^T \boldsymbol{R}_{r_j} \boldsymbol{H}_{j,l} \boldsymbol{R}_{r_j}^T (\boldsymbol{r}_j - \boldsymbol{o}_l) - 1 \geq 0 \tag{6.25}$$

with

$$\boldsymbol{H}_{j,l} = \text{diag}\{(\alpha_j + \rho_l + d_c)^{-2}, (\beta_j + \rho_l + d_c)^{-2}, (\gamma_j + \rho_l + d_c)^{-2}\}.$$

where $d_c > 0$ is a small safety clearance. The collision avoidance constraints to be applied at (6.24f) can be written as

$$h_{j,l}(\boldsymbol{x}_{k|i}, \boldsymbol{o}_{l,k|i}) \geq 0, \quad i = 0, ..., N, \quad j = 1, ..., n_b, \quad l = 1, ..., n_o$$

where $\boldsymbol{o}_{l,k|i}$ is the position of the the l-th obstacle at the predicted time instant t_{k+i}.

For the self-collision avoidance constraint let us consider the j-th robot body and the l-th robot body, with $j \neq l$. We say that the two bodies are susceptible to collision if they are structurally reachable, in the sense that there exists a robot configuration $\boldsymbol{q}$ in which the two bodies can collide, and their collision cannot be avoided by the considered joint bounds in the NMPC scheme (6.24d). We denote by $\mathcal{S}$ the set of pairs of indices (j, l) that correspond to the robot bodies that are susceptible to collision.

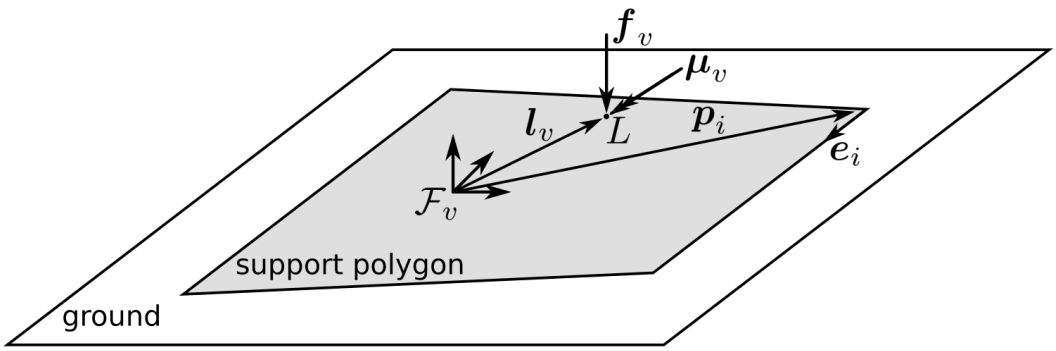

Figure 6.5. The robot support polygon. By $\boldsymbol{e}_i$ we denote the unit vector of its i-th edge and by $\boldsymbol{p}_i$ the position of its starting point, all expressed in $\mathcal{F}_v$.

Given a pair of susceptible to collision robot bodies, we follow the same approach as for the collision between the robot and the environment obstacles. Specifically, we assign to the robot bodies the bounding ellipsoid and the bounding sphere based on their shape (see Fig. 6.4). Let us assign to the j-th body the bounding ellipsoid with its centroid denoted by R_j and the length of its principal semi-axes denoted by α_j, β_j and γ_j. To the l-th body we assign a bounding sphere with its center placed at R_l and with radius ρ_l, large enough in order to envelop the volume of the body. By $\boldsymbol{r}_l$ we denote the position vector of R_l which is now a function of the robot configuration, $\boldsymbol{r}_l = \boldsymbol{\sigma}_l(\boldsymbol{q})$.

To ensure that the j-th and the l-th robot bodies will not collide we use the inequality constraint

$$h^s_{j,l}(\boldsymbol{x}) = (\boldsymbol{r}_j - \boldsymbol{r}_l)^T \boldsymbol{R}_{r_j} \boldsymbol{H}^s_{j,l} \boldsymbol{R}^T_{r_j} (\boldsymbol{r}_j - \boldsymbol{r}_l) - 1 \geq 0, \quad (j,l) \in \mathcal{S} \tag{6.26}$$

where $\boldsymbol{r}_j = \boldsymbol{\sigma}_j(\boldsymbol{q})$ the position vector of R_j, $\boldsymbol{R}_{r_j}(\boldsymbol{q})$ the rotation matrix of $\mathcal{F}_{r_j}$ with respect to $\mathcal{F}_w$ and

$$\boldsymbol{H}^s_{j,l} = \text{diag}\{(\alpha_j + \rho_l + d_s)^{-2}, (\beta_j + \rho_l + d_s)^{-2}, (\gamma_j + \rho_l + d_s)^{-2}\},$$

where $d_s > 0$ is a small safety clearance. Note, that now the self-collision avoidance constraint depends only on the robot state.

So, the self-collision avoidance constraints to be included in (6.24g) can be written as

$$h^s_{j,l}(\boldsymbol{x}_{k|i}) \geq 0, \quad i = 0, ..., N, \quad (j,l) \in \mathcal{S}$$

6.6 Robot balance

In this section, we first present the criterion that we will use for the evaluation of the robot balance. Then, we derive the constraint to be applied in (6.24h) and introduce an additional term in the cost function to improve the robot balance.

6.6.1 Balance criterion

As a criterion to evaluate the robot balance, we use the moments that the robot exerts around the support polygon edges. The robot maintains its balance if the

resulting moments are non-negative. Denote by $\boldsymbol{e}_i$ the unit vector of the i-th edge and by $\boldsymbol{p}_i$ the position of its starting point, both expressed in $\mathcal{F}_v$ (see Fig. 6.5). If μ_i is the moment that the robot exerts around the i-th edge of the support polygon then the criterion for robot balance is expressed as:

$$\mu_i = \boldsymbol{e}_i^T(-(\boldsymbol{p}_i - \boldsymbol{l}_v) \times \boldsymbol{f}_v + \boldsymbol{\mu}_v) \geq 0, \quad \forall\, i = 1, ..., n_e, \tag{6.27}$$

where $\boldsymbol{l}_v$ the position vector of L with respect to $\mathcal{F}_v$ and n_e the number of support polygon edges. Note that constraining the moments around the support polygon edges to remain non-negative is equivalent to constraining the ZMP to lie within the support polygon (see Appendix B).

Substituting (6.20), (6.21) and (6.19) in (6.27) and after simple manipulation we get the balance criterion in the form:

$$\boldsymbol{A}_b(\boldsymbol{x})\boldsymbol{u} \leq \boldsymbol{b}_b(\boldsymbol{x}), \tag{6.28}$$

where the expressions of $\boldsymbol{A}_b$ and $\boldsymbol{b}_b$ can easily derive from the aforementioned equations.

6.6.2 Balance constraint

As we already mentioned, for the solution of the NMPC we are using the RTI scheme based on an SQP algorithm. To reduce the computational time, software like `ACADO Toolkit`, which is the one used here, are based on the symbolic representation of the OCP and the generation of optimized code. Although this method is very effective for a fast and reliable solution of the optimization problem it comes with a drawback. When the prediction model and the considered constraints are long and complex, like in the case of a MM, this approach can lead to a memory overhead in common processing platforms.

In order to deal with this issue, instead of using the inherently nonlinear balance constraint (6.28) directly in the NLP, we take advantage of the recursive nature of the NMPC in order to simplify it. Specifically, let us first collect the robot state and control inputs predicted at time t_k in the vectors

$$\boldsymbol{X}_k = \left(\boldsymbol{x}_{k|0}, ..., \boldsymbol{x}_{k|N}\right),$$

$$\boldsymbol{U}_k = \left(\boldsymbol{u}_{k|0}, ..., \boldsymbol{u}_{k|N-1}\right).$$

and let us consider that the solution of the NMPC does not change significantly between two consecutive control cycles. For the balance constraints of the NLP to be solved at time instant t_k we will use the NLP solution $\boldsymbol{U}_{k-1}$ obtained at time instant t_{k-1}. Starting from the initial state $\boldsymbol{x}_{k-1|0}$ and using the control sequence $\boldsymbol{U}_{k-1}$, we can integrate the robot equations to obtain its trajectory throughout the prediction horizon computed at time instant t_{k-1}, that is $\boldsymbol{X}_{k-1} = (\boldsymbol{x}_{k-1|0}, \boldsymbol{x}_{k-1|1}, .., \boldsymbol{x}_{k-1|N})$. Given that the predictive model is accurate, by applying the control input $\boldsymbol{u}_{k-1|0}$ at time instant t_{k-1} we get the robot initial state at time instant t_k, i.e., $\boldsymbol{x}_{k|0} = \boldsymbol{x}_{k-1|1}$. Assuming no significant changes in the environment between the consecutive time instants t_{k-1} and t_k, we can use $\boldsymbol{X}_{k-1}$ as an estimation of the first $N-1$ states of

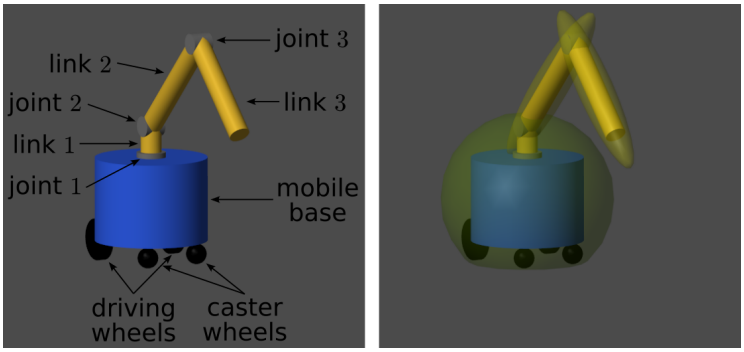

Figure 6.6. The 3-link mobile manipulator considered for the simulations (left). For the collision avoidance constraints, we use bounding ellipsoids in order to envelop the links 2 and 3 and a bounding sphere in order to envelop the mobile base (right).

the trajectory $\boldsymbol{X}_k$. We will use this estimation in order to evaluate $\boldsymbol{A}_b(\boldsymbol{x})$ and $\boldsymbol{b}_b(\boldsymbol{x})$ in (6.28) for each time instant throughout the prediction horizon. The resulting balance constraints that will be applied at (6.24h) of the NLP to be solved at t_k are now linear combinations of the control inputs:

$$\begin{aligned}
\boldsymbol{A}_b(\boldsymbol{x}_{k|0})\boldsymbol{u}_{k|0} &\leq \boldsymbol{b}_b(\boldsymbol{x}_{k|0}) \\
\boldsymbol{A}_b(\boldsymbol{x}_{k-1|2})\boldsymbol{u}_{k|1} &\leq \boldsymbol{b}_b(\boldsymbol{x}_{k-1|2}) \\
&\vdots \\
\boldsymbol{A}_b(\boldsymbol{x}_{k-1|N})\boldsymbol{u}_{k|N-1} &\leq \boldsymbol{b}_b(\boldsymbol{x}_{k-1|N})
\end{aligned}$$

Note that the solution of the NLP might entail some *predicted* control inputs that lead to unbalanced states, due to the obsolete information used by the previous solution. However, the control input that will be applied to the robot at the time instant t_k, namely $\boldsymbol{u}_{k|0}$, guarantees balance at time instant t_k since $\boldsymbol{x}_{k|0}$ is independent of the NMPC solution at t_k.

6.6.3 Improving balance

Even if the balance constraint is satisfied, the robot might get dangerously close to losing balance, unless we introduce an appropriate term in the cost function that improves the robot balance. Thus, we add in the running cost the term $\boldsymbol{v}_{k|i}^T \boldsymbol{\Lambda} \boldsymbol{v}_{k|i}$, where $\boldsymbol{v}_{k|i} = \boldsymbol{A}_b(\boldsymbol{x}_{k-1|i+1})\boldsymbol{u}_{k|i} - \boldsymbol{b}_b(\boldsymbol{x}_{k-1|i+1})$ and $\boldsymbol{\Lambda}$ the corresponding weighting matrix. This term helps to evenly distribute the moments among the edges of our support polygon.

6.7 Simulations

In order to show the effectiveness of the proposed method, we conducted a series of simulations assigning to the robot end-effector tasks that require aggressive motions. All the simulations were implemented using Simscape within Simulink on an Intel Core i9-9900K CPU running at 3.60 GHz. For the numerical solution of the NMPC, we used the RTI scheme [107] implemented within the `ACADO Toolkit` [17].

The robotic platform: The robot used for our simulations is an MM consisting of a differential-drive mobile base with two caster wheels carrying a 3-link manipulator on top (see Fig. 6.6). The height of the mobile base is 0.4 m and the length of the manipulator (fully extended) is 1.35 m. The total robot weight is 44.5 kg. The robot is torque controlled. By τ_r and τ_l we denote the torques at the right and left driving wheels respectively and by τ_1, τ_2 and τ_3 the torques of the actuators at the manipulator joints (the enumeration starts from the closest to the base joint).

Collision avoidance with the environment: For the collision avoidance constraints between the robot and its environment, we used 2 bounding ellipsoids in order to envelop the 2$^{\text{nd}}$ and 3$^{\text{rd}}$ link of the manipulator and a bounding sphere (ellipsoid with semi-axes of equal size) in order to envelop the robot mobile base. In an attempt to reduce the number of the considered constraints, we consider for collision avoidance only the closest obstacle to each robot body. The closeness is evaluated using (6.25).

Self-collision avoidance: For the considered robot, the only bodies susceptible to self-collision are the 3$^{\text{rd}}$ link of the manipulator and the mobile base. Note that the 2$^{\text{nd}}$ link will be protected from collision with the base by the considered joint limits. To build the corresponding self-collision avoidance constraint of the form (6.26) we envelop the manipulator link with the bounding ellipsoid and the base with the bounding sphere.

Balance constraints: Regarding the balance constraints, the considered robot is in contact with the ground via 2 driving and 2 caster wheels, forming a support polygon with 4 edges. If we locate the frame $\mathcal{F}_v$ along the axis of the robot base, the position of the support polygon edges expressed in this frame is:

$$\begin{aligned}
\boldsymbol{p}_1 &= (0.2, -0.133, 0)\,, \\
\boldsymbol{p}_2 &= (0.2, 0.133, 0)\,, \\
\boldsymbol{p}_3 &= (-0.15, 0.2, 0)\,, \\
\boldsymbol{p}_4 &= (-0.15, -0.2, 0)\,.
\end{aligned}$$

However, we consider a more conservative support polygon[2] by reducing its dimensions by 10%.

We wish to compare the proposed method with two approaches from the literature: (1) the NMPC proposed in [92] and (2) the QP proposed in [103]. These methods were originally applied to velocity controlled robots, considering only their kinematic models. So for the purpose of this comparison, the compared methods were slightly modified in order to be applied to the considered torque controlled robot.

In the rest of this section we briefly present the compared methods and then we show the simulation results.

[2]Note that for the robot that we consider for our simulations, the use of the reduced support polygon accounts for integration errors of the simulation environment. However, the consideration of a reduced support polygon is a valid approach when one deals with more realistic robots equipped with caster wheels whose exact configuration, and in extension the position of the contact point cannot be predicted.

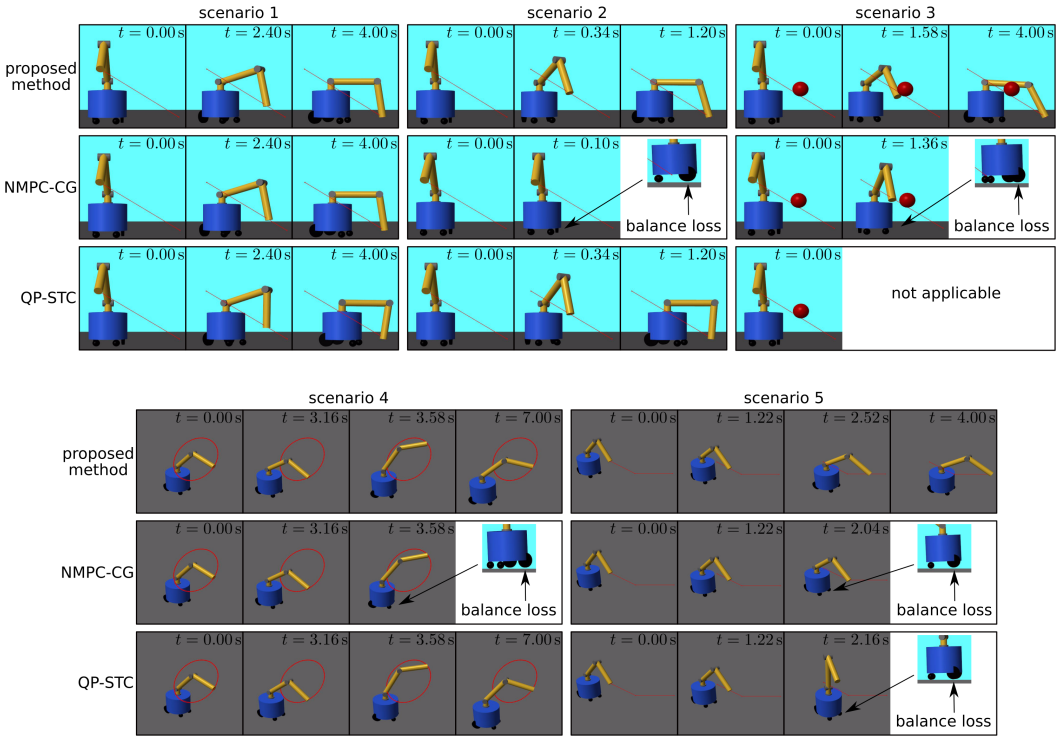

Figure 6.7. Snapshots of the motion generated by the proposed method and the two compared methods in the five scenarios.

