SAPIENZA
UNIVERSITÀ DI ROMA

Generation and Control of Motion for 3D Humanoids and Steerable WMRs

Sapienza University of Rome

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

Michele Cipriano

ID number 1764645

Advisor

Prof. Giuseppe Oriolo

Academic Year 2023/2024

Thesis not yet defended

Generation and Control of Motion for 3D Humanoids and Steerable WMRs
Sapienza University of Rome

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

Version: February 21, 2024

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

Abstract

Abstract.

Contents

Chapter 1

Introduction

Todo.

From RAS2023:

Humanoid robots, thanks to their ability to perform legged locomotion, are in principle capable of moving through 3D environments, for which complex motions might be required. These can include stepping over or onto obstacles, climbing and descending stairs, moving through surfaces located at different height, overcoming gaps, and so on.

Walking on uneven ground, however, is a challenging task. Management of the footstep locations is subject to several constraints that are completely absent in flat ground locomotion, such as the necessity to avoid placing the feet over the edge of a stair step, or determining whether a given feature of the environment should be considered useful surface onto which the robot can step or simply regarded as an obstacle and avoided altogether.

Also from a control standpoint, walking on uneven ground introduces complications in the dynamic model that are not present in flat ground locomotion. In fact, traditional humanoid locomotion is realized by approximating the dynamics using linear models, which are relatively accurate in many application. Straight up extending these models to 3D environments can lead, without additional considerations, to nonlinearities which can severely impact the performance and theoretical guarantees of the controller.

1.1 Section

Todo.

Chapter 2

Literature review

Brief intro about literature here.

2.1 Humanoid robots

2.1.1 Gait generation

Once a footstep sequence has been planned, a whole-body motion must be generated in order to let the robot move by stepping over said sequence without falling. On flat ground, it is common to enforce dynamic balance by requiring that the Zero Moment Point (ZMP, the point with respect to which horizontal components of the momentum of contact forces are zero) is always contained inside the convex hull of contact surfaces, i.e., the *support polygon*. The ZMP cannot be controlled directly, but its dynamics can be related to the position and acceleration of the CoM. The traditional approach is to assume a constant CoM height and a negligible derivative of the angular momentum around the CoM, which leads to a Linear Inverted Pendulum (LIP) [1] model. The LIP has seen widespread use, also thanks to the fact that it allows to perform Model Predictive Control (MPC) using linear-quadratic optimization techniques, enforcing balance by means of constraints on the ZMP. However the constant CoM height assumption limits its ability to be employed for motion on uneven ground.

Traditionally, conditions on the ZMP implicitly assume that the latter is located on the ground, making these conditions obviously unsuitable to the 3D case where there is no univocally defined ground surface. In this work, we adopt an extension [2] of the basic criterion, in which the ZMP is instead a point in 3D space. According to this modified criterion, the 3D ZMP must belong to a pyramidal region whose extent is defined by the position of the Center of Mass (CoM) and the contact surfaces.

By letting the CoM height be a variable quantity, the CoM-ZMP relation takes the form of a Variable Height Inverted Pendulum (VH-IP) [3], where the stiffness of the pendulum itself is a control input. This model is nonlinear, which is usually a problem when trying to perform fast MPC, unless further approximations are introduced. However, it is possible to restrict the allowed trajectories of the CoM in such a way that the dynamics are linear and 3D motions are allowed [4, 5]. To do this, the pendulum stiffness is picked a priori, and the ZMP/CoM trajectories are generated in such a way to satisfy a linear relation.

A common problem in the field of humanoid gait generation is given by the fact that humanoid dynamics are unstable. This is seen in the LIP by the presence of an unstable mode, and signifies that even if one were to determine a gait such that the ZMP trajectory is always within the appropriate bounds, this might still not be enough, as the associated CoM trajectory might be divergent, rendering the resulting motion infeasible in practice. This issue is crucial and must be accounted for when designing the gait generation module, by providing appropriate guarantees against the divergence of the generated trajectories.

2.1.2 Footstep planning

Footstep planners can be subdivided in two broad categories based on whether they employ continuous or discrete techniques.

Planners based on continuous techniques compute sequences of footsteps via optimization, treating their poses as continuous decision variables. Several methods in this category (e.g., [6, 7, 8]) rely on the implicit assumption that the ground is completely flat and, therefore, are not tailored for motion generation in 3D environments. Explicit account of 3D environment is instead made in [9]: the ground surface is decomposed as a set of convex regions (with the aid of a manual initialization phase) and footsteps are placed by solving a mixed-integer quadratic problem (MIQP). A more recent work [10] casts the MIQP into a l_1-minimization problem: to reduce the computational complexity, a suboptimal solution is found by considering only those regions that intersect with the reachable workspace of the feet along a pre-planned trajectory for the floating base of the robot.

Planners based on discrete techniques find a solution by searching among particular sequences of footsteps. These sequences are generated by concatenation of *primitives*. A primitive is a displacement between two consecutive footsteps, selected among a finite number of possible displacements from a catalogue.

To search among all possible sequences, one possibility is to use a deterministic approach, which is typically represented by a variant of A*. Although this is possible and has been applied to 3D environments [11], the approach suffers from two main issues: the performance strongly depends on the chosen heuristic, which is often difficult to design, and node expansion can be very expensive when using a large set of primitives, because it requires the evaluation of all possible successors.

An alternative option is to use a randomized approach such as a variant of the Rapidly-exploring Random Tree (RRT) algorithm. This has been applied in simple 3D environments [12], showing good performance both in planning and replanning for dynamic environments. Clearly the disadvantage of RRT over a deterministic approach is that it does not account, at least in its basic form, for the quality of the footstep plan.

2.1.3 Sensor-based locomotion

So far, it was assumed that a complete knowledge of the environment is available from the start, or that, in case of dynamic environment, changes to the latter are readily communicated to the planner. However, this is not often the case in practical

situations. In fact, the environment could be unknown, either partially or completely, and it must be reconstructed online with the aid of on-board sensors.

Many existing methods exploit information acquired through on-board sensors to identify planar surfaces that define safe regions where the robot can step onto. Such environment representation was used in combination with different kinds of footstep planners, for example, based on simple geometric criteria [13], A* [14] and MIQP [15]. Other methods maintain a more complete representation of the environment by employing an elevation map. Examples can be found in [16, 17], where ARA*-based approaches are used to plan footsteps on uneven ground; the use of on-line information is aimed at improving the plan during the execution, and not for replanning/extension using newly acquired information. To achieve more flexibility in the on-line capabilities, [18] proposed to use adaptive sets of possible foot displacements in an A*-based planner, which proved to be effective in relatively simple scenarios. Alternatively, [19] proposed a two-stage method that first finds a collision-free path for a bounding occupancy volume and then computes a compatible sequence of footsteps, which is a suitable technique as long as it is not necessary to traverse narrow passages.

2.1.4 Footstep and timing adaptation

From Humanoids 2023:

Humanoid robot locomotion is a complex task that involves multiple concurrent activities. It is usually tackled by breaking it down into several subproblems and solving each of them more or less independently. The first component is in general a footstep planner, which determines a sequence of footstep, e.g., leading the robot to some desired location. This sequence of footsteps must be kinematically realizable at least in terms of step lengths. The humanoid dynamics are usually accounted for in a second stage, typically based on Model Predictive Control (MPC), using a simplified robot model which is used to generate Center of Mass (CoM) trajectories. MPC, in its basic form, allows to perform real-time footstep position adaptation [20] and obtain reactive stepping so to reject pushes and impacts. However, in order to be able to formulate the optimization problem as a Quadratic Program (QP), constraints should be kept linear. For this reason, most schemes only adapt footstep positions, leaving out footstep orientation and step timing.

Several efforts to improve this basic paradigm have been made. To include automatic step timing adaptation, one could make the MPC nonlinear [21, 22, 23, 24], denying real-time implementation or requiring significant compromise in the control rate. A linear formulation is obtainable by considering only the duration of the first footstep [25, 26]. As for footstep orientation, this is also often ignored or planned independently of the dynamics [20]. To couple rotation decision with the dynamics, some schemes employ non-convex optimization through nonlinear [27, 28] or Mixed-Integer Programming (MIP) [21]. MIP can also be used to alternatively select between multiple convex regions in which to place the footsteps, which would otherwise constitute a non-convex constraint [29, 9].

2.2 Motion control for steerable wheeled mobile robots

From SWMR paper: Mobile robots equipped with multiple steerable wheels (Fig. 8.1) have greater maneuverability than other wheeled mobile robots, since they are omnidirectional [30]. Besides, they can transport higher payloads than omnidirectional robots equipped with mecanum wheels or with omni wheels. Nevertheless, modeling and controlling these robots is not trivial due to the presence of kinematic singularities [31], which need to be handled with particular care, in order to avoid negatively affecting their functionalities.