6.7.1 Compared methods

NMPC using a ZMP position approximation The NMPC presented in [92] uses a balance constraint built upon the approximation of the ZMP position:

$$\tilde{\boldsymbol{p}}_{\text{zmp}} = \frac{\boldsymbol{n}_g \times (\boldsymbol{p}_{\text{cog}} \times \boldsymbol{f}_g - \boldsymbol{p}_{ee} \times \boldsymbol{f}_{ee} - \boldsymbol{\mu}_e)}{\boldsymbol{n}_g \cdot (\boldsymbol{f}_g - \boldsymbol{f}_{ee})},$$

where $\boldsymbol{n}_g$ is the ground normal vector, $\boldsymbol{p}_{\text{cog}}$ and $\boldsymbol{f}_g$ are the position of the robot center of gravity and the gravitational forces applied to the robot, $\boldsymbol{p}_{ee}$ the position of the end-effector and $\boldsymbol{f}_{ee}$ and $\boldsymbol{\mu}_e$ are the external forces and moments applied to the end-effector, all expressed in $\mathcal{F}_v$. The considered balance constraint takes the form:

$$g_{\text{zmp}}(\boldsymbol{x}) = \rho_{sc}^2 - \|\tilde{\boldsymbol{p}}_{\text{zmp}}\|^2 \geq 0 \tag{6.29}$$

where ρ_{sc} is the radius of a circle enveloped by the support polygon. Since in [92] the NMPC is solved using the SLQ method, the balance constraint (6.29) was included in the cost function using a relaxed barrier function. However, for the purpose of this comparison we will use an NLP as the one proposed in (6.24) using (6.29) as balance constraint. To improve the robot balance we include in the running and terminal costs (6.22) and (6.23) the terms $w(\ln(g_{\text{zmp}}(\boldsymbol{x}_{k|i})/\rho_{sc}^2))^2$ and $w_N(\ln(g_{\text{zmp}}(\boldsymbol{x}_{k|N})/\rho_{sc}^2))^2$ where w and w_N are the associated weights.

In the simulation results we will refer to this method as NMPC-CG (CG stands for center of gravity).

QP using the STC The method in [103] uses a QP in order to solve the inverse kinematics problem given an end-effector task at the velocity level. For the purpose of this comparison, we slightly modified the QP in order to consider the robot dynamics. If $\boldsymbol{q}_{k+i}$, $\boldsymbol{\nu}_{k+i}$, $\boldsymbol{x}_{k+i}$ and $\boldsymbol{u}_{k+i}$ are the robot configuration, velocity, state and control inputs at time t_{k+i} respectively, then the QP to be solved at t_k is:

$$\begin{aligned}
&\min_{\boldsymbol{u}_k} \boldsymbol{\nu}_{k+1}^T \boldsymbol{Q} \boldsymbol{\nu}_{k+1} + \boldsymbol{v}_k^T \boldsymbol{S}_v \boldsymbol{v}_k + \boldsymbol{u}_k^T \boldsymbol{R}_u \boldsymbol{u}_k \\
&\text{subject to:} \\
&\dot{\boldsymbol{y}}_d(t_{k+1}) - \boldsymbol{J}(\boldsymbol{q}_{k+1}) \boldsymbol{\nu}_{k+1} = \boldsymbol{0} \\
&\boldsymbol{q}_{\min} \leq \boldsymbol{q}_{k+2} \leq \boldsymbol{q}_{\max} \\
&\boldsymbol{\nu}_{\min} \leq \boldsymbol{\nu}_{k+1} \leq \boldsymbol{\nu}_{\max} \\
&\boldsymbol{u}_{\min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{\max} \\
&\boldsymbol{A}_b(\boldsymbol{x}_k) \boldsymbol{u}_k \leq \boldsymbol{b}_b(\boldsymbol{x}_k)
\end{aligned}$$

where $\boldsymbol{q}_{k+1}$, $\boldsymbol{q}_{k+2}$ and $\boldsymbol{\nu}_{k+1}$ result from the integration of the equations of motion (6.16) using the Euler method and they are functions of $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$. The balance is maintained using (6.28) while the term $\boldsymbol{v}_k^T \boldsymbol{S}_v \boldsymbol{v}_k$ improves the robot balance with $\boldsymbol{v}_k = \boldsymbol{A}_b(\boldsymbol{x}_k)\boldsymbol{u}_k - \boldsymbol{b}_b(\boldsymbol{x}_k)$ and $\boldsymbol{S}_v$ the associated weight. Note that here we cannot apply collision avoidance constraints in the form of (6.25) and (6.26) since they are nonlinear functions of the state, while the QP can only accept constraints formulated as linear combinations of the control inputs. Of course, this does not mean that we cannot apply collision avoidance constraints of an appropriate form in the QP as it will be shown in the following chapter, however, this is out of the scope of this chapter.

We will refer to this method as QP-STC and for its solution we use `quadprog`.

6.7.2 Simulation results

We compare the three methods in five different scenarios of assigned end-effector tasks. For all methods the sampling time is $\delta = 23\,\text{ms}$ while the prediction horizon for the proposed method and the NMPC-CG is $T = 0.23\,\text{s}$, chosen as large as possible to allow real-time performance. In Fig. 6.7 we offer a series of snapshots from the simulations while at `https://youtu.be/xp3qVcyYww8` we offer the full video of the simulations. Note that the maximum iteration time for the proposed method was less than 23 ms in all considered scenarios, showing that it can perform in real-time.

Scenario 1: The assigned end-effector task is to track a linear path of length 1.45 m following a trapezoidal velocity profile. The task duration is 4 s and the maximum acceleration along the path is $0.44\,\text{m/s}^2$. The environment is obstacle-free. In this scenario, all three methods generate feasible motions (see Fig. 6.7 and video). However, Fig. 6.8 shows that the proposed method and the NMPC-CG (inseparable in the plot) give a smoother ZMP path than the QP-STC.

Scenario 2: The assigned end-effector task is the same of Scenario 1 only now the task duration is decreased to 1.2 s while the maximum acceleration is increased to $4.84\,\text{m/s}^2$. The environment is again obstacle-free. Fig. 6.7 (and the video) shows that only the methods that consider the full robot dynamics for the evaluation of the robot balance, i.e., the proposed method and the QP-STC, are able to generate

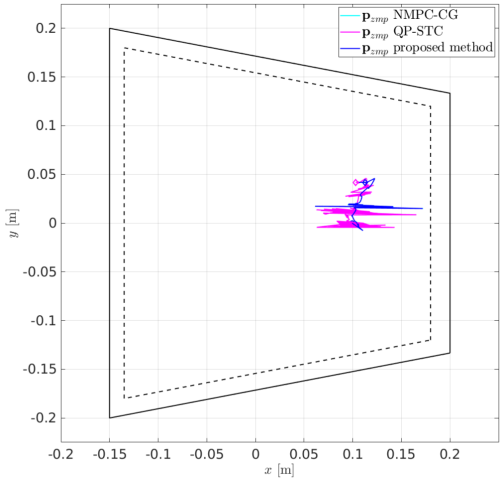

Figure 6.8. The ZMP position resulting in Scenario 1

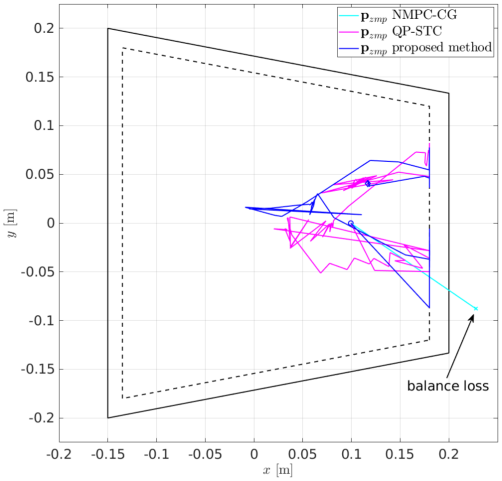

Figure 6.9. The ZMP position resulting in Scenario 2

feasible motions while the NMPC-CG fails as the rear wheels of the MM lose contact with the ground. It is apparent that, unlike Scenario 1, here the high acceleration required at the beginning of the robot motion makes the ZMP position approximation of the NMPC-CG inadequate and the robot loses balance immediately (see Fig. 6.9 for the ZMP position).

Scenario 3: The assigned end-effector task is the same as Scenario 1 but now the desired path is obstructed by a spherical obstacle of radius 0.1 m (see Fig. 6.7). The presence of the obstacle leads to an increase in the required acceleration at its vicinity (see Fig. 6.10). In Fig. 6.7 (and the video) we can see that while our method performs satisfactory, the NMPC-CG fails and the robot loses its balance when it is called to avoid the obstacle (see Fig. 6.10). Note that as we mentioned in Sect. 6.7.1, the QP-STC cannot be applied in this scenario since it does not support the collision avoidance constraints.

Scenario 4: The end-effector has to follow a circular path completing two full circles in 7 sec. The radius of the circle is 0.5 m. The environment is obstacle-free. In this scenario, only the proposed method and the QP-STC were able to complete the task, while in Fig. 6.7 (and the video) we can see the NMPC-CG losing balance as the rear wheels of the robot detach from the ground at the beginning of the second circle (see also Fig. 6.11 for the ZMP position). In Fig. 6.11 we can also see that the proposed method generates motions that result in a smoother ZMP path than the QP-STC.

Scenario 5: The end-effector has to follow a linear path of length 0.7 m followed by a second linear path of length 0.7 m that forms an angle of $\pi/4$ rad with the first one (see Fig. 6.7). The desired velocity has again a trapezoidal profile. The task duration is 4 s and the maximum acceleration along the path is 0.44 m/s^2. The environment is obstacle-free. In Fig. 6.7 (and the video) we can see that due to the abrupt change of direction, that leads to increased required accelerations (see Fig. 6.12), only the proposed method is able to generate feasible motions, while the rest lose balance.

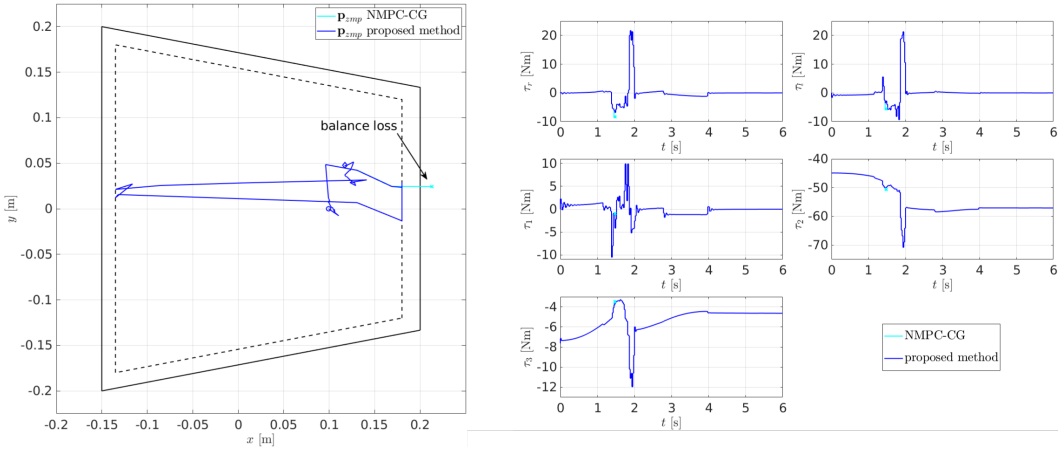

Figure 6.10. The ZMP position (left) and the control inputs (right) resulting in Scenario 3

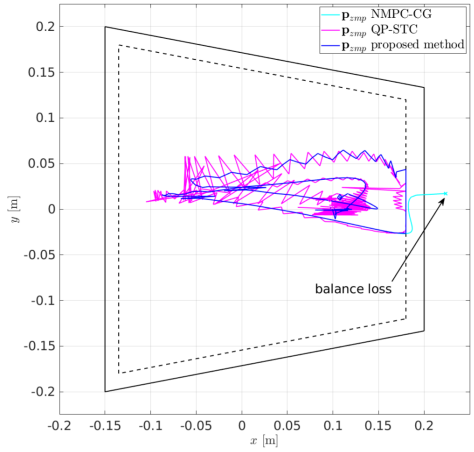

Figure 6.11. The ZMP position resulting in Scenario 4

6.8 Conclusions

In this chapter, we presented a novel real-time motion generation approach for MMs using an NMPC. Kinodynamic feasibility is enforced through the use of the robot dynamic model as prediction model and via input and state constraints, while collisions are avoided using appropriate collision and self-collision avoidance constraints. To prevent the robot loss of balance in cases where it is called to execute aggressive motions we included a constraint that restricts the robot feasible motions to those that result in non-negative moments around the support polygon edges. To enable the solution of the proposed NMPC scheme with off-the-shelf solvers that require symbolic representation of the OCP, like `ACADO Toolkit`, we lifted the inherent nonlinearity of the balance constraint by linearizing it using the solution of the previous iteration of the NMPC.

The proposed method was compared with two other methods in five scenarios. The simulation results showed that the proposed method can effectively handle end-effector tasks that require aggressive motions even in cases where the other methods fail.

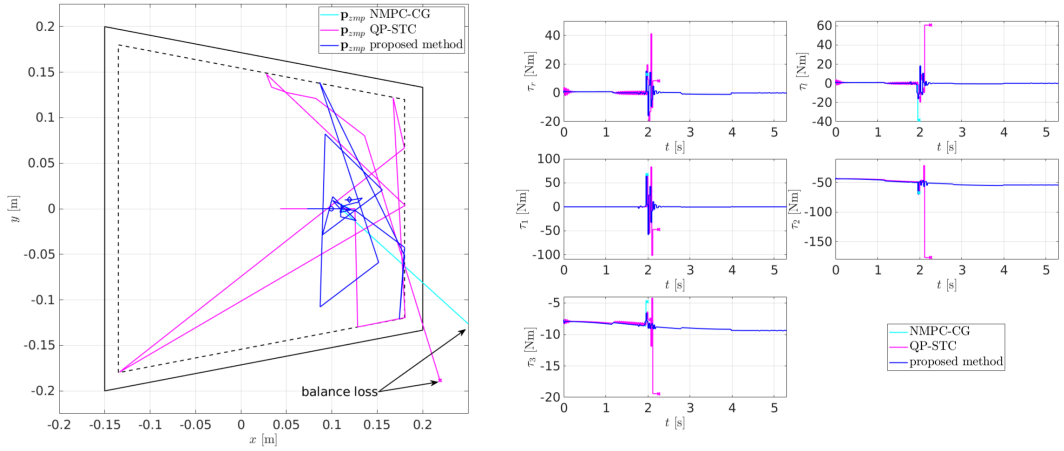

Figure 6.12. The ZMP position (left) and the control inputs (right) resulting in Scenario 5

Future work aims at

- the experimental validation of the proposed method;
- the extension of the proposed method to consider additional aspects of the robot dynamics (e.g., the effect of the wheel-ground friction).

Chapter 7

An optimization-based controller for enforcing safety constraints in mobile manipulators

Consider the case in which a simplified model of the robot is used for motion generation in an attempt to reduce the required computational time. In this chapter, we propose an Optimization-Based Controller (OBC) that can be interposed between a motion generation module that considers a simplified model of the robot and the robot itself in order to enforce safety.

7.1 Introduction

In Chap. 6 we were called to solve the problem of real-time motion generation for a MM called to execute a desired task in an environment populated by obstacles, possibly requiring aggressive motions. For the generation of such a motion, of particular importance is the considered prediction model of the robot. Its kinematic model is essential as it ensures that the resulting motion will be compatible with the assigned task and the geometric constraints imposed by the environment, i.e., the collision avoidance constraints. Nevertheless, in order to ensure that the resulting motion will be kinodynamically feasible and to properly reason about the robot balance, one has to consider also the robot full dynamics, as we did in Chap. 6. However, one can notice that the robot considered in Chap. 6 was relatively simple, as it was equipped with a 3-DOF manipulator. Recently, MMs equipped with high DOF manipulators and MM with multi-arm configurations have become rather popular. Of great interest are also the legged mobile manipulators which in principle have a rather complex structure. In cases like these, the dynamic model of the robot can end up being rather complicated. Apart from the memory overhead that may emerge from the use of such a model (an issue that we also discussed in Chap. 6), it may also lead to unacceptably large computational times which is incompatible with the real-time requirement. As a result, approaches that use the simplest possible prediction model of the robot, without compromising safety are preferable.

In order to efficiently plan motions for complex robots the authors in [108] suggest the use of simplified dynamic models, specifically the robot centroidal dynamics [109],

in combination with the robot kinematics. The robot centroidal dynamics model is simpler than the full dynamics model as the DOF of the robot increase and it provides a relation between the robot motion and the external generalized forces applied to the robot, making it ideal for reasoning about the robot balance. However, in contrast to the robot full dynamics, the robot centroidal dynamics do not consider the generalized forces exerted by the motors of the robot. So, if the robot motors do not have enough control authority, the resulting motion might not be implementable. As a result, a controller that considers the robot full dynamics should be interposed between the motion planning module and the robot in order to transform the planned motion to appropriate torque commands for the robot. In the case of [108] the OBC presented in [110] was used for this purpose.

A similar approach is used within a real-time NMPC framework in [111] and applied to a legged mobile manipulator. The prediction model uses the robot centroidal dynamics and the resulting motion is tracked by an OBC [112] that considers the robot full dynamics in order to generate appropriate torque commands. Although in this particular work the robot is not protected from collisions with its environment, its extension, presented in [113], includes collision avoidance constraints as soft constraints in the NMPC using an appropriate barrier function.

However, the use of a controller that considers the robot full dynamics in order to track a collision-free but not necessarily kinodynamically feasible whole-body motion can eventually lead to tracking errors, which can potentially jeopardize the robot safety. To illustrate this point, let us consider the case in which a motion generation module, that does not consider the robot full dynamics, generates an aggressive motion with high accelerations. If the robot does not have the actuation capabilities to generate those accelerations at the current state of the robot, then the controller that is called to track this motion and considers the robot full dynamics will generate control commands that lead to a tracking error. Although this tracking error might be negligible, if the robot moves in the vicinity of an obstacle, a collision can occur.

This issue can be avoided if the controller considers appropriate collision avoidance constraints. In [114] the authors suggest the use of the *Velocity Damper* [115] method in order to include collision avoidance constraints in a QP. However, recently CBFs [64] have been effectively exploited for this purpose. For robot manipulators, representative examples where CBF are used as collision avoidance constraints between the robot and its environment can be found in [116, 117, 118]. However, in their implementation [116] and [118] consider collision avoidance constraint only for the robot end-effector, ignoring the rest of the robot, while [117] although it considers the whole robot for collision it uses only its kinematic model.

In this chapter, we will solve the motion generation problem presented in Sect. 6.2 using a motion generation scheme comprised of a motion generation module that uses a simplified model of the robot and an OBC that is used in order to track the generated motion and apply appropriate torque commands to the considered robot. The proposed OBC, which is interposed between the motion generation module and the robot, is implemented as a QP and considers the robot full dynamics, joint limits, velocity limits and torque bounds. To ensure that the resulting motion will be at least collision-free, the controller considers collision and self-collision avoidance constraints built upon appropriate CBFs. The controller is complete with a balance

constraint that restricts the robot motions to those generating non-negative moments around the support polygon edges. Simulation results indicate the effectiveness of the proposed OBC, even when the motion generation module ignores completely the robot dynamics, illustrating also the importance of including the collision avoidance constraint in the tracking controller.

To the best of our knowledge, an OBC like the one presented in this chapter has not been applied to mobile manipulators.

This chapter is organized as follows. In Sect. 7.2 we restate the motion generation problem presented in Sect. 6.2 for completeness and in Sect. 7.3 we describe the proposed motion generation approach. Sect. 7.4 presents in detail the motion generation module, while Sect. 7.5 presents the proposed OBC, which is the main focus of this chapter. Simulation results are presented in Sect. 7.6, followed by a discussion on the proposed method in Sect. 7.7. Finally, concluding remarks are offered in Sect. 7.8.

7.2 Problem formulation

Consider a general wheeled mobile manipulator with configuration $\boldsymbol{q}$. The robot operates in a 3-dimensional workspace $\mathcal{W}$, moving on an horizontal ground. The considered workspace is populated by static and/or moving obstacles. Denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$ and by $\mathcal{O}(t) \subset \mathcal{W}$ the volume occupied by the obstacles at time t.

The robot is called to execute a task in terms of a vector $\boldsymbol{y} \in \mathcal{Y}$ which describes the position of the end-effector and is assigned as a desired end-effector trajectory $\boldsymbol{y}_d(t)$ with defined derivatives and $t \in [0, t_f]$ where t_f the duration of the assigned task.

The problem consists in generating in real-time a motion that:

R1: starts from the robot initial configuration and follows as close as possible the desired end-effector trajectory $\boldsymbol{y}_d$;

R2: is kinodynamically feasible, in the sense that it is consistent with the robot dynamics and respects joint and velocity limits and the joint torque bounds;

R3: maintains the robot balance, in the sense that the robot wheels are always in contact with the ground;

R4: is collision-free, avoiding not only collisions with the environment obstacles, but also self-collisions.

We assume that the robot is always aware of its own state as well as the position and velocity of the obstacles populating its environment.

7.3 Proposed approach

To solve the problem at hand we use the motion generation scheme illustrated in Fig. 7.1. It consists of two modules; a *motion generation module* that generates

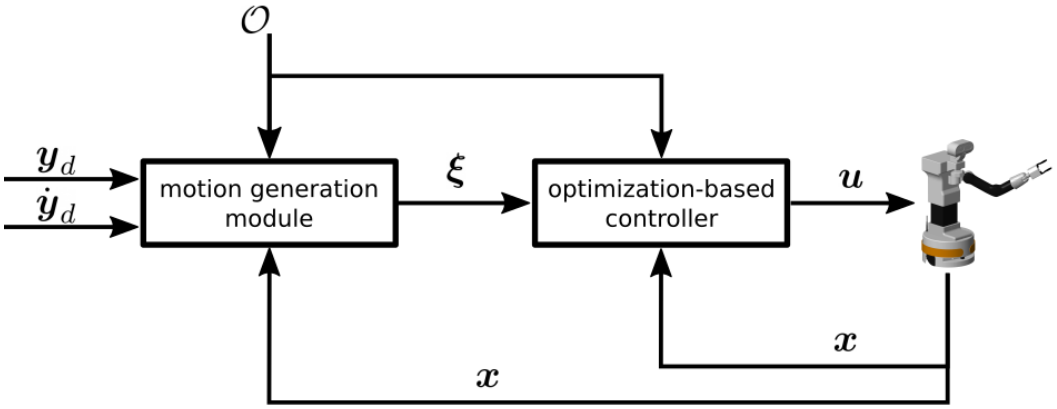

Figure 7.1. The block diagram of the proposed motion generation scheme. It consists of a motion generation module that plans whole-body motions for the robot, using a simplified model of it. Those motions are used as a reference from the optimization-based controller in order to generate appropriate torque commands for the robot while enforcing safety.

collision-free whole-body reference motions for the robot consistent with the assigned end-effector task, and an *optimization-based controller* that generates torque commands for the robot joints in order to follow the reference motion.

The *motion generation module*, which in this work is implemented as a real-time NMPC, uses as prediction model a simplified model of the robot in order to generate whole-body motions that are at least consistent with the assigned end-effector task and account for the geometric constraints imposed by the robot structure, i.e., self-collision avoidance constraints, and its interaction with the environment, i.e., collision avoidance constraints with the environment obstacles. The motion generation module will be presented in detail in Sect. 7.4.

Since the resulting motion is not necessarily kinodynamically feasible, as the motion generation module does not consider the robot full dynamics, the *optimization-based controller* (OBC) is interposed between the motion generation module and the robot. The OBC, working at a higher frequency than the motion generation module, receives as a reference the generated motion and tries to follow it as close as possible while accounting for the robot full dynamics as well as joint limits, velocity limits and torque bounds (requirement R2). The proposed OBC considers also balance constraints (requirement R3) as well as collision avoidance constraints (requirement R4) built upon appropriate CBFs. In this way, it ensures that even if the robot cannot follow precisely the generated motion, the resulting robot motion will be at least safe. The OBC, which is the main contribution of this work, will be presented in detail in Sect. 7.5.

7.4 Motion generation module

As it was mentioned above we will use an NMPC in order to generate whole-body motions that are at least consistent with the assigned end-effector task, as well as with the geometry of the robot and its environment, in terms of collision avoidance.

To do so, it is sufficient to consider the robot full kinematics. The prediction model that will be used is

$$\dot{\boldsymbol{x}} = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{\upsilon}) = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \\ \boldsymbol{\upsilon} \end{pmatrix} \tag{7.1}$$

where the robot state is denoted by $\boldsymbol{x} = (\boldsymbol{q}_e, \boldsymbol{\nu})$, while $\boldsymbol{\upsilon} = \dot{\boldsymbol{\nu}}$ is the robot (pseudo-) acceleration and correspond to the control inputs of the considered model. Note that we could have used a combination of the robot full kinematics with any simplified version of the robot dynamics (e.g., the robot centroidal dynamics). However, we decided to use a model that does not consider the robot dynamics at all, in order to highlight the effectiveness of the proposed OBC, which is the main contribution of this work.

Denote by T the prediction horizon, by δ_{NMPC} the sampling time and by $N = T/\delta_{\text{NMPC}}$ the number of the control intervals in the prediction horizon. Consider the NLP to be solved at the discrete time instant $t_\kappa = \kappa\delta_{\text{NMPC}}$. Denote by $\boldsymbol{\upsilon}_{\kappa|i}$ and $\boldsymbol{x}_{\kappa|i}$ the predicted acceleration and state of the robot at the time instant $t_{\kappa+i}$.

The objective is to minimize the task error while guaranteeing collision avoidance between the robot and its environment as well as self-collision avoidance, ideally with the minimum acceleration. Denote the predicted task error at time $t_{\kappa|i}$ by $\boldsymbol{e}_{\kappa|i} = \boldsymbol{y}_d(t_{\kappa+i}) - \boldsymbol{y}_{\kappa|i}$ and its derivative by $\dot{\boldsymbol{e}}_{\kappa|i} = \dot{\boldsymbol{y}}_d(t_{\kappa+i}) - \dot{\boldsymbol{y}}_{\kappa|i}$, where $\boldsymbol{y}_{\kappa|i}$ and $\dot{\boldsymbol{y}}_{\kappa|i}$ are the predicted end-effector position and velocity at time $t_{\kappa|i}$. The running and terminal costs are

$$V_{\kappa|i}(\boldsymbol{x}_{\kappa|i}, \boldsymbol{\upsilon}_{\kappa|i}) = \boldsymbol{e}_{\kappa|i}^T \boldsymbol{Q}_p \boldsymbol{e}_{\kappa|i} + \dot{\boldsymbol{e}}_{\kappa|i}^T \boldsymbol{Q}_d \dot{\boldsymbol{e}}_{\kappa|i} + \boldsymbol{\nu}_{\kappa|i}^T \boldsymbol{S} \boldsymbol{\nu}_{\kappa|i} + \boldsymbol{\upsilon}_{\kappa|i}^T \boldsymbol{R}_\upsilon \boldsymbol{\upsilon}_{\kappa|i}$$

$$V_{\kappa|N}(\boldsymbol{x}_{\kappa|N}) = \boldsymbol{e}_{\kappa|N}^T \boldsymbol{Q}_{p,N} \boldsymbol{e}_{\kappa|N} + \dot{\boldsymbol{e}}_{\kappa|N}^T \boldsymbol{Q}_{d,N} \dot{\boldsymbol{e}}_{\kappa|N} + \boldsymbol{\nu}_{\kappa|N}^T \boldsymbol{S}_N \boldsymbol{\nu}_{\kappa|N}.$$

The running cost consists of terms that, in the order of appearance, penalize the task error, its derivative, the robot velocity and acceleration throughout the prediction horizon, with $\boldsymbol{Q}_p$, $\boldsymbol{Q}_d$, $\boldsymbol{S}$ and $\boldsymbol{R}_\upsilon$ the associated weighting matrices. The terminal cost consists of terms that penalize the task error, its derivative and the robot velocity at the final time instant, with $\boldsymbol{Q}_{p,N}$, $\boldsymbol{Q}_{d,N}$ and $\boldsymbol{S}_N$ the associated weighting matrices.

The NLP to be solved at time t_κ is

$$\min_{\substack{\boldsymbol{x}_{\kappa|0},\dots,\boldsymbol{x}_{\kappa|N} \\ \boldsymbol{\upsilon}_{\kappa|0},\dots,\boldsymbol{\upsilon}_{\kappa|N-1}}} \sum_{i=0}^{N-1} V_{\kappa|i}(\boldsymbol{x}_{\kappa|i}, \boldsymbol{\upsilon}_{\kappa|i}) + V_{\kappa|N}(\boldsymbol{x}_{\kappa|N}) \tag{7.2a}$$

subject to:

$$\boldsymbol{x}_{\kappa|0} - \boldsymbol{x}_\kappa = \boldsymbol{0} \tag{7.2b}$$

$$\boldsymbol{x}_{\kappa|i+1} - \boldsymbol{\phi}_{\text{d-t}}(\boldsymbol{x}_{\kappa|i}, \boldsymbol{\upsilon}_{\kappa|i}) = \boldsymbol{0}, \quad i = 0, \dots, N-1 \tag{7.2c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_{\kappa|i} \leq \boldsymbol{x}_{\max}, \quad i = 0, \dots, N \tag{7.2d}$$

$$\boldsymbol{\upsilon}_{\min} \leq \boldsymbol{\upsilon}_{\kappa|i} \leq \boldsymbol{\upsilon}_{\max}, \quad i = 0, \dots, N-1 \tag{7.2e}$$

$$\textit{collision avoidance constraints at } t_\kappa, \dots, t_{\kappa+N} \tag{7.2f}$$

$$\textit{self-collision avoidance constraints at } t_\kappa, \dots, t_{\kappa+N} \tag{7.2g}$$

where $\boldsymbol{x}_\kappa$ represents the state of the robot at time t_κ, $\boldsymbol{\phi}_{\text{d-t}}(\cdot,\cdot)$ represents the prediction model (7.1) in discrete-time obtained via numerical integration under the

assumption of piecewise-constant robot accelerations, while $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$ and $\boldsymbol{\upsilon}_{\min}$, $\boldsymbol{\upsilon}_{\max}$ are respectively the lower/upper bounds on the state variables and on the robot acceleration.

Collision and self-collision avoidance constraints: For the collision and self-collision avoidance constraints we will use the same approach followed in Sect. 6.5. We briefly remind that for the collision avoidance constraints we envelop the j-th robot body with an ellipsoid and l-th obstacle with a sphere. To ensure that the two bodies do not collide we use the inequality constraint

$$h_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l) = (\boldsymbol{r}_j - \boldsymbol{o}_l)^T \boldsymbol{R}_{r_j} \boldsymbol{H}_{j,l} \boldsymbol{R}_{r_j}^T (\boldsymbol{r}_j - \boldsymbol{o}_l) - 1 \geq 0 \tag{7.3}$$

with

$$\boldsymbol{H}_{j,l} = \text{diag}\{(\alpha_j + \rho_l + d_c)^{-2}, (\beta_j + \rho_l + d_c)^{-2}, (\gamma_j + \rho_l + d_c)^{-2}\}.$$

For more details, refer to Sect. 6.5. The collision avoidance constraints to be applied at (7.2f) can be written as

$$h_{j,l}(\boldsymbol{x}_{\kappa|i}, \boldsymbol{o}_{l,\kappa|i}) \geq 0, \quad i = 0, ..., N, \quad j = 1, ..., n_b, \quad l = 1, ..., n_o$$

where $\boldsymbol{o}_{l,\kappa|i}$ is the predicted position of the the l-th obstacle at time instant $t_{\kappa+i}$.