While many different approaches for modeling and control of steerable wheeled mobile robots (SWMRs) exist in literature, none of them fully exploits their potentialities. The main property of this kind of robots, indeed, is that their instanteneous center of rotation (ICR) can be located anywhere on the plane [32]. This naturally leads to a parametrization based on two-dimensional cartesian [33] or polar coordinates [34], which, however, leads to singularities that can make it difficult to develop a control scheme. Sorour et al. [31] developed an ICR-based controller which handles singularities of the steering axes. The work is further improved in [35], where the singularity of the ICR at infinity is taken into account through a complementary route strategy. While these approaches consider all singularities of their parametrization, the velocity and acceleration bounds are only considered at the level of the ICR, often resulting in undesired motions with high velocity and high acceleration of the steerable wheels. A singularity-free representation is presented in [36] and [37], and used in of [38], where a free-of-singularity motion controller is developed. Here, time scaling is performed to satisfy velocity and acceleration constraints on the wheels, resulting however in non-optimal motion execution.

Part I

Motion generation for humanoid robots

Chapter 3

Dynamics of humanoid locomotion

Humanoid locomotion is based on exchanging contact forces with the environment, which, thanks to friction, allows the robot to move. In order for the humanoid to successfully complete locomotion tasks, such as walking or running, the motion of the robot must be dynamically balanced, i.e., the robot must not fall.

To formally define the condition for dynamic balance, it is important to understand the dynamics of humanoid locomotion. The motion of humanoid robots is characterized by a sequence of foot contact phases. For example, a walking gait is composed by *double support* and *single support* phases. During double support, both feet are in contact with the ground. During single support, only one foot is in contact with the ground, while the other one (the *swing foot*) moves towards the subsequent footstep location. Each single and double support phase has a duration, which determines how quickly the robot walks. Moreover, the swing foot motion is described by a swing foot trajectory, which must be collision free. Single and double support durations are chosen accordingly to the desired behavior and the phyisical limitations of the robot.

The problem of generation of a walking gait therefore depends on the computation of contact forces that keeps the robot balanced during its motion. In particular, contact forces must be such that they lie within the friction cone relative to their contact surface, in order for the rigid body in contact not to slip. In literature, this condition is often simplified by assuming sufficient (or infinite) friction, and it allows to focus only on unilaterality of contact. This makes it possible to define a condition on the zero-tilting moment point (ZMP), which must lie within a *support region* during locomotion. This support region is called *support polygon* when walking on flat ground, and it is defined as the convex hull of all contact points.

In this chapter, we will define a condition of dynamic balance that can be used for walking over terrains composed of stairs. To do so, we will first introduce the *Lagrangian dynamics* of the humanoids, which fully describes the evolution of the system through exchange of forces with the environment. We will then consider the *centroidal dynamics* of the system, which is the dynamics of the humanoid project at the center of mass (CoM). The centroidal dynamics allows us to introduce template models such as the Variable-Height Inverted Pendulum (VH-IP) and Linear Inverted

Pendulum (LIP), which can be used to generate a walking gait in real-time. In particular, in this manuscript the LIP model will play a fundamental role, as it will be used to design a control framework for humanoid walking on stairs.

3.1 Lagrangian dynamics

Consider a humanoid robot as an open kinematic chain composed of $n+1$ rigid bodies (*links*), connected by n *joints*. We define the *joint configuration* of the robot[1] as $\boldsymbol{q}_j \in (\mathrm{SO}(2))^n$, and assume they are actuated by a torque $\boldsymbol{\tau} \in \mathbb{R}^n$. The root of the tree defining the kinematic chain represents the *floating base* link, as it is not fixed to the world frame. The *floating base configuration* is parametrized by $\boldsymbol{q}_b \in \mathrm{SE}(3)$, and it is composed by the position $\boldsymbol{p}_b \in \mathbb{R}^3$ and the orientation $\boldsymbol{\theta}_b \in \mathrm{SO}(3)$ of the floating base itself. We define the configuration of the robot as

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_b \\ \boldsymbol{q}_j \end{bmatrix} \in \mathrm{SE}(3) \times (\mathrm{SO}(2))^n .$$

As already mentioned before, humanoid robots move by interacting with the environment through the exchange of contact forces. Let $\boldsymbol{f}_k \in \mathbb{R}^3$ be the unilateral contact force exchanged with a contact surface with normal $\boldsymbol{n}_k$ at the k-th contact point $\boldsymbol{p}_k \in \mathbb{R}^3$. Unilaterality of k-th contact is formally defined by

$$\boldsymbol{n}_k^\top \boldsymbol{f}_k > 0,$$

and it describes the impossibility of pulling on the ground. Moreover, we assume constant contact phase, hence not considering impact dynamics, which would make the model hybrid.