Similarly, for the self-collision avoidance constraints, given a pair of susceptible to collision bodies, we envelop the j-th robot body with the ellipsoid and the l-th robot body with the sphere. To ensure that the two bodies do not collide we use the inequality constraint

$$h^s_{j,l}(\boldsymbol{x}) = (\boldsymbol{r}_j - \boldsymbol{r}_l)^T \boldsymbol{R}_{r_j} \boldsymbol{H}^s_{j,l} \boldsymbol{R}^T_{r_j} (\boldsymbol{r}_j - \boldsymbol{r}_l) - 1 \geq 0, \quad (j,l) \in \mathcal{S} \tag{7.4}$$

with

$$\boldsymbol{H}^s_{j,l} = \text{diag}\{(\alpha_j + \rho_l + d_s)^{-2}, (\beta_j + \rho_l + d_s)^{-2}, (\gamma_j + \rho_l + d_s)^{-2}\}.$$

Again, for more details refer to Sect. 6.5. The self-collision avoidance constraints to be applied at (7.2g) can be written as

$$h^s_{j,l}(\boldsymbol{x}_{\kappa|i}) \geq 0, \quad i = 0, ..., N, \quad (j,l) \in \mathcal{S}$$

with $\mathcal{S}$ the set of pairs of indices (j,l) that correspond to robot bodies that are susceptible to collision.

7.5 Optimization-based controller

This section presents in detail the OBC, which is the main contribution of this work. The OBC is implemented as a QP. At each discrete time instant $t_k = k\delta_{\text{QP}}$, with δ_{QP} being the considered sampling time, the OBC receives from the motion generation module a whole-body reference motion in terms of configuration, velocity and acceleration $\boldsymbol{\xi}_k = (\boldsymbol{q}^r_{e,k}, \boldsymbol{\nu}^r_k, \dot{\boldsymbol{\nu}}^r_k)$ and based on the measurement for the current state of the robot $\boldsymbol{x}_k = (\boldsymbol{q}_{e,k}, \boldsymbol{\nu}_k)$, generates appropriate torque commands $\boldsymbol{u}_k$ for the robot (see Fig.7.1).

The elements of the proposed optimization-based controller are presented in the following.

7.5.1 Cost function

The ultimate goal of the proposed motion generation approach is to generate motions that track the desired end-effector trajectory as close as possible (requirement R1). So, it is sufficient for the OBC to follow the reference $\boldsymbol{\xi}_k$, which is consistent with the end-effector task, in order to satisfy R1.

However, it is clear that if the constraints related to requirements R2-R4 are active, the optimization-based controller will not be able to follow the reference motion. In this case, we want at least to follow the end-effector motion that corresponds to $\boldsymbol{\xi}_k$, i.e., $(\boldsymbol{y}_k^r, \dot{\boldsymbol{y}}_k^r, \ddot{\boldsymbol{y}}_k^r)$ which is obtained by substituting $\boldsymbol{\xi}_k$ in (6.11), (6.12) and (6.13). So, the considered cost function is

$$C_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \left\| \ddot{\boldsymbol{q}}_{e,k} - \boldsymbol{\zeta}_k \right\|_{\boldsymbol{Q}}^2 + \| \ddot{\boldsymbol{y}}_k - \boldsymbol{\eta}_k \|_{\boldsymbol{Y}}^2 . \tag{7.5}$$

The first term of (7.5) minimizes the deviation of the robot acceleration at the generalized coordinates level, $\ddot{\boldsymbol{q}}_{e,k}$, from a reference acceleration $\boldsymbol{\zeta}_k$ defined as a Proportional-Derivative (PD) control rule with a feed-forward term

$$\boldsymbol{\zeta}_k = \ddot{\boldsymbol{q}}_{e,k}^r + \boldsymbol{K}_D^{\zeta}(\dot{\boldsymbol{q}}_{e,k}^r - \dot{\boldsymbol{q}}_{e,k}) + \boldsymbol{K}_P^{\zeta}(\boldsymbol{q}_{e,k}^r - \boldsymbol{q}_{e,k}),$$

with $\dot{\boldsymbol{q}}_{e,k}^r$ and $\ddot{\boldsymbol{q}}_{e,k}^r$ obtained by substituting $\boldsymbol{\xi}_k$ in (6.9) and (6.10), while $\dot{\boldsymbol{q}}_{e,k}$ obtained by substituting $\boldsymbol{\nu}_k$ in (6.9). $\boldsymbol{K}_D^{\zeta}$ and $\boldsymbol{K}_P^{\zeta}$ are gain matrices of appropriate dimensions.

The second term of (7.5) minimizes the deviation of the end-effector acceleration, $\ddot{\boldsymbol{y}}_k$, from a reference $\boldsymbol{\eta}_k$ defined also as PD control rule with a feed-forward term

$$\boldsymbol{\eta}_k = \ddot{\boldsymbol{y}}_k^r + \boldsymbol{K}_D^{\eta}(\dot{\boldsymbol{y}}_k^r - \dot{\boldsymbol{y}}_k) + \boldsymbol{K}_P^{\eta}(\boldsymbol{y}_k^r - \boldsymbol{y}_k)$$

where $\boldsymbol{y}_k$ and $\dot{\boldsymbol{y}}_k$ are obtained by substituting $\boldsymbol{x}_k$ in (6.11) and (6.12). $\boldsymbol{K}_D^{\eta}$ and $\boldsymbol{K}_P^{\eta}$ are gain matrices of appropriate dimensions. $\boldsymbol{Q}$ and $\boldsymbol{Y}$ are the weighting matrices associated with the two terms of the cost function.

7.5.2 State and input bounds

At each time instant t_k the control inputs have to remain within the lower and upper bounds $\boldsymbol{u}_{\min}$ and $\boldsymbol{u}_{\max}$ that are dictated by the hardware limitations of the robot

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{\max}. \tag{7.6}$$

Moreover, the control inputs at time t_k affect the velocity and the configuration of the robot at time instant t_{k+1}, namely $\boldsymbol{\nu}_{k+1}$ and $\boldsymbol{q}_{k+1}$, which have to remain within the considered velocity and joint bounds

$$\boldsymbol{\nu}_{\min} \leq \boldsymbol{\nu}_k + \dot{\boldsymbol{\nu}}_k \delta_{\mathrm{QP}} \leq \boldsymbol{\nu}_{\max} \tag{7.7}$$

$$\boldsymbol{q}_{\min} \leq \boldsymbol{q}_k + \dot{\boldsymbol{q}}_k \delta_{\mathrm{QP}} + \frac{1}{2} \ddot{\boldsymbol{q}}_k \delta_{\mathrm{QP}}^2 \leq \boldsymbol{q}_{\max} \tag{7.8}$$

where $\boldsymbol{\nu}_{k+1}$ and $\boldsymbol{q}_{k+1}$ are approximated by a first and second order Taylor series, respectively, and $\boldsymbol{\nu}_{\min}$, $\boldsymbol{\nu}_{\max}$ and $\boldsymbol{q}_{\min}$, $\boldsymbol{q}_{\max}$ are the lower/upper bounds of the robot velocity and joint position. Using (6.9), (6.10) and (6.16) the constraints (7.7) and

(7.8) can be expressed as a function of the robot state and control inputs at t_k, and after some manipulation the constraints (7.6), (7.7) and (7.8) can be expressed in the form

$$\boldsymbol{A}_c(\boldsymbol{x}_k)\boldsymbol{u}_k \leq \boldsymbol{b}_c(\boldsymbol{x}_k)$$

where the matrix $\boldsymbol{A}_c$ and the vector $\boldsymbol{b}_c$ can be easily derived.

7.5.3 Balance constraint

To maintain the robot balance we will use the constraint (6.27), that practically restricts the moments exerted by the robot around the support polygon edges to remain non-negative. After simple manipulation (see Sect. 6.6) the constraint can be expressed in the form

$$\boldsymbol{A}_b(\boldsymbol{x})\boldsymbol{u} \leq \boldsymbol{b}_b(\boldsymbol{x}). \tag{7.9}$$

7.5.4 Collision avoidance constraints

Although the reference motion is collision-free, it is not necessarily kinodynamically feasible and so exact tracking cannot be guaranteed. To ensure that any inability of the robot to track the reference motion will at least lead to a collision-free motion, we include appropriate collision avoidance constraints in the OBC. Since (7.3) and (7.4) cannot be applied directly in a QP as they are nonlinear functions of the robot configuration, we will build a constraint using an appropriate CBF.

Regarding the collision avoidance constraints between the robot and the environment obstacles. Let us consider the j-th robot body, the l-th obstacle and the set comprised by the robot states that satisfy (7.3), i.e.,

$$\mathcal{C}_{j,l} = \{\boldsymbol{x} \in \mathcal{X} : h_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l) \geq 0\}.$$

From now on we will refer to the $\mathcal{C}_{j,l}$ as *safe-set*.

Note that collision avoidance between the j-th robot body and the l-th obstacle is enforced, if the safe-set $\mathcal{C}_{j,l}$ is forward-invariant. CBFs are an appropriate tool for attaining forward-invariance of a set and so we intend to find such a CBF. For this purpose, we need a function that for the considered dynamic system (6.16) its relative degree is 1. Since the function $h_{j,l}(\cdot,\cdot)$ has relative degree 2, as it requires its second derivative for the control inputs to appear, we can use the method of *exponential* CBFs [119]. Taking inspiration from the procedure followed in [120], we define the auxiliary function

$$\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l) = \dot{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l) + a_{j,l} h_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l)$$

with $a_{j,l} > 0$. Note that if moving obstacles are considered, this auxiliary function depends also on their velocity. This function can be used as an appropriate CBF, while by enforcing the constraint

$$\dot{\tilde{h}}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l) \geq -b_{j,l}\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l) \tag{7.10}$$

with $b_{j,l} > 0$, the forward-invariance of the safe-set $\mathcal{C}_{j,l}$ is implied. The constraint (7.10) can be rewritten as

$$\frac{\partial}{\partial \boldsymbol{x}}\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l)\dot{\boldsymbol{x}} + \frac{\partial}{\partial t}\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l) \geq -b_{j,l}\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l). \tag{7.11}$$

Before we proceed, let us rewrite the state-space reduced model of the robot (6.16) in the convenient for this analysis form:

$$\dot{\boldsymbol{x}} = \underbrace{\begin{pmatrix} \boldsymbol{G}(\boldsymbol{q})\boldsymbol{\nu} \\ -\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{m}(\boldsymbol{q},\boldsymbol{\nu}) \end{pmatrix}}_{\boldsymbol{f}(\boldsymbol{x})} + \underbrace{\begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{E}(\boldsymbol{q}) \end{pmatrix}}_{\boldsymbol{C}(\boldsymbol{x})} \boldsymbol{u} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{C}(\boldsymbol{x})\boldsymbol{u} \tag{7.12}$$

By substituting (7.12) in (7.11) we get the constraint to be included in the OBC

$$-\mathcal{L}_{\boldsymbol{C}}\tilde{h}_{j,l}(\boldsymbol{x},\boldsymbol{o}_l,\dot{\boldsymbol{o}}_l)\boldsymbol{u} \leq b_{j,l}\tilde{h}_{j,l}(\boldsymbol{x},\boldsymbol{o}_l,\dot{\boldsymbol{o}}_l) + \mathcal{L}_{\boldsymbol{f}}\tilde{h}_{j,l}(\boldsymbol{x},\boldsymbol{o}_l,\dot{\boldsymbol{o}}_l) + \frac{\partial}{\partial t}\tilde{h}_{j,l}(\boldsymbol{x},\boldsymbol{o}_l,\dot{\boldsymbol{o}}_l) \tag{7.13}$$

with $j = 1, ..., n_b$ and $l = 1, ..., n_o$.

Similarly, for the self-collision avoidance constraints, consider the j-th and l-th robot bodies with $(j,l) \in \mathcal{S}$. The safe-set comprised by the robot states that satisfy (7.4) is defined as

$$\mathcal{C}^s_{j,l} = \{\boldsymbol{x} \in \mathcal{X} : h^s_{j,l}(\boldsymbol{x}) \geq 0\}.$$

To ensure self-collision avoidance we want to attain the safe-set $\mathcal{C}^s_{j,l}$ forward-invariant.

Again, we define the auxiliary function

$$\tilde{h}^s_{j,l}(\boldsymbol{x}) = \dot{h}^s_{j,l}(\boldsymbol{x}) + a^s_{j,l}h^s_{j,l}(\boldsymbol{x})$$

with $a^s_{j,l} > 0$, that can be used as an appropriate CBF. Enforcing the constraint

$$\dot{\tilde{h}}^s_{j,l}(\boldsymbol{x}) \geq -b^s_{j,l}\tilde{h}^s_{j,l}(\boldsymbol{x}) \tag{7.14}$$

with $b^s_{j,l} > 0$, the forward-invariance of safe-set $\mathcal{C}^s_{j,l}$ is implied. By substituting (7.12) in (7.14) we get the constraint to be included in the OBC

$$-\mathcal{L}_{\boldsymbol{C}}\tilde{h}^s_{j,l}(\boldsymbol{x})\boldsymbol{u} \leq b^s_{j,l}\tilde{h}^s_{j,l}(\boldsymbol{x}) + \mathcal{L}_{\boldsymbol{f}}\tilde{h}^s_{j,l}(\boldsymbol{x}) \tag{7.15}$$

with $(j,l) \in \mathcal{S}$.

7.5.5 Optimization scheme

The OBC is implemented as a QP. Given the robot state $\boldsymbol{x}_k$ at time instant t_k, the QP to be solved is

$$\begin{aligned}
&\min_{\boldsymbol{u}_k} \left\| \ddot{\boldsymbol{q}}_{e,k} - \boldsymbol{\zeta}_k \right\|^2_{\boldsymbol{Q}} + \|\ddot{\boldsymbol{y}}^r_k - \boldsymbol{\eta}_k\|^2_{\boldsymbol{Y}} \\
&\text{subject to:} \\
&\boldsymbol{A}_c(\boldsymbol{x}_k)\boldsymbol{u}_k \leq \boldsymbol{b}_c(\boldsymbol{x}_k) \\
&\boldsymbol{A}_b(\boldsymbol{x}_k)\boldsymbol{u}_k \leq \boldsymbol{b}_b(\boldsymbol{x}_k) \\
&-\mathcal{L}_{\boldsymbol{C}}\tilde{h}_{j,l}(\boldsymbol{x}_k,\boldsymbol{o}_{l,k},\dot{\boldsymbol{o}}_{l,k})\boldsymbol{u}_k \leq b_{j,l}\tilde{h}_{j,l}(\boldsymbol{x}_k,\boldsymbol{o}_{l,k},\dot{\boldsymbol{o}}_{l,k}) \\
&\qquad + \mathcal{L}_{\boldsymbol{f}}\tilde{h}_{j,l}(\boldsymbol{x}_k,\boldsymbol{o}_{l,k},\dot{\boldsymbol{o}}_{l,k}) + \frac{\partial}{\partial t}\tilde{h}_{j,l}(\boldsymbol{x}_k,\boldsymbol{o}_{l,k},\dot{\boldsymbol{o}}_{l,k}), \quad j = 1, ..., n_b,\ l = 1, ..., n_o \\
&-\mathcal{L}_{\boldsymbol{C}}\tilde{h}^s_{j,l}(\boldsymbol{x}_k)\boldsymbol{u}_k \leq b^s_{j,l}\tilde{h}^s_{j,l}(\boldsymbol{x}_k) + \mathcal{L}_{\boldsymbol{f}}\tilde{h}^s_{j,l}(\boldsymbol{x}_k), \quad (j,l) \in \mathcal{S}
\end{aligned}$$

where $\boldsymbol{o}_{l,k}$ and $\dot{\boldsymbol{o}}_{l,k}$ are the position and velocity of the l-th obstacle at time t_k.

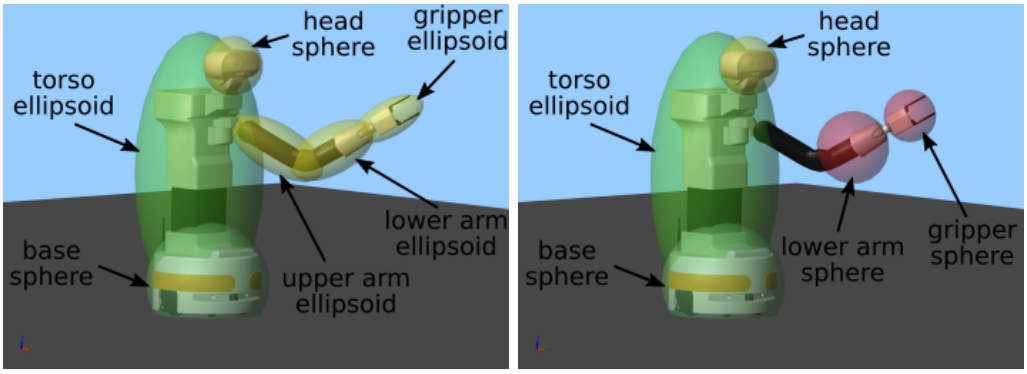

Figure 7.2. The bounding ellipsoids and spheres considered for the collision avoidance constraints for the TIAGo MM. The left figure shows the bounding ellipsoids and spheres (ellipsoids with semi-axes of equal size) considered for the collision avoidance constraints with the environment. The right figure shows the ellipsoids and spheres for the self-collision avoidance constraint. Note that the gripper and the lower arm are enveloped with red spheres and in the constraints are treated as if they were obstacles.

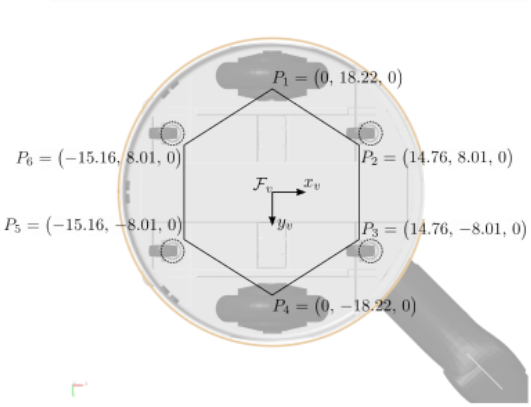

Figure 7.3. The considered support polygon for the TIAGo MM. With dashed circles we denote the admissible path of the caster wheels. The coordinates of the points are in cm.

7.6 Simulations

The proposed motion generation scheme has been validated for the TIAGo mobile manipulator via dynamic simulations in Simscape within Simulink on an Intel Core i9-9900K CPU running at 3.60GHz. In the simulations we control the robot at the joint level sending torque/force commands. The NMPC was solved using the RTI scheme [107] within `ACADO Toolkit` [17]. The sampling time used for the NMPC was $\delta_{\text{NMPC}} = 30\text{ms}$ and the prediction horizon $T = 0.36\text{s}$.

The proposed OBC was also set up in MATLAB using sampling time $\delta_{\text{QP}} = 3\text{ms}$. The dynamic model of TIAGo required for the OBC was extracted from its URDF[1] using the `Robotics Systems Toolbox`, while for the solution of the resulting QP we used `quadprog` with an active-set algorithm. The performance of the OBC in terms of computational time was boosted using the `MATLAB Coder`.

[1] `https://github.com/pal-robotics/tiago_robot`

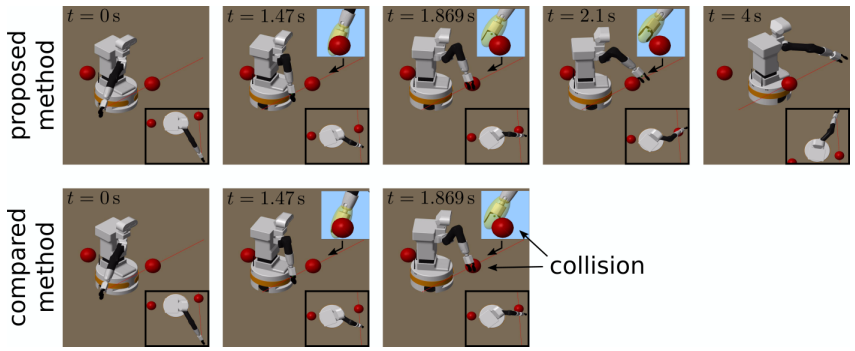

Figure 7.4. Snapshots of the motion generated during the simulation scenario 1 by the proposed method (top) and the compared method that does not consider collision avoidance constraints in the OBC (bottom).

	joint range	velocity limits	acceleration limits	torque/force limits
driving wheels	-	$\pm 10.15\,\mathrm{rad/s}$	$60\,\mathrm{rad/s^2}$	$\pm 6\,\mathrm{Nm}$
torso	$[0, 0.35]\,\mathrm{m}$	$\pm 0.07\,\mathrm{m/s}$	$160\,\mathrm{m/s^2}$	$\pm 2000\,\mathrm{N}$
arm joint 1	$[0, 157.5]\,\mathrm{deg}$	$\pm 2.7\,\mathrm{rad/s}$	$39\,\mathrm{rad/s^2}$	$\pm 39\,\mathrm{Nm}$
arm joint 2	$[-90, 62.5]\,\mathrm{deg}$	$\pm 3.66\,\mathrm{rad/s}$	$38\,\mathrm{rad/s^2}$	$\pm 39\,\mathrm{Nm}$
arm joint 3	$[-202.5, 90]\,\mathrm{deg}$	$\pm 4.58\,\mathrm{rad/s}$	$67\,\mathrm{rad/s^2}$	$\pm 17.86\,\mathrm{Nm}$
arm joint 4	$[-22.5, 135]\,\mathrm{deg}$	$\pm 4.58\,\mathrm{rad/s}$	$73\,\mathrm{rad/s^2}$	$\pm 17.86\,\mathrm{Nm}$
arm joint 5	$[-120, 120]\,\mathrm{deg}$	$\pm 1.95\,\mathrm{rad/s}$	$105\,\mathrm{rad/s^2}$	$\pm 3\,\mathrm{Nm}$
arm joint 6	$[-81, 81]\,\mathrm{deg}$	$\pm 1.76\,\mathrm{rad/s}$	$400\,\mathrm{rad/s^2}$	$\pm 6.6\,\mathrm{Nm}$
arm joint 7	$[-120, 120]\,\mathrm{deg}$	$\pm 1.76\,\mathrm{rad/s}$	$5000\,\mathrm{rad/s^2}$	$\pm 6.6\,\mathrm{Nm}$

Table 7.1. The range, velocity limits, acceleration limits and torque/force limits of the TIAGo joints.

State and input bounds: Table 7.1 reports the range of the robot joints as well as the joint velocity and torque/force[2] limits, all obtained from the values reported at the URDF of TIAGo. For the acceleration limits that are required in the NMPC, the values were obtained after a series of simulations and are also reported in Table 7.1.

Collision avoidance with the environment: For the collision avoidance constraints between the robot and its environment, 4 bounding ellipsoids and 2 bounding spheres (ellipsoids with semi-axes of equal size) are used in order to cover the whole robot as illustrated in Fig. 7.2 (left). Note that regarding the ellipsoid that envelops the robot torso, the size of its vertical semi-axis is a function of the prismatic joint position. In an attempt to reduce the number of the considered constraints, only the closest obstacle is considered for each robot body based on (7.3).

Self-collision avoidance: Regarding the self-collision avoidance constraints, taking into account the joint limits and the structure of the robot we consider the following cases of possible self-collisions: gripper-head, gripper-torso, gripper-base, lower arm-head, lower arm-torso and lower arm-base. We consider one constraint for each

[2]The robot torso has a prismatic joint.

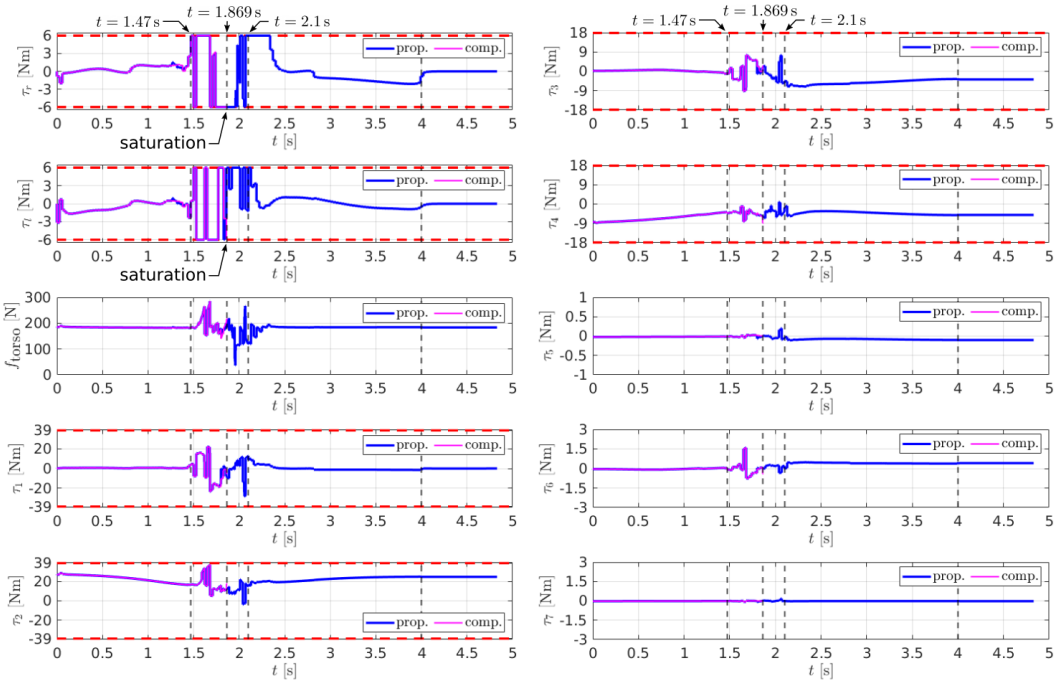

Figure 7.5. The joint torques (force for the prismatic joint of the robot torso) generated by the proposed method and the compared method. The dashed vertical lines highlight the time instants that correspond to the snapshots of Fig. 7.4.

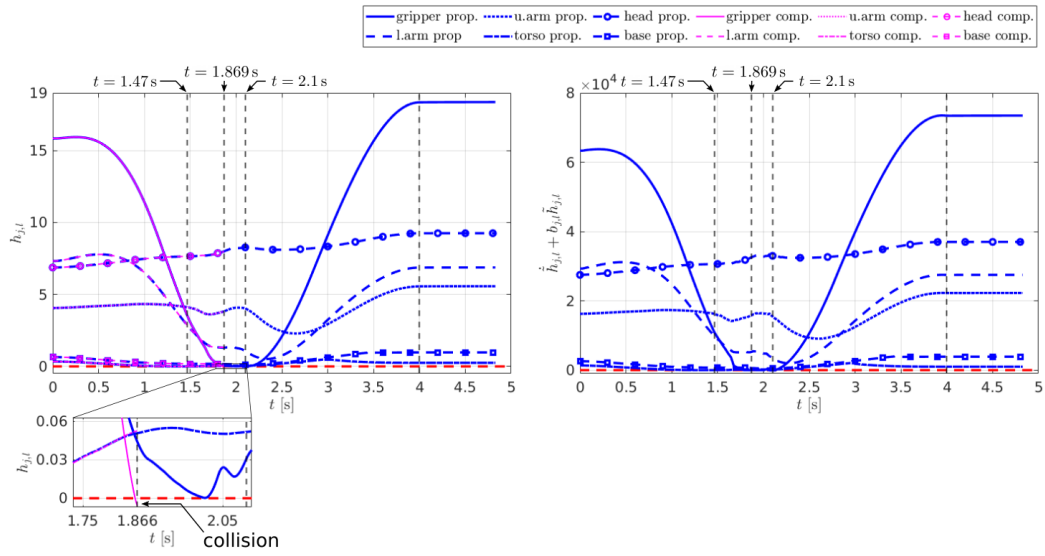

Figure 7.6. On the left, the values of the collision avoidance constraints between the robot bodies (gripper, lower arm, upper arm, head torso and base) and their closest obstacle, $h_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l)$, during the robot motion under the proposed and the compared method. On the right, the values of the CBF constraints $\dot{\tilde{h}}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l) + b_{j,l}\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l)$ for the proposed method.

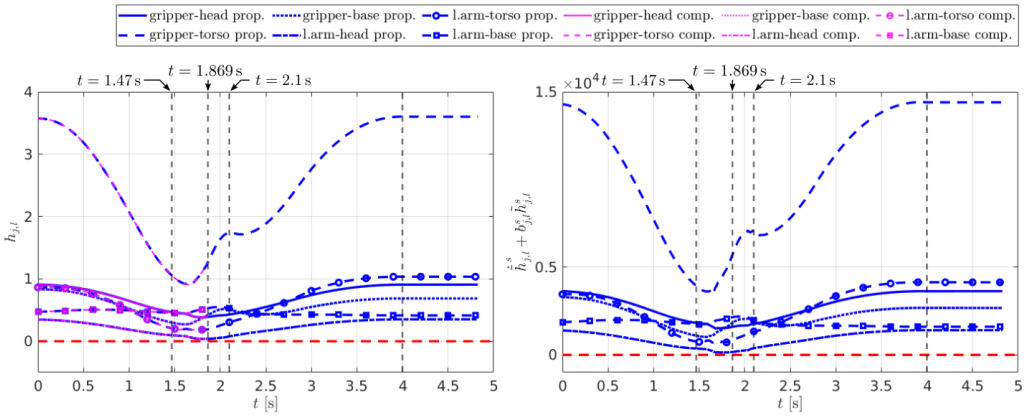

Figure 7.7. On the left, the values of the self-collision avoidance constraints, $h^s_{j,l}(\boldsymbol{x})$, during the robot motion under the proposed and the compared method. On the right, the value of the CBF constraints $\dot{\tilde{h}}^s_{j,l}(\boldsymbol{x}) + b^s_{j,l}\tilde{h}^s_{j,l}(\boldsymbol{x})$ for the proposed method.

combination treating the gripper and the lower arm as obstacles enveloping them with a sphere whose position is a function of the robot configuration, as illustrated in Fig. 7.2 (right).

Balance constraints: Regarding the balance constraints, TIAGo is in contact with the ground via 2 driving and 4 caster wheels, forming a support polygon with 6 edges. The considered support polygon is illustrated in Fig. 7.3. We take a conservative version of the support polygon in order to account for the lack of knowledge about the actual position of the caster wheels during the robot motion and any deformations of the robot wheels that could lead to contact points that do not coincide with the center of the wheel[3].

7.6.1 Simulation scenario 1

In this scenario, the end-effector has to follow a linear path of length 1.89 m (see Fig. 7.4). The task duration is 4 s, the desired velocity has a trapezoidal profile and the maximum acceleration along the path is $0.57\,\mathrm{m/s^2}$. The environment is populated by 2 spherical obstacles (see the two red spheres in Fig. 7.4). The first one $\mathcal{O}_1$ with radius $\rho_1 = 0.1$ m is placed in such a way that obstructs the desired end-effector path. The second obstacle $\mathcal{O}_2$ with radius $\rho_2 = 0.1$ m is placed in such a way that obstructs the robot base from moving freely.

To show the effectiveness of the proposed method we compare it with a version of it that does not consider the collision and self-collision avoidance constraints in the OBC. Note that in both cases the motion generation module has collision avoidance constraints, so the generated reference motion is collision-free.

In all the CBFs that correspond to the collision avoidance constraints between the robot and the obstacles we used the same parameter values $a_{j,l} = 20$ and $b_{j,l} = 200$. Similarly, in all the CBFs that correspond to the self-collision avoidance constraints we also used the parameters $a^s_{j,l} = 20$ and $b^s_{j,l} = 200$.

[3]Note that deformation of the wheel will not appear in the simulations since the wheels are considered rigid. However, in a real-world experiment, deformation of the wheel is a valid concern.

For the purpose of this comparison, we consider the robot being in collision if the collision avoidance constraints (7.3) and (7.4) are violated. Clearly, violation of these constraints does not necessarily mean that the robot actually experiences a collision, since the considered bounding geometries are rather conservative. However, it means that the imposed safety criteria are violated and the resulting solutions are infeasible.

The robot motion resulting from the use of the proposed and the compared method is illustrated in the accompanying video `https://youtu.be/xGqKRNUYpFU`, while snapshots of the motion are offered in Fig. 7.4. The actuation torques/forces generated by the two methods are illustrated in Fig. 7.5. Fig. 7.6 (left) illustrates the values of the collision avoidance constraints between the robot bodies and the environment obstacles $h_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l)$ throughout the robot motion, while Fig. 7.6 (right) illustrates the values of the corresponding CBF-based constraints $\dot{\tilde{h}}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l) + b_{j,l}\tilde{h}_{j,l}(\boldsymbol{x}, \boldsymbol{o}_l, \dot{\boldsymbol{o}}_l)$ for the proposed method throughout the robot motion. Finally, Fig. 7.7 (left) illustrates the values of the self-collision avoidance constraints $h^s_{j,l}(\boldsymbol{x})$, while Fig. 7.7 (right) the values of the corresponding CBF-based constraints $\dot{\tilde{h}}^s_{j,l}(\boldsymbol{x}) + b^s_{j,l}\tilde{h}^s_{j,l}(\boldsymbol{x})$.