The equation of motion of the humanoid [39] is characterized by the following Lagrangian dynamics:

$$\begin{bmatrix} \boldsymbol{M}_u \\ \boldsymbol{M}_a \end{bmatrix} \ddot{\boldsymbol{q}} + \begin{bmatrix} \boldsymbol{c}_u(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ \boldsymbol{c}_a(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \end{bmatrix} + \sum_{k=1}^{K} \begin{bmatrix} \boldsymbol{J}_{k,u}^\top \\ \boldsymbol{J}_{k,a}^\top \end{bmatrix} \boldsymbol{f}_k, \tag{3.1}$$

where $\boldsymbol{M}$ is the inertia matrix, $\boldsymbol{b}$ encodes Coriolis, centrifugal and gravitational terms, and $\boldsymbol{J}_k$ is the contact Jacobian relative to the k-th contact point $\boldsymbol{p}_k$. Note that the structure of the above dynamics highlights the *underactuated* (u subscript) and the *actuated* (a subscript) dynamics of the system, which respectively describe the evolution of the floating base and the joints.

Note that, in order for the dynamics (3.1) to be consistent, the contact mode must be fixed [40], (which implies that bodies in contact must not roll or slide). A contact force $\boldsymbol{f}_k$ is *feasible* if it lies in the friction cone $\mathcal{C}_k$ directed by the contact normal $\boldsymbol{n}_k$:

$$\|\boldsymbol{f}_k - (\boldsymbol{f}_k \cdot \boldsymbol{n}_k)\boldsymbol{n}_k\|_2 \leq \mu_k(\boldsymbol{f}_k \cdot \boldsymbol{n}_k) \tag{3.2}$$

with μ_k static friction coefficient.

In the following we will assume that there always exists joint torques $\boldsymbol{\tau}$ that realize the actuated part of eq. (3.1).

[1] Note that, while this definition of joint configuration considers humanoids composed only by revolute joints, the presented analysis is valid also for robots composed by prismatic joints, or a combination of both.

3.2 Centroidal dynamics

The above hypothesis allows as to focus on the unactuated part of the equation (3.1), and define the *centroidal dynamics* [41] of the humanoid:

$$\begin{bmatrix} m\ddot{\boldsymbol{p}}_c \\ \dot{\boldsymbol{L}}_c \end{bmatrix} = \begin{bmatrix} m\boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix} + \sum_{k=1}^{K} \begin{bmatrix} \boldsymbol{f}_k \\ (\boldsymbol{p}_c - \boldsymbol{p}_k) \times \boldsymbol{f}_k \end{bmatrix}, \tag{3.3}$$

where m is the total mass of the robot, $\boldsymbol{p}_c$ is the position of its center of mass (CoM), defined as

$$\boldsymbol{p}_c = \frac{\sum_{i=1}^{n+1} m_i \boldsymbol{p}_{l_i}}{\sum_{i=1}^{n+1} m_i} = \frac{\sum_{i=1}^{n+1} m_i \boldsymbol{p}_{l_i}}{m},$$

with $\boldsymbol{p}_{l_i}$ and m_i are respectively the position and the mass of the i-th link, $\boldsymbol{g} = (0\ 0\ -g)^\top$ is the gravity vector ($g = 9.81$ [m/s^2]), $\boldsymbol{f}_k$ is the contact force applied at a point with coordinates $\boldsymbol{p}_k$ over a contact surface with normal $\boldsymbol{n}_k$, K is the total number of contacts, and $\boldsymbol{L}_c$ is the angular momentum of the robot taken at the CoM.

Let us define the *gravito-inertial wrench* taken at point O as

$$\boldsymbol{w}_O^{\text{gi}} = \begin{bmatrix} \boldsymbol{f}^{\text{gi}} \\ \boldsymbol{\tau}_O^{\text{gi}} \end{bmatrix} = \begin{bmatrix} m\boldsymbol{g} - m\ddot{\boldsymbol{p}}_c \\ (\boldsymbol{p}_c - \boldsymbol{p}_O) \times (m\boldsymbol{g} - m\ddot{\boldsymbol{p}}_c) - \dot{\boldsymbol{L}}_c \end{bmatrix}. \tag{3.4}$$

Similarly, the *contact wrench* $\boldsymbol{w}_O^{\text{c}}$ can be defined as

$$\boldsymbol{w}_O^{\text{c}} = \begin{bmatrix} \boldsymbol{f}^{\text{c}} \\ \boldsymbol{\tau}_O^{\text{c}} \end{bmatrix} = \sum_{k=1}^{K} \begin{bmatrix} \boldsymbol{f}_k \\ (\boldsymbol{p}_k - \boldsymbol{p}_O) \times \boldsymbol{f}_k \end{bmatrix}. \tag{3.5}$$

Note that the centroidal dynamics (3.3) can be rewritten as a sum of the two above wrenches

$$\boldsymbol{w}_O^{\text{gi}} + \boldsymbol{w}_O^{\text{c}} = 0. \tag{3.6}$$

3.3 Zero-tilting moment point

Consider the gravito-inertial wrench defined in 3.4. Zero-tilting moment points (ZMPs) are points z where the moment of the contact wrench aligns with the normal $\boldsymbol{n}$ of the contact surface [42], i.e.,

$$\boldsymbol{\tau}_z^{\text{gi}} \times \boldsymbol{n} = \boldsymbol{0}, \tag{3.7}$$

which, using Varignon formula[2], can be rewritten as

$$\left(\boldsymbol{\tau}_O^{\text{gi}} + (\boldsymbol{p}_O - \boldsymbol{p}_z) \times \boldsymbol{f}^{\text{gi}}\right) \times \boldsymbol{n} = \boldsymbol{0},$$

[2]A screw $\boldsymbol{w}_O = (\boldsymbol{f}, \boldsymbol{\tau}_O)$ represents the generalized force acting on a rigid body [43], and it is composed by a linear force $\boldsymbol{f}$ passing through O, together with the total moment $\boldsymbol{\tau}_O$ about O. That total moment around any other point A can be computed using Varignon formula as $\boldsymbol{\tau}_A = \boldsymbol{\tau}_O + \boldsymbol{f} \times (\boldsymbol{p}_A - \boldsymbol{p}_O)$.

which, developing the triple cross product[3], the above equation becomes

$$\boldsymbol{\tau}_O^{\text{gi}} \times \boldsymbol{n} - (\boldsymbol{n} \cdot \boldsymbol{f}^{\text{gi}})(\boldsymbol{p}_O - \boldsymbol{p}_z) - (\boldsymbol{n} \cdot (\boldsymbol{p}_O - \boldsymbol{p}_z))\, \boldsymbol{f}^{\text{gi}} = \boldsymbol{0}. \tag{3.8}$$

Assuming that a point Z lies on a plane with normal $\boldsymbol{n}$ intersecting the point O, i.e. $z \in \Pi(O, n)$, the term $\boldsymbol{n} \cdot (\boldsymbol{p}_O - \boldsymbol{p}_z) = 0$, and the above equation can be easily rewritten as

$$\boldsymbol{p}_z = \boldsymbol{p}_O + \frac{\boldsymbol{n} \times \boldsymbol{\tau}_O^{\text{gi}}}{\boldsymbol{n} \cdot \boldsymbol{f}^{\text{gi}}}, \tag{3.9}$$

finally defining the ZMP z. Note that, more in general, there exists an infinity of ZMPs which lie on the non-central axis defined by (3.7). For more details, please refer to [42].

3.3.1 Relationship between CoM, ZMP and angular momentum

Consider the non-tilting condition of Eq. (3.7). Using Varignon formula $\boldsymbol{\tau}_z^{\text{gi}} = \boldsymbol{\tau}_c^{\text{gi}} + \boldsymbol{f}^{\text{gi}} \times (\boldsymbol{p}_z - \boldsymbol{p}_c)$, we have that

$$\left(\boldsymbol{\tau}_c^{\text{gi}} + \boldsymbol{f}^{\text{gi}} \times (\boldsymbol{p}_z - \boldsymbol{p}_c)\right) \times \boldsymbol{n} = \boldsymbol{0}, \tag{3.10}$$

which, computing the triple product, becomes

$$\boldsymbol{\tau}_c^{\text{gi}} \times \boldsymbol{n} - (\boldsymbol{n} \cdot (\boldsymbol{p}_z - \boldsymbol{p}_c))\, \boldsymbol{g}^{\text{gi}} + (\boldsymbol{n} \cdot \boldsymbol{f}^{\text{gi}})(\boldsymbol{p}_z - \boldsymbol{p}_c) = \boldsymbol{0}. \tag{3.11}$$

Applying the definition of *gravito-inertial wrench* of eq. (3.4) and rearranging the terms, it is simple to prove [44] the following relationship between the CoM acceleration, the ZMP position and the angular momentum:

$$\ddot{\boldsymbol{p}}_c = \boldsymbol{g} + \frac{\boldsymbol{n} \cdot (\ddot{\boldsymbol{p}}_c - \boldsymbol{g})}{\boldsymbol{n} \cdot (\boldsymbol{p}_c - \boldsymbol{p}_z)}(\boldsymbol{p}_c - \boldsymbol{p}_z) + \frac{\boldsymbol{n} \times \dot{\boldsymbol{L}}_c}{m\,(\boldsymbol{n} \cdot (\boldsymbol{p}_c - \boldsymbol{p}_z))}. \tag{3.12}$$