We can notice that although the proposed method was able to generate the desired motion and accomplish the task (see Fig. 7.4), the compared method violated the collision avoidance constraint between the gripper and obstacle $\mathcal{O}_1$ at time $t = 1.869\,\text{s}$, as illustrated in Fig. 7.6 (zoomed area). One can notice that at the moment of the collision, the torques of the right and left driving wheels, τ_r and τ_l respectively, were saturated (see Fig. 7.5). This implies that although the motion generation module created a reference motion that was collision-free, that motion was not kinodynamically feasible and when the OBC was called to track it, a tracking error occurred that eventually lead the compared method to a collision.

On the contrary, we can notice that the proposed method does not violate the constraint, thanks to the CBF-based collision avoidance constraints incorporated in the OBC, ensuring that if the OBC is unable to accurately track the reference motion, the resulting motion will be at least collision-free.

Regarding the computational time required from the motion generation module (NMPC), the maximum reported was 19.4 ms while for the proposed OBC the maximum reported computational time was 1.8 ms, so lower than the considered sampling times

7.6.2 Simulation scenario 2

In the previous scenario, the robot motion was not aggressive enough in order to challenge the robot balance and so the balance constraints (7.9) were never active during the robot motion. To this behavior contributes the rather heavy base of the considered robot.

In this simulation scenario, we try to generate a highly dynamic motion in order to see how the proposed method behaves at the boundary of balance loss. Here the robot end-effector has to follow an "eight-curve" (see Fig. 7.8). The center of the curve is placed 0.75 m over the ground, while the curve is tilted around its horizontal axis by 45 deg. The total length of the path is $\approx 6.1\,\text{m}$. The task duration is

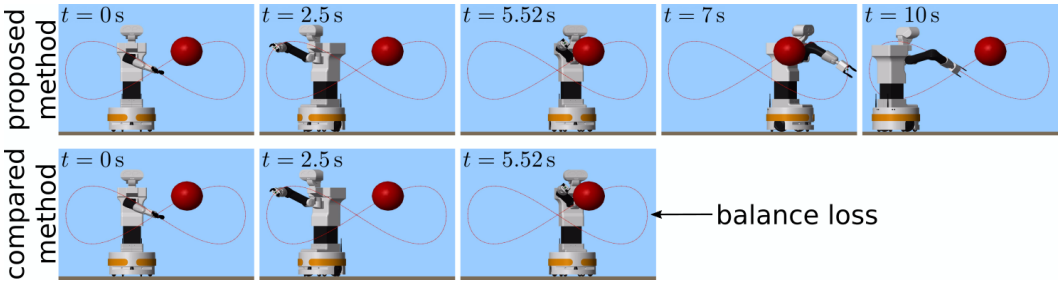

Figure 7.8. Snapshots of the motion generated during the simulation scenario 2 by the proposed method (top) and the compared method that does not consider balance constraints in the OBC (bottom).

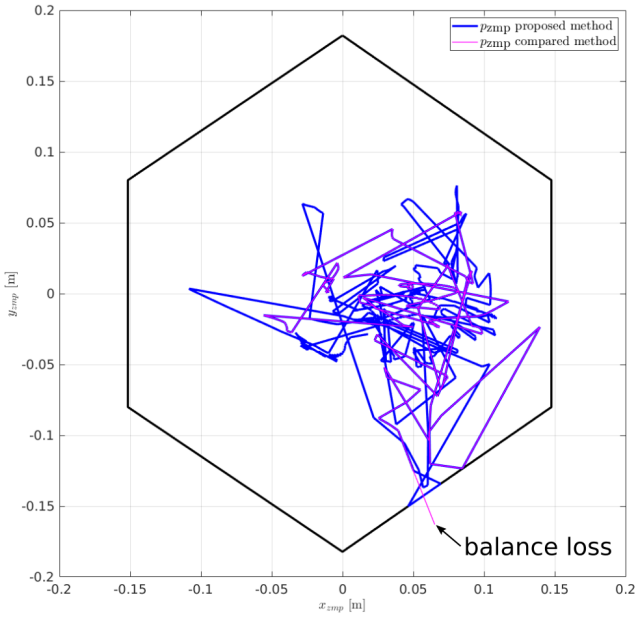

Figure 7.9. The position of the ZMP during the motion generated by the proposed and the compared method.

10 s. The desired end-effector velocity has a trapezoidal profile, while the maximum acceleration along the path is $0.38\,\mathrm{m/s^2}$. The desired end-effector path is obstructed by an obstacle $\mathcal{O}$ with radius $\rho = 0.18\,\mathrm{m}$ as illustrated in Fig. 7.8.

In order to show the effectiveness of the proposed OBC, we compare it with a version of it that does not consider the balance constraints (7.9). In both methods, we set the parameters of the CBFs for the collision avoidance constraints to $a_{j,l} = 20$ and $b_{j,l} = 20$ and for the self-collision avoidance constraints to $a^s_{j,l} = 20$ and $b^s_{j,l} = 20$. Note that the motion generation module, since it does not consider the robot dynamics, generates references that might cause loss of balance.

For the purpose of the simulation, we consider the robot losing its balance if its ZMP is outside of the considered support polygon.

The robot motion resulting from the use of the proposed and the compared method is also illustrated in the accompanying video, while snapshots of the motion are offered in Fig. 7.8. The proposed method was able to complete the task without

experiencing collisions or balance loss. On the contrary, the compared method in the vicinity of the obstacle violated the balance constraint as it is evident from the position of the ZMP in Fig. 7.9.

Regarding the computational time required from the motion generation module, the maximum reported was 18.4 ms while for the proposed OBC the maximum reported computational time was 2.2 ms, always lower than the considered sampling times, even in cases where both the collision and the balance constraints were active.

7.7 Discussion

In this section, we offer a brief discussion on the presented motion generation scheme and the proposed OBC.

First, one can notice that the motion generation module and the OBC use different collision avoidance constraints, with the constraints in the OBC being more conservative. Since the constraints (7.3) and (7.4) used in the motion generation module cannot be applied immediately in the OBC, one could use the more conservative constraints (7.10) and (7.14) of the OBC in the motion generation module in an attempt to introduce the same level of conservativeness to both modules. Note, however, that even in this case, the resulting constraints would have been different as a CBF constraint involves the robot model and the considered robot models in the two modules are different. Moreover, using the constraints (7.10) and (7.14) in the motion generation module would lead to an increase in the required computational time. So, in an attempt to maintain the real-time performance, one would have to decrease the prediction horizon. Clearly, a deeper investigation on the advantages and disadvantages of such a choice is required.

Note also that the collision avoidance constraints used in the OBC, should not be less conservative than the ones used in the motion generation module. In a different case, even if in the vicinity of an obstacle the OBC leads the robot to collision-free states, it is possible that these same states violate the more conservative collision avoidance constraints of the motion generation module, leading to infeasibility.

In any case, in the proposed scheme, one can try to reduce the conservativeness of the OBC by appropriately tuning the parameters $a_{j,l}$, $b_{j,l}$, $a^s_{j,l}$ and $b^s_{j,l}$ of the CBFs.

A second comment concerns the behavior of the proposed method during the execution of aggressive motions that could result in loss of balance. As we showed in simulation scenario 2, the proposed OBC managed to maintain the ZMP within the limits of the support polygon, thanks to the considered balance constraints. On the contrary, the compared method, which did not have such constraints, in an attempt to follow the reference motion generated by the motion generation module, had the ZMP outside the support polygon.

These results reveal the ability of the proposed OBC to enforce safety when the reference motion tends to drive the system to loss of balance. However, this ability of the OBC is limited to cases where a balance loss is not inevitable. This means that the OBC can prevent a balance loss if at the considered state the robot has the necessary actuation capabilities to do so. Clearly, an inevitable balance loss can emerge when the considered motion generation module sends constantly reference

motions that drive the robot to unbalanced states. In this case, the OBC will try to track these motions as close as possible while maintaining the ZMP inside the support polygon. This will continue until the moment when the robot will not have the necessary actuation capabilities to do so and an infeasibility will be reported.

There are two ways to reduce the possibility for an inevitable loss of balance to appear. The first is to include in the cost function of the OBC a term that will improve the robot balance, i.e., it will bring the ZMP toward the center of the support polygon. However, this approach would result in tracking errors even in cases where the robot balance is not challenged.

The second way, which is clearly the most effective, suggests the use by the motion generation module of a robot model that considers at least a simplified version of the robot dynamics (e.g., the robot centroidal dynamics). This will enable to include appropriate balance constraints or terms in the cost function that improve the robot balance. In this way, the reference motion will not constantly challenge the robot balance. Note, however, that also in this case the OBC will be required to have balance constraints since in the event of a kinodynamically infeasible yet balanced reference motion, the OBC will have to ensure that any tracking error would not lead to a loss of balance.

Note that our choice to use the simplest possible prediction model for the motion generation module was aiming to highlight the effectiveness of the proposed OBC and its ability to enforce safety even in the most challenging situations.

7.8 Conclusions

In this chapter, we considered the case in which the motion of a MM is generated by a motion generation module that does not consider the robot full dynamics, in an attempt to reduce the required computational time and achieve real-time performance. We introduced an optimization-based controller that considers the robot full dynamics and can be interposed between the motion generation module and the considered robot in order to track the reference motion and generate appropriate kinodynamically feasible commands for the robot. The OBC considers collision and self-collision avoidance constraints, built upon appropriate CBFs, in order to ensure that the resulting motion will be at least collision-free. The controller is complete with balance constraints to ensure that the robot will maintain its balance throughout the robot motion. The simulation results revealed that the introduced collision avoidance constraints can indeed protect the robot from collisions that could have occurred if an OBC that does not consider the proposed collision avoidance constraints had attempted to track a collision-free, but not kinodynamically feasible, motion in the vicinity of an obstacle. The simulation results also showed that the proposed OBC is able to maintain the robot balance even if the reference motion generated by the motion generation module leads to loss of balance.

Future work aims at

- the experimental validation of the proposed optimization-based controller;
- lifting the assumption for adequate friction between the driving wheels and the ground and including appropriate constraints that guarantee no-slipping

conditions;

- testing the controller alongside a motion generation module that considers the robot centroidal dynamics or another simplified model of the robot that can reason about the robot balance;
- conducting a comprehensive study on the influence of the parameters $a_{j,l}$, $b_{j,l}$, $a^s_{j,l}$ and $b^s_{j,l}$ to the conservativeness of the CBF-based collision avoidance constraints, in an attempt to reduce it as much as possible without compromising the robot safety.

Chapter 8

Conclusions

This thesis addressed the problem of safe motion generation for two robotic platforms, namely wheeled mobile robots and wheeled mobile manipulators. Due to their different structure and thus their individual safety risks, the motion generation problem for each considered robotic platform was addressed in separate parts of this thesis.

In the first part, we investigated the problem of motion generation for mobile robots called to navigate in environments populated by static and/or moving obstacles. During a navigation task, the robot has to maintain its safety as well as the safety of the environment in terms of collision avoidance.

For the solution of the considered problem, we used a real-time NMPC algorithm. Safety is enforced via appropriate collision avoidance constraints. However, we argue that in a real-time NMPC, where relatively short prediction horizons are dictated by the real-time requirement, the use of collision avoidance constraints that are based on purely distance information can jeopardize the robot safety. For this reason, we proposed two alternative collision avoidance constraints that enabled the robot to navigate safely in situations where their distance-based counterparts fail to protect the robot from collision.

In Chap. 4, within a framework for safe robot navigation in a human crowd, we employed an NMPC that considers collision avoidance constraints built upon appropriate CBFs. These constraints consider in addition to the robot-obstacle distance, its rate of change, giving a notion of lookahead to the collision avoidance constraint. Simulation results revealed the superiority of the proposed constraint against its purely distance-based counterpart and the effectiveness of the whole safe robot navigation framework in general.

In Chap. 5 we introduced the concept of avoidable collision states (ACS) and we proposed a collision avoidance constraint built upon it. The proposed constraint essentially requires that the robot is always at a state from which it can avoid collision with a certain obstacle. The constraint was applied to a real-time NMPC for robot navigation and the simulation results showed its superior performance over collision avoidance constraints that are based on purely distance information, especially when the robot navigates at high speed among moving obstacles.

Regarding the future work, in addition to the directions mentioned in each individual chapter, ensuring the robot safety requires to consider cases in which

the motion generation method fails to provide a feasible solution and the robot is exposed to the danger of collision. Clearly, an emergency stop is not ideal when the environment is populated by moving obstacles. Thus, the development of strategies that bring the robot to a safe state with respect to its environment in an event of an emergency is an interesting direction for future work.

In the second part of this thesis, we investigated the problem of motion generation for mobile manipulators, called to execute tasks that require aggressive motions (high accelerations) in environments populated by obstacles.

Clearly, while performing aggressive motions the robot is not only at risk of collision with its environment or itself, but it also faces the risk of losing its balance. As a result, in order to generate the desired motion one has to take into account all those risks.

In Chap. 6, we proposed an appropriate NMPC scheme to generate motions in real-time for a MM called to execute aggressive tasks. In order to ensure kinodynamic feasibility, we considered the full robot dynamics as a prediction model in the NMPC using also state and input bounds. To ensure that the resulting motion will be collision-free, we included appropriate collision and self-collision avoidance constraints. In order to maintain the robot balance we considered a constraint that essentially limits the moments exerted by the robot around the edges of the support polygon to remain non-negative. In order to enable the solution of the proposed NMPC using off-the-shelf solvers that require a symbolic representation of the OCP, we lifted the inherent nonlinearity of the balance constraint by linearizing it using the solution of the previous NMPC iteration.

Although this method was effective for MM of low dimensions, when the robots become more complex and the DOF increase, the use of the robot full dynamics as a prediction model in an NMPC might be prohibitive for achieving real-time performance. For this reason, in principle, simplified models of the robot dynamics are used. Since these models are unable to enforce kinodynamic feasibility, the resulting motion should be tracked by a controller that considers the robot full dynamics in order to generate appropriate commands for the robot. We argue, however, that if this controller does not consider constraints that ensure the robot safety in terms of collision and balance, then any tracking error that might occur due to a kinodynamically infeasible reference motion, could put the robot in danger. For this purpose, in Chap. 7 we proposed an optimization-based controller that not only enforces kinodynamic feasibility using the robot full dynamics but it also considers collision and self-collision avoidance constraint built upon appropriate CBFs and maintains the robot balance with balance constraints. In this way, the proposed controller ensures that the resulting motion will be at least safe, a property that was also verified by the simulation results.

In addition to the future works discussed at the end of each individual chapter, which aim at the improvement of the proposed methods, enforcing safety during the operation of a MM also requires the consideration of the, not impossible, scenario in which the considered motion generation approach fails to find a feasible solution. In this case, the robot and its environment are exposed to danger and an emergency action is required. For a wheeled mobile robot or a robot manipulator, this emergency action would consist of blocking the motion of the robot joints. On the contrary, blocking the joints motion in a MM could lead to loss of balance, damaging the

robot and its environment. Clearly, this leaves space for the development of motion generation approaches that during an emergency would drive the robot to safe states that permit the joint motion to stop. Such approaches would be able to complete a general framework for safe motion generation for mobile manipulators.

Appendix A

Robot dynamics

The dynamic model of the robot provides a mapping between the generalized forces acting on the robot and its motion. Two methods commonly used for the derivation of the robot equations of motion are the *Lagrange formulation* and the *Newton-Euler formulation*. Here we briefly present the *Lagrange formulation* based on [52].

Consider an n-dimensional mechanical system[1] (robot). Its generalized coordinates q_i, with $i = 1, ..., n$, represent the variables of the n joints and can be collected in the vector

$$\boldsymbol{q} = (q_1, ..., q_n)$$

Let us also consider the robot moving possibly under the influence of kinematic constraints that can be expressed in Pfaffian form as

$$\boldsymbol{A}^T(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0} \tag{A.1}$$

where $\boldsymbol{A}^T(\boldsymbol{q})$ a matrix with dimensions $k \times n$ that characterizes the constraints, with k the number of constraints.

We define the *Lagrangian* of the system as the difference between its kinetic energy $\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and its potential energy $\mathcal{U}(\boldsymbol{q})$

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \mathcal{U}(\boldsymbol{q}). \tag{A.2}$$

Having computed the Lagrangian as a function of the generalized coordinates and the generalized velocities, the Lagrange equations are

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}}\right)^T - \left(\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}}\right)^T = \boldsymbol{\xi} \tag{A.3}$$

with $\boldsymbol{\xi}$ being the vector of the generalized forces associated with the generalized coordinates. These generalized forces are comprised of the generalized forces from the robot actuators, joint friction and forces resulting from the contact of the robot with its environment.

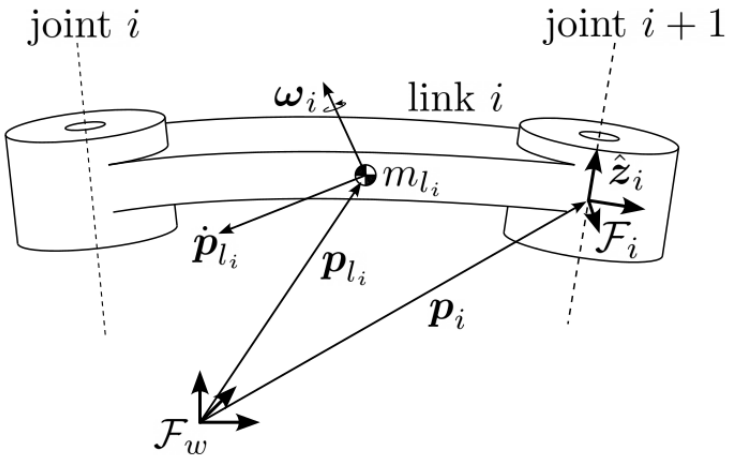

Figure A.1. The link i connected with the previous and following links with the joint i and joint $i+1$, respectively. The frame of the link $\mathcal{F}_i$ is chosen based on the Denavit-Hartenberg convention.

A.1 Kinetic energy

Consider the n links of the robotic system. The total kinetic energy of the robot is the sum of the kinetic energies of each individual link[2][3]

$$\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{i=1}^{n} \mathcal{T}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) \tag{A.4}$$

Consider the i-th link of the robot (see Fig. A.1). Denote by $\mathcal{F}_w$ the world reference frame and by $\mathcal{F}_i$ a reference frame attached to the i-th link while by $\boldsymbol{R}_i(\boldsymbol{q})$ denote the rotation matrix from frame $\mathcal{F}_i$ to the world frame $\mathcal{F}_w$. For the analysis that follows we consider the reference frames on the links selected by using the *Denavit-Hartenberg convention* [52]. With $\boldsymbol{p}_i$ denote the position vector of the frame $\mathcal{F}_i$ origin while with $\hat{\boldsymbol{z}}_i$ a unit vector along the z-axis of $\mathcal{F}_i$. Denote by $\boldsymbol{p}_{l_i}$ the position vector of the center of mass (CoM) with respect to $\mathcal{F}_w$, by m_{l_i} the mass of the i-th link and by $\boldsymbol{I}_{l_i}^i$ the inertia tensor of the i-th link expressed in frame $\mathcal{F}_i$. By $\dot{\boldsymbol{p}}_{l_i}$ and $\boldsymbol{\omega}_i$ denote the translational velocity of CoM and the angular velocity of the i-th link, respectively.

The kinetic energy consists of the sum of the contributions that correspond to the translational and rotational motion of the link

$$\mathcal{T}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} m_{l_i} \dot{\boldsymbol{p}}_{l_i}^T \dot{\boldsymbol{p}}_{l_i} + \frac{1}{2} \boldsymbol{\omega}_i^T \boldsymbol{R}_i(\boldsymbol{q}) \boldsymbol{I}_{l_i}^i \boldsymbol{R}_i^T(\boldsymbol{q}) \boldsymbol{\omega}_i \tag{A.5}$$

Note that $\dot{\boldsymbol{p}}_{l_i}$ and $\boldsymbol{\omega}_i$ are functions of the robot generalized velocities $\dot{\boldsymbol{q}}$ and can be

[1]For our analysis we assume that the robot consists of only rigid bodies (links).

[2]Note that in principle, for the total kinetic energy of the robot, the kinetic energy of the motors has to be also considered. However, for simplicity we neglect the kinetic energy of the motors.

[3]In the case where fictitious joints are used for the representation of the mobile base, like in Chapter 6, it is assumed that the intermediate fictitious links are mass-less and thus they do not contribute to the kinetic energy of the robot

expressed respectively as

$$\dot{\boldsymbol{p}}_{l_i} = \boldsymbol{J}_P^{l_i}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{A.6a}$$

$$\boldsymbol{\omega}_i = \boldsymbol{J}_O^{l_i}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{A.6b}$$

with

$$\boldsymbol{J}_P^{l_i}(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{\jmath}_{P,1}^{l_i}(\boldsymbol{q}) & \dots & \boldsymbol{\jmath}_{P,n}^{l_i}(\boldsymbol{q}) \end{pmatrix} \tag{A.7a}$$

$$\boldsymbol{J}_O^{l_i}(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{\jmath}_{O,1}^{l_i}(\boldsymbol{q}) & \dots & \boldsymbol{\jmath}_{O,n}^{l_i}(\boldsymbol{q}) \end{pmatrix} \tag{A.7b}$$

In equation (A.6a), $\boldsymbol{J}_P^{l_i}(\boldsymbol{q})$ is the $3 \times n$ matrix relating the contribution of the generalized velocities $\dot{\boldsymbol{q}}$ to the translational velocity of the i-th link, $\dot{\boldsymbol{p}}_{l_i}$, while the product $\boldsymbol{\jmath}_{P,j}^{l_i}(\boldsymbol{q})\dot{q}_j$ corresponds to the contribution of the j-th joint. Similarly, in (A.6b), $\boldsymbol{J}_O^{l_i}(\boldsymbol{q})$ is the $3\times n$ matrix relating the contribution of the generalized velocities $\dot{\boldsymbol{q}}$ to the angular velocity of link i, $\boldsymbol{\omega}_i$, while the product $\boldsymbol{\jmath}_{O,j}^{l_i}(\boldsymbol{q})\dot{q}_j$ corresponds to the contribution of the j-th joint,

Note that only $n_i \leq n$ joints contribute to the motion of link i. Thus, if $\boldsymbol{\jmath}_{P,j}^{l_i}(\boldsymbol{q})$ and $\boldsymbol{\jmath}_{O,j}^{l_i}(\boldsymbol{q})$ correspond to one of the $n - n_i$ joints that do not contribute to the motion of link i, then they have zero entries. Otherwise, if the j-th joint is prismatic then

$$\boldsymbol{\jmath}_{P,j}^{l_i}(\boldsymbol{q}) = \hat{\boldsymbol{z}}_{j-1}$$
$$\boldsymbol{\jmath}_{O,j}^{l_i}(\boldsymbol{q}) = \boldsymbol{0}$$

while if the joint is revolute then

$$\boldsymbol{\jmath}_{P,j}^{l_i}(\boldsymbol{q}) = \hat{\boldsymbol{z}}_{j-1} \times (\boldsymbol{p}_{l_i} - \boldsymbol{p}_{j-1})$$
$$\boldsymbol{\jmath}_{O,j}^{l_i}(\boldsymbol{q}) = \hat{\boldsymbol{z}}_{j-1}$$

where $\boldsymbol{p}_{j-1}$ is the position vector of frame $\mathcal{F}_{j-1}$ and $\hat{\boldsymbol{z}}_{j-1}$ is the unit vector of z-axis of frame $\mathcal{F}_{j-1}$.

In the end, the kinetic energy of the link can be expressed as

$$\mathcal{T}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} m_{l_i} \dot{\boldsymbol{q}}^T {\boldsymbol{J}_P^{l_i}}^T \boldsymbol{J}_P^{l_i} \dot{\boldsymbol{q}} + \frac{1}{2} \dot{\boldsymbol{q}}^T {\boldsymbol{J}_O^{l_i}}^T \boldsymbol{R}_i \boldsymbol{I}_{l_i}^i \boldsymbol{R}_i^T \boldsymbol{J}_O^{l_i} \dot{\boldsymbol{q}} \tag{A.8}$$

where the dependency on $\boldsymbol{q}$ was dropped.

Substituting (A.8) in (A.4) we get the total kinetic energy in the form

$$\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{i=1}^{n} \frac{1}{2} m_{l_i} \dot{\boldsymbol{q}}^T {\boldsymbol{J}_P^{l_i}}^T \boldsymbol{J}_P^{l_i} \dot{\boldsymbol{q}} + \frac{1}{2} \dot{\boldsymbol{q}}^T {\boldsymbol{J}_O^{l_i}}^T \boldsymbol{R}_i \boldsymbol{I}_{l_i}^i \boldsymbol{R}_i^T \boldsymbol{J}_O^{l_i} \dot{\boldsymbol{q}}$$

Note that we can write the total kinetic energy also in the quadratic form

$$\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{i=1}^{n} \frac{1}{2} m_{l_i} \dot{\boldsymbol{q}}^T {\boldsymbol{J}_P^{l_i}}^T \boldsymbol{J}_P^{l_i} \dot{\boldsymbol{q}} + \frac{1}{2} \dot{\boldsymbol{q}}^T {\boldsymbol{J}_O^{l_i}}^T \boldsymbol{R}_i \boldsymbol{I}_{l_i}^i \boldsymbol{R}_i^T \boldsymbol{J}_O^{l_i} \dot{\boldsymbol{q}} = \frac{1}{2} \dot{\boldsymbol{q}}^T \boldsymbol{B}(\boldsymbol{q}) \dot{\boldsymbol{q}} \tag{A.9}$$

where

$$\boldsymbol{B}(\boldsymbol{q}) = \sum_{i=1}^{n} m_{l_i} {\boldsymbol{J}_P^{l_i}}^T \boldsymbol{J}_P^{l_i} + {\boldsymbol{J}_O^{l_i}}^T \boldsymbol{R}_i \boldsymbol{I}_{l_i}^i \boldsymbol{R}_i^T \boldsymbol{J}_O^{l_i}$$

is the *inertia matrix*.

A.2 Potential energy

The potential energy of the robot is the sum of the potential energy of each individual link

$$\mathcal{U}(\boldsymbol{q}) = \sum_{i=1}^{n} \mathcal{U}_i(\boldsymbol{q}). \tag{A.10}$$

Again we work with the assumption that the links are rigid and we define the potential energy of the i-th link as

$$\mathcal{U}_i(\boldsymbol{q}) = -m_{l_i} \boldsymbol{g}_w^T \boldsymbol{p}_{l_i} \tag{A.11}$$

where $\boldsymbol{g}_w$ is the gravity acceleration vector expressed in the world frame $\mathcal{F}_w$.

Substituting (A.11) in (A.10) we get the total potential energy of the system

$$\mathcal{U}(\boldsymbol{q}) = -\sum_{i=1}^{n} m_{l_i} \boldsymbol{g}_w^T \boldsymbol{p}_{l_i}. \tag{A.12}$$

A.3 Equations of motion

Having computed the total kinetic energy (A.9) and the total potential energy (A.12) as a function of the generalized coordinates and the generalized velocities, we can substitute the Lagrangian (A.2) in (A.3) that gives the equations of motion in the form

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\xi} \tag{A.13}$$

where $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in I\!R^n$ is the vector of velocity and gravitational terms for which holds

$$\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{B}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \boldsymbol{q}}\left(\dot{\boldsymbol{q}}^T \boldsymbol{B}(\boldsymbol{q})\dot{\boldsymbol{q}}\right)\right)^T + \left(\frac{\partial \mathcal{U}(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)^T. \tag{A.14}$$

Regarding the generalized forces $\boldsymbol{\xi}$ that perform work on the robot joints, we consider the generalized forces from the actuators and the reaction forces emanating from the imposed to the robot kinematic constraints[4]

$$\boldsymbol{\xi} = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{\tau} + \boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda} \tag{A.15}$$

where $\boldsymbol{\tau} \in I\!R^{n_\tau}$ is the vector of the forces/moments applied by the n_τ robot actuators, $\boldsymbol{S}(\boldsymbol{q}) \in I\!R^{n \times n_\tau}$ the matrix that maps the actuator generalized forces to generalized forces performing work on the generalized coordinates, $\boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda}$ the vector of the reaction forces at the generalized coordinates level emanating from the kinematic constraints (A.1) and $\boldsymbol{\lambda} \in I\!R^k$ the vector of the *Lagrange multipliers*.

Substituting (A.15) in (A.13) and given the kinematic constraints (A.1) imposed to the robot, the equation of motion of the robot can be written as

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{\tau} + \boldsymbol{A}(\boldsymbol{q})\boldsymbol{\lambda} \tag{A.16a}$$

$$\boldsymbol{A}^T(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0} \tag{A.16b}$$

[4]Note that for a more detailed dynamic model one has to also consider the generalized forces due to the friction at the robot joints. However, in this work we will assume that they are negligible.

Appendix B

Zero moment point

The Zero-Moment Point (ZMP), originally introduced for humanoid robots [97], was also considered in the case of wheeled mobile manipulators as a measure to evaluate the robot balance. According to the definition given in [100] *ZMP is the point on the ground about which the sum of all the moments of active forces are equal to zero.* The robot will not lose balance, if the ZMP lies within the support polygon, which is defined by the contact points between the robot wheels and the ground.

Using Fig. B.1 as a reference, let us denote by $\mathcal{F}_v$ a reference frame whose origin lies on the ground, by $\boldsymbol{e}_i$ the unit vector of the i-th support polygon edge and by $\boldsymbol{p}_i$ the position of its starting point (the i-th contact point), both expressed in $\mathcal{F}_v$. Let us also denote by $\boldsymbol{f}_v$ and $\boldsymbol{\mu}_v$ the forces orthogonal to the ground and the moments parallel to the ground, respectively, that the robot exerts at a point L on the ground, while denote by $\boldsymbol{l}_v$ the position vector of L with respect to $\mathcal{F}_v$.

For our analysis, we assume that the ground-wheel friction is adequate to balance the parallel to the ground forces and the orthogonal to the ground moments that the robot exerts to the ground.

If $\boldsymbol{f}_t$ is the total ground support force due to the contact of the robot with the ground and $\boldsymbol{p}_{\text{zmp}}$ the position vector of the ZMP, according to the definition for the ZMP the following static equilibrium holds

$$\boldsymbol{f}_t + \boldsymbol{f}_v = \boldsymbol{0} \tag{B.1a}$$

$$\boldsymbol{p}_{\text{zmp}} \times \boldsymbol{f}_t + \boldsymbol{l}_v \times \boldsymbol{f}_v + \boldsymbol{\mu}_v = \boldsymbol{0} \tag{B.1b}$$

while by substituting (B.1a) in (B.1b) we get:

$$\boldsymbol{l}_v \times \boldsymbol{f}_v + \boldsymbol{\mu}_v = \boldsymbol{p}_{\text{zmp}} \times \boldsymbol{f}_v \tag{B.2}$$

Now we will show the relation between the ZMP position $\boldsymbol{p}_{\text{zmp}}$ and the moment μ_i that the robot exerts around the i-th support polygon edge.

The moment μ_i is defined as

$$\mu_i = \boldsymbol{e}_i^T(-(\boldsymbol{p}_i - \boldsymbol{l}_v) \times \boldsymbol{f}_v + \boldsymbol{\mu}_v) \tag{B.3}$$

If we substitute (B.2) in (B.3) we get the relation between the moment around the i-th edge of the support polygon and the position of the ZMP, that is:

$$\boldsymbol{\mu}_i = \boldsymbol{f}_v^T(\boldsymbol{e}_i \times (\boldsymbol{p}_{\text{zmp}} - \boldsymbol{p}_i)) \tag{B.4}$$

From this relation we can deduce that:

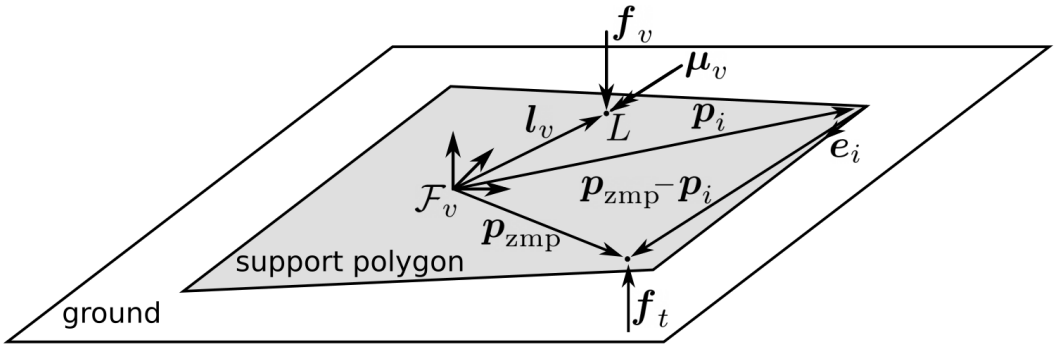

Figure B.1. The robot support polygon. By $\boldsymbol{e}_i$ we denote the unit vector of its i-th edge and by $\boldsymbol{p}_i$ the position of its starting point, all expressed in $\mathcal{F}_v$.

- if the ZMP lies on the half-plane that is defined by e_i and contains the support polygon, then $\mu_i > 0$;
- if the ZMP lies on the half-plane that is defined by e_i but it does not contain the support polygon, then $\mu_i < 0$;
- if the ZMP lies on the line generated by $\boldsymbol{e}_i$, then $\mu_i = 0$.