3.4 Inverted pendulum models

3.4.1 Variable-Height Inverted Pendulum

Consider again the centroidal dynamics of eq. (3.3) and assume that the rate of change of angular momentum is negligible (i.e., $\dot{\boldsymbol{L}}_c = \boldsymbol{0}$). We define the dynamics of the *Variable-Height Inverted Pendulum* (VH-IP) [45] as

$$m\ddot{\boldsymbol{p}}_c = m\boldsymbol{g} + \boldsymbol{f}^{\text{c}}. \tag{3.13}$$

Note that, because $\dot{\boldsymbol{L}}_c = \boldsymbol{0}$, the contact force $\boldsymbol{f}^{\text{c}}$ can be parametrized [46] as

$$\boldsymbol{f}^{\text{c}} = m\lambda(t)(\boldsymbol{p}_c - \boldsymbol{p}_z), \tag{3.14}$$

with $\lambda(t)$ natural frequency of the VH-IP (where we explicitely denote the time dependency of lambda to highlight the nonlinearity in the dynamics). Note that $\lambda(t) > 0$ because of unilaterality of contact. The dynamics of the VH-IP can be rewritten as

$$\ddot{\boldsymbol{p}}_c = \lambda(t)(\boldsymbol{p}_c - \boldsymbol{p}_z) + \boldsymbol{g} \tag{3.15}$$

by plugging the contact force (3.14) into eq. (3.13).

[3]The triple cross product between three vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathrm{R}^n$ is defined as the cross product of the vector $\boldsymbol{a}$ with the cross product of the other two: $\boldsymbol{a} \times (\boldsymbol{b} \times \boldsymbol{c}) = (\boldsymbol{a} \cdot \boldsymbol{c})\boldsymbol{b} - (\boldsymbol{a} \cdot \boldsymbol{b})\boldsymbol{c}$. Note that, since the cross product is anticommutative, the following holds: $(\boldsymbol{a} \times \boldsymbol{b}) \times \boldsymbol{c} = -(\boldsymbol{c} \cdot \boldsymbol{b})\boldsymbol{a} + (\boldsymbol{c} \cdot \boldsymbol{a})\boldsymbol{b}$.

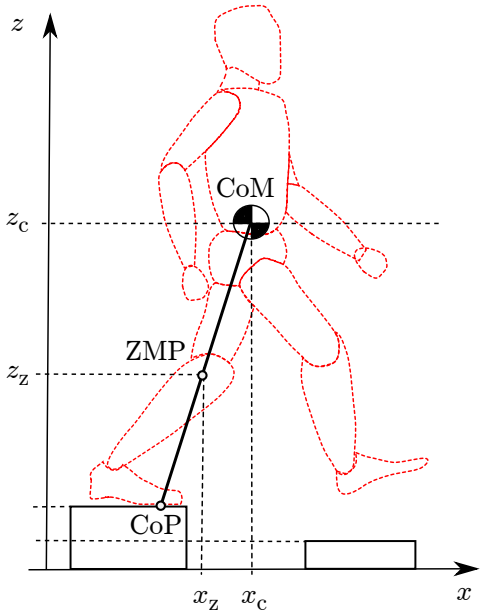

Figure 3.1. A humanoid walking under LIP dynamics. Note the positions of the CoM, the ZMP and the CoP, which lie on the same axis because of the assumption of conservation of angular momentum.

3.4.2 Linear Inverted Pendulum

The dynamics of the VH-IP, as already mentioned, is nonlinear due to the variable frequency $\lambda(t)$. In this section, we derive the dynamics of the *Linear Inverted Pendulum* (LIP) [47]. To do so, we constrain the vertical motion of the CoM [4] so that

$$\lambda(t) = \frac{\ddot{z}_c + g}{z_c - z_z} = \frac{\boldsymbol{n} \cdot (\ddot{\boldsymbol{p}}_c - \boldsymbol{g})}{\boldsymbol{n} \cdot (\boldsymbol{p}_c - \boldsymbol{p}_z)} = \eta^2, \tag{3.16}$$

with η an arbitrary constant. In this way, the dynamics (3.15) becomes the dynamics of the LIP:

$$\ddot{\boldsymbol{p}}_c = \eta^2(\boldsymbol{p}_c - \boldsymbol{p}_z) + \boldsymbol{g}, \tag{3.17}$$

which is in equilibrium when the CoM and the ZMP are displaced by $\boldsymbol{g}/\eta^2$ [48]. Here, the gravity vector $\boldsymbol{g}$ acts as a constant drift. Figure 3.1 shows the humanoid walking under the above dynamics.

When walking on flat floor (a single contact surface with normal $\boldsymbol{e}_z = (0\ 0\ 1)^\top$), a common choice is to constrain the ZMP to lie on a plane which coincides with the ground (i.e., without loss of generality $z_z = 0$). As a consequence, the CoM is constrained to lie on a parallel plane displaced by h from the ZMP plane (i.e., $z_c = h$), and the LIP dynamics further simplifies to the following dynamics:

$$\begin{aligned} \ddot{x}_c &= \eta^2(x_c - x_z) \\ \ddot{y}_c &= \eta^2(y_c - y_z). \end{aligned} \tag{3.18}$$

Note that, in this particular case, the ZMP coincides with the center of pressure (CoP), which is the point of application of the contact force $\boldsymbol{f}^{\mathrm{c}}$ on the ground [42].

The dynamics of eq. (3.17) is unstable [49]. Indeed, by rewriting it in state space form as

$$\begin{bmatrix} \dot{\boldsymbol{p}}_c \\ \ddot{\boldsymbol{p}}_c \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \boldsymbol{I} \\ \eta^2 \boldsymbol{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_c \\ \dot{\boldsymbol{p}}_c \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\eta^2 \boldsymbol{I} \end{bmatrix} \boldsymbol{p}_z + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{g} \end{bmatrix}, \tag{3.19}$$

it is possible to see that the state matrix has eigenvalues in $\pm\eta$. The instability of the system can be further highlight by the following change of coordinates:

$$\begin{bmatrix} \boldsymbol{p}_u \\ \boldsymbol{p}_s \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \frac{1}{\eta}\boldsymbol{I} \\ \boldsymbol{I} & -\frac{1}{\eta}\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_c \\ \dot{\boldsymbol{p}}_c \end{bmatrix}, \tag{3.20}$$

which can be used to decouple the unstable and the stable subsystem. Indeed, by taking the time derivative of eq. (3.20), we have

$$\begin{bmatrix} \dot{\boldsymbol{p}}_u \\ \dot{\boldsymbol{p}}_s \end{bmatrix} = \begin{bmatrix} \eta\boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & -\eta\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_u \\ \boldsymbol{p}_s \end{bmatrix} + \begin{bmatrix} -\eta\boldsymbol{I} \\ \eta\boldsymbol{I} \end{bmatrix} \boldsymbol{p}_z + \begin{bmatrix} \frac{\boldsymbol{g}}{\eta} \\ -\frac{\boldsymbol{g}}{\eta} \end{bmatrix}$$

The coordinate $\boldsymbol{p}_u$ highlights the unstable component of the system (3.17), and it is referred to as *divergent component of motion* [50] or *capture point* [51].

3.5 Contact equilibrium

We are interested in defining the condition of equilibrium of a humanoid robot. While there exists a general condition on contact equilibrium based on contact wrench cone [52], in this manuscript we assume that the friction coefficient defined in (3.2) is sufficiently large to avoid slipping at the contact surfaces.

This hypothesis allows us to focus on a simplified contact equilibrium condition, which is relatively easy to deal with when developing a locomotion scheme. In particular, consider the case of LIP dynamics of eq. (3.17) with all contact forces directed towards the CoM, as described in [53]. In this case, the contact forces can be rewritten as

$$\boldsymbol{f}_k = \frac{\boldsymbol{p}_c - \boldsymbol{p}_k}{\|\boldsymbol{p}_c - \boldsymbol{p}_k\|_2} f_k, \tag{3.21}$$

with $f_k > 0$ the norm of $\boldsymbol{f}_k$. By plugging eq. (3.21) into eq. (3.3), we obtain

$$m\ddot{\boldsymbol{p}}_c = m\boldsymbol{g} + \sum_{k=1}^{K} \frac{\boldsymbol{p}_c - \boldsymbol{p}_k}{\|\boldsymbol{p}_c - \boldsymbol{p}_k\|_2} f_k. \tag{3.22}$$

Moreover, using the dynamics of eq. (3.17), we can obtain obtain (after rearranging the terms), the following:

$$\boldsymbol{p}_z = \boldsymbol{p}_c - \sum_{k=1}^{K} \frac{\boldsymbol{p}_c - \boldsymbol{p}_k}{\|\boldsymbol{p}_c - \boldsymbol{p}_k\|_2} \frac{f_k}{m\eta^2}.$$