Bibliography

[1] 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 Conf. on Decision and Control*, pp. 3321–3328, 2022.

[2] S. G. Tarantos and G. Oriolo, "A dynamics-aware NMPC method for robot navigation among moving obstacles," in *Intelligent Autonomous Systems 17* (I. Petrovic, E. Menegatti, and I. Marković, eds.), (Cham), pp. 216–230, Springer Nature Switzerland, 2023.

[3] S. G. Tarantos and G. Oriolo, "Real-time motion generation for mobile manipulators via NMPC with balance constraints," in *2022 30th Med. Conf. on Control and Automation*, pp. 853–860, 2022.

[4] R. Findeisen and F. Allgöwer, "An introduction to nonlinear model predictive control," in *21st Benelux meeting on systems and control*, vol. 11, pp. 119–141, 2002.

[5] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[6] K. R. Muske and J. B. Rawlings, "Model predictive control with linear models," *AIChE Journal*, vol. 39, no. 2, pp. 262–287, 1993.

[7] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *2012 IEEE 51st IEEE Conf. on Decision and Control*, pp. 668–674, 2012.

[8] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[9] J. V. Frasch, S. Sager, and M. Diehl, "A parallel quadratic programming method for dynamic optimization problems," *Mathematical programming computation*, vol. 7, no. 3, pp. 289–329, 2015.

[10] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *2014 European Control Conf.*, pp. 128–133, 2014.

[11] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2014.

[12] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear mpc: bridging the gap via the real-time iteration," *Int. J. of Control*, vol. 93, pp. 1–19, 09 2016.

[13] T. Albin, D. Ritter, D. Abel, N. Liberda, R. Quirynen, and M. Diehl, "Nonlinear MPC for a two-stage turbocharged gasoline engine airpath," in *2015 54th IEEE Conf. on Decision and Control*, pp. 849–856, 2015.

[14] S. Gros, R. Quirynen, and M. Diehl, "Aircraft control based on fast non-linear MPC & multiple-shooting," in *2012 IEEE 51st IEEE Conf. on Decision and Control*, pp. 1142–1147, 2012.

[15] S. Gros, M. Vukov, and M. Diehl, "A real-time MHE and NMPC scheme for wind turbine control," in *52nd IEEE Conf. on Decision and Control*, pp. 1007–1012, 2013.

[16] M. Vukov, W. Van Loock, B. Houska, H. J. Ferreau, J. Swevers, and M. Diehl, "Experimental validation of nonlinear MPC on an overhead crane using automatic code generation," in *2012 American Control Conference*, pp. 6264–6269, 2012.

[17] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279 – 2285, 2011.

[18] M. Bardi and I. Capuzzo-Dolcetta, *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*. Modern Birkhäuser Classics, Birkhäuser Boston, 2009.

[19] R. F. Hartl, S. Sethi, and R. Vickson, "A survey of the maximum principles for optimal control problems with state constraints," *SIAM Review*, vol. 37, no. 2, pp. 181–218, 1995.

[20] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, Second Edition*. Society for Industrial and Applied Mathematics, 2010.

[21] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[22] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, 2e ed., 2006.

[23] M. Zanon, A. Boccia, V. G. S. Palma, S. Parenti, and I. Xausa, *Direct Optimal Control and Model Predictive Control*, pp. 263–382. Springer International Publishing, 2017.

[24] H. Bock and K. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *9th IFAC World Congress*, vol. 17, no. 2, pp. 1603–1608, 1984.

[25] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, May 2006.

[26] H. G. Bock, *Recent Advances in Parameter identification Techniques for O.D.E.*, pp. 95–121. Boston, MA: Birkhäuser Boston, 1983.

[27] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, "Auto-generated algorithms for nonlinear model predictive control on long and on short horizons," in *52nd IEEE Conference on Decision and Control*, pp. 5113–5118, 2013.

[28] M. Diehl, H. J. Ferreau, and N. Haverbeke, *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*, pp. 391–417. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[29] W. C. Li and L. T. Biegler, "Multistep, Newton-type control strategies for constrained, nonlinear processes," *Chemical Engineering Research and Design*, vol. 67, pp. 562 – 577, 1989.

[30] T. Ohtsuka, "A continuation/GMRES method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40, no. 4, pp. 563–574, 2004.

[31] V. M. Zavala and L. T. Biegler, "The advanced-step NMPC controller: Optimality, stability and robustness," *Automatica*, vol. 45, no. 1, pp. 86–93, 2009.

[32] M. Diehl, H. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *J. of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[33] M. Diehl, H. Bock, and J. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM J. Control and Optimization*, vol. 43, pp. 1714–1736, 2005.

[34] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Oct 2021.

[35] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[36] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 685–704, 2015.

[37] J. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019.

[38] G. Frison, D. Kouzoupis, A. Zanelli, and M. Diehl, "Blasfeo: Basic linear algebra subroutines for embedded optimization," *ACM Trans. on Mathematical Software*, vol. 44, 04 2017.

[39] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[40] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 3, p. 463–497, 2015.

[41] B. Patle, G. Babu L, A. Pandey, D. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.

[42] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[43] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[44] S. Lavalle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: New directions*, 01 2000.

[45] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The Int. J. of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[46] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The Int. J. of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[47] P. Abichandani, G. Ford, H. Y. Benson, and M. Kam, "Mathematical programming for multi-vehicle motion planning problems," in *IEEE Int. Conf. on Robotics and Automation*, pp. 3315–3322, 2012.

[48] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[49] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[50] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The Int. J. of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

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

[52] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, London, 2009.

[53] R. B. Patel and P. J. Goulart, "Trajectory generation for aircraft avoidance maneuvers using online optimization," *J. of Guidance, Control, and Dynamics*, vol. 34, no. 1, pp. 218–230, 2011.

[54] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Trans. on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2021.

[55] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl, "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles," in *2013 European Control Conference*, pp. 4136–4141, 2013.

[56] 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.

[57] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The Int. J. of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[58] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *2019 Int. Conf. on Robotics and Automation*, pp. 6741–6747, 2019.

[59] A. Sathya, P. Sopasakis, R. Van Parys, A. Themelis, G. Pipeleers, and P. Patrinos, "Embedded nonlinear model predictive control for obstacle avoidance using PANOC," in *2018 European Control Conference*, pp. 1523–1528, 2018.

[60] H. Febbo, J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "Moving obstacle avoidance for large, high-speed autonomous ground vehicles," in *2017 American Control Conference*, pp. 5568–5573, 2017.

[61] C. Jewison, R. S. Erwin, and A. Saenz-Otero, "Model predictive control with ellipsoid obstacle constraints for spacecraft rendezvous," *IFAC-PapersOnLine*, vol. 48, no. 9, pp. 257–262, 2015. 1st IFAC Workshop on Advanced Control and Navigation for Autonomous Aerospace Vehicles ACNAAV'15.

[62] H. Febbo, P. Jayakumar, J. L. Stein, and T. Ersal, "Real-Time Trajectory Planning for Automated Vehicle Safety and Performance in Dynamic Environments," *J. of Autonomous Vehicles and Systems*, vol. 1, 12 2021. 041001.

[63] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4459–4466, 2019.

[64] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Trans. on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[65] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference*, pp. 3420–3431, 2019.

[66] A. Agrawal and K. Sreenath, "Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation.," in *Robotics: Science and Systems*, vol. 13, Cambridge, MA, USA, 2017.

[67] T. D. Son and Q. Nguyen, "Safety-critical control for non-affine nonlinear systems with application on autonomous vehicle," in *2019 IEEE 58th Conf. on Decision and Control*, pp. 7623–7628, 2019.

[68] A. Thirugnanam, J. Zeng, and K. Sreenath, "Safety-critical control and planning for obstacle avoidance between polytopes with control barrier functions," in *2022 Int. Conf. on Robotics and Automation*, pp. 286–292, 2022.

[69] J. Zeng, B. Zhang, and K. Sreenath, "Safety-critical model predictive control with discrete-time control barrier function," in *2021 American Control Conference*, pp. 3882–3889, 2021.

[70] T. Fraichard and H. Asama, "Inevitable collision states. a step towards safer robots?," in *2003 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 1, pp. 388–393, 2003.

[71] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2210–2215, 2005.

[72] O. Gal, Z. Shiller, and E. Rimon, "Efficient and safe on-line motion planning in dynamic environments," in *2009 IEEE Int. Conf. on Robotics and Automation*, pp. 88–93, 2009.

[73] B. Damas and J. Santos-Victor, "Avoiding moving obstacles: the forbidden velocity map," in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 4393–4398, 2009.

[74] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," *IEEE Trans. on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, 2016.

[75] G. Buizza Avanzini, A. M. Zanchettin, and P. Rocco, "Constrained model predictive control for mobile robotic manipulators," *Robotica*, vol. 36, no. 1, p. 19–38, 2018.

[76] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[77] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. v. Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, pp. 1–37, 2021.

[78] T. Sandakalum and M. H. Ang, "Motion planning for mobile manipulators-a systematic review," *Machines*, vol. 10, no. 2, 2022.

[79] J. Desai and V. Kumar, "Nonholonomic motion planning for multiple mobile manipulators," in *IEEE Int. Conf. on Robotics and Automation*, vol. 4, pp. 3409–3414, 1997.

[80] J. Ward and J. Katupitiya, "Mobile manipulator motion planning towards multiple goal configurations," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2283–2288, 2006.

[81] G. Oriolo and C. Mongillo, "Motion planning for mobile manipulators along given end-effector paths," in *IEEE Int. Conf. on Robotics and Automation*, pp. 2154–2160, 2005.

[82] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 576–584, 2010.

[83] R. Seyboldt, C. Frese, and A. Zube, "Sampling-based path planning to cartesian goal positions for a mobile manipulator exploiting kinematic redundancy," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pp. 1–9, 2016.

[84] R. Luna, M. Moll, J. Badger, and L. E. Kavraki, "A scalable motion planner for high-dimensional kinematic systems," *The Int. J. of Robotics Research*, vol. 39, no. 4, pp. 361–388, 2020.

[85] M. Cefalo, P. Ferrari, and G. Oriolo, "An opportunistic strategy for motion planning in the presence of soft task constraints," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6294–6301, 2020.

[86] H. Tanner and K. Kyriakopoulos, "Nonholonomic motion planning for mobile manipulators," in *IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 1233–1238, 2000.

[87] O. Brock and L. Kavraki, "Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces," in *IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 1469–1474, 2001.

[88] A. Sideris and J. Bobrow, "An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems," in *2005 American Control Conf.*, pp. 2275–2280 vol. 4, 2005.

[89] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE Int. Conf. on Robotics and Automation*, pp. 2536–2542, 2020.

[90] M. Giftthaler, F. Farshidian, T. Sandy, L. Stadelmann, and J. Buchli, "Efficient kinematic planning for mobile manipulators with non-holonomic constraints using optimal control," in *2017 IEEE Int. Conf. on Robotics and Automation*, pp. 3411–3417, 2017.

[91] M. Logothetis, G. C. Karras, S. Heshmati-Alamdari, P. Vlantis, and K. J. Kyriakopoulos, "A model predictive control approach for vision-based object grasping via mobile manipulator," in *2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1–6, 2018.

[92] J. Pankert and M. Hutter, "Perceptive model predictive control for continuous mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6177–6184, 2020.

[93] M. Spahn, B. Brito, and J. Alonso-Mora, "Coupled mobile manipulation via trajectory optimization with free space decomposition," in *2021 IEEE Int. Conf. on Robotics and Automation*, pp. 12759–12765, 2021.

[94] S. Dubowsky and E. E. Vance, "Planning mobile manipulator motions considering vehicle dynamic stability constraints," in *1989 IEEE Int. Conf. on Robotics and Automation*, pp. 1271–1276, 1989.

[95] E. G. Papadopoulos and D. A. Rey, "A new measure of tipover stability margin for mobile manipulators," in *1996 IEEE Int. Conf. on Robotics and Automation*, pp. 3111–3116, 1996.

[96] D. A. Rey and E. G. Papadoupoulos, "Online automatic tipover prevention for mobile manipulators," in *in 1997 IEEE/RSJ Int. Conf. on Intelligent Robot and Systems*, pp. 1273–1278, 1997.

[97] M. Vukobratovic and B. Borovac, "Zero-moment point - thirty five years of its life.," *The Int. J. of Humanoid Robotics*, vol. 1, pp. 157–173, 2004.

[98] S. Sugano, Q. Huang, and I. Kato, "Stability criteria in controlling mobile robotic systems," in *1993 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 832–838, 1993.

[99] Q. Huang, S. Sugano, and I. Kato, "Stability control for a mobile manipulator using a potential method," in *1994 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 839–846, 1994.

[100] Q. Huang, K. Tanie, and S. Sugano, "Coordinated motion planning for a mobile manipulator considering stability and manipulation.," *The Int. J. of Robotic Research*, vol. 19, pp. 732–742, 2000.

[101] J. Kim, W. K. Chung, Y. Youm, and B. H. Lee, "Real-time ZMP compensation method using null motion for mobile manipulators," in *2002 IEEE Int. Conf. on Robotics and Automation*, pp. 1967–1972, 2002.

[102] S. Lee, M. Leibold, M. Buss, and F. C. Park, "Rollover prevention of mobile manipulators using invariance control and recursive analytic zmp gradients," *Advanced Robotics*, vol. 26, no. 11-12, pp. 1317–1341, 2012.

[103] C. Qiu, Q. Cao, L. Yu, and S. Miao, "Improving the stability level for on-line planning of mobile manipulators," *Robotica*, vol. 27, no. 3, p. 389–402, 2009.

[104] L. Yu, Q. Cao, C. Li, and C. Qiu, "On-line planning of nonholonomic mobile manipulators based on stability twist constraint," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 32, pp. 165–170, 2010.

[105] X. Ding, Y. Liu, J. Hou, and Q. Ma, "Online dynamic tip-over avoidance for a wheeled mobile manipulator with an improved tip-over moment stability criterion," *IEEE Access*, vol. 7, pp. 67632–67645, 2019.

[106] J. Chen, W. Zhan, and M. Tomizuka, "Constrained iterative lqr for on-road autonomous driving motion planning," in *2017 IEEE 20th Int. Conf. on Intelligent Transportation Systems*, pp. 1–7, 2017.

[107] M. Diehl, H. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[108] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *2014 IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 295–302, 2014.

[109] D. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, vol. 35, 10 2013.

[110] S. Kuindersma, F. Permenter, and R. Tedrake, "An efficiently solvable quadratic program for stabilizing dynamic locomotion," in *2014 IEEE Int. Conf. on Robotics and Automation*, pp. 2589–2594, 2014.

[111] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified MPC framework for whole-body dynamic locomotion and manipulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4688–4695, 2021.

[112] C. Dario Bellicoso, C. Gehring, J. Hwangbo, P. Fankhauser, and M. Hutter, "Perception-less terrain adaptation through whole body control and hierarchical optimization," in *2016 IEEE-RAS 16th Int. Conf. on Humanoid Robots*, pp. 558–564, 2016.

[113] J.-R. Chiu, J.-P. Sleiman, M. Mittal, F. Farshidian, and M. Hutter, "A collision-free MPC for whole-body dynamic locomotion and manipulation," in *2022 Int. Conf. on Robotics and Automation*, pp. 4686–4693, 2022.

[114] K. Bouyarmane, S. Caron, A. Escande, and A. Kheddar, *Multi-contact Motion Planning and Control*, pp. 1–42. Springer Netherlands, 2018.

[115] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," in *1987 IEEE Int. Conf. on Robotics and Automation*, vol. 4, pp. 1152–1159, 1987.

[116] M. Rauscher, M. Kimmel, and S. Hirche, "Constrained robot control using control barrier functions," in *2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 279–285, 2016.

[117] F. Ferraguti, C. Talignani Landi, S. Costi, M. Bonfè, S. Farsoni, C. Secchi, and C. Fantuzzi, "Safety barrier functions and multi-camera tracking for human–robot shared environment," *Robotics and Autonomous Systems*, vol. 124, p. 103388, 2020.

[118] M. A. Murtaza, S. Aguilera, V. Azimi, and S. Hutchinson, "Real-time safety and control of robotic manipulators with torque saturation in operational space," in *2021 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 702–708, 2021.

[119] Q. Nguyen and K. Sreenath, "Exponential Control Barrier Functions for enforcing high relative-degree safety-critical constraints," in *2016 American Control Conf.*, pp. 322–328, 2016.

[120] R. Grandia, A. J. Taylor, A. D. Ames, and M. Hutter, "Multi-layered safety for legged robots via control barrier functions and model predictive control," in *IEEE Int. Conf. on Robotics and Automation*, pp. 8352–8358, 2021.