Because of the assumption of sufficient joint torque actuation made in Sec. 3.1, in principle, we could modulate f_k to position the ZMP within the area defined by the following set:

$$\mathcal{Z} = \left\{ \boldsymbol{p}_z \;\middle|\; \boldsymbol{p}_z = \boldsymbol{p}_c + \sum_{k=1}^{K} \gamma_k (\boldsymbol{p}_k - \boldsymbol{p}_c),\ \gamma_k > 0 \right\}, \tag{3.23}$$

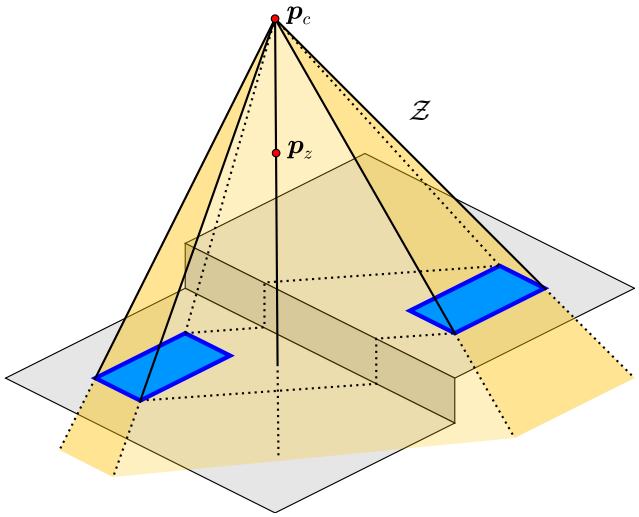

Figure 3.2. 3D balance: the ZMP $\boldsymbol{p}_z$ must be inside the polyhedral cone $\mathcal{Z}$.

which is a polyhedral cone with apex $\boldsymbol{p}_c$ (Fig. 3.2), and is known in literature as *support region* [2], as it contains all feasible positions of the ZMP. Note that under dynamics (3.18), the support region becomes the *support polygon*, which is defined as the convex hull of all contact points [42].

Despite the above hypothesis on contact forces seems a strong assumption, it has been used successfully for both walking and running tasks [53, 4, 2, 54]. Moreover, when in single support (or when all contacts are coplanar), the above condition is always satisfied if $\dot{\boldsymbol{L}}_c = \boldsymbol{0}$ [55].

Chapter 4

Gait generation via IS-MPC

In the last chapter, we have given an overview on humanoid locomotion dynamics, focusing in particular on the model of the Linear Inverted Pendulum (LIP). We have described its dynamics, we have seen that it is unstable, and we have given a condition for contact equilibrium on non-coplanar surfaces for the humanoid. In this chapter, we exploit the previously presented analysis to develop a gait generation scheme for humanoid locomotion in a *world of stairs*, an uneven terrain where all contact surfaces are piecewise horizontal.

The approach adopted in this manuscript is based on Intrinsically-Stable MPC (IS-MPC) [49], a model predictive control scheme which, given as input a *footstep plan* (which defines the desired motion of the robot at high-level specifying desired footstep positions, orientations, single and double support durations, and swing foot trajectories), generates a stable center of mass (CoM) trajectory, which can be tracked by a whole-body controller. In IS-MPC, where the model is the one of the LIP, stability is guaranteed via a stability constraint, which bounds the displacement between the CoM and the zero-tilting moment point (ZMP). Moreover, dynamic balance is enforced via constraints on the ZMP position. In particular, since the polyhedral cone of eq. (3.23) would result in a nonlinear constraint, it is conservatively approximated with a convex region. Because both constraints are linear, IS-MPC can be formulated as a Quadratic Programming (QP) program, and solved efficiently at each control iteration.

This chapter presents IS-MPC in its entirety, describing the prediction model (a LIP with a dynamic extension on the ZMP), the *stability constraint* and the *ZMP constraint*, and the definition of the QP problem. In the end, the condition of feasibility of IS-MPC optimization problem is discussed.

4.1 Overview

In this section, we describe the use of IS-MPC within a simple locomotion framework (Fig. 4.1). Before going into the details of the scheme, we define the *footstep plan* $\mathcal{P} = (\mathcal{S}_f, \mathcal{S}_\varphi)$ as the sequence of footsteps $\mathcal{S}_f$ and swing foot trajectories $\mathcal{S}_\varphi$ that bring the robot from its initial configuration to a desired goal. In particular, we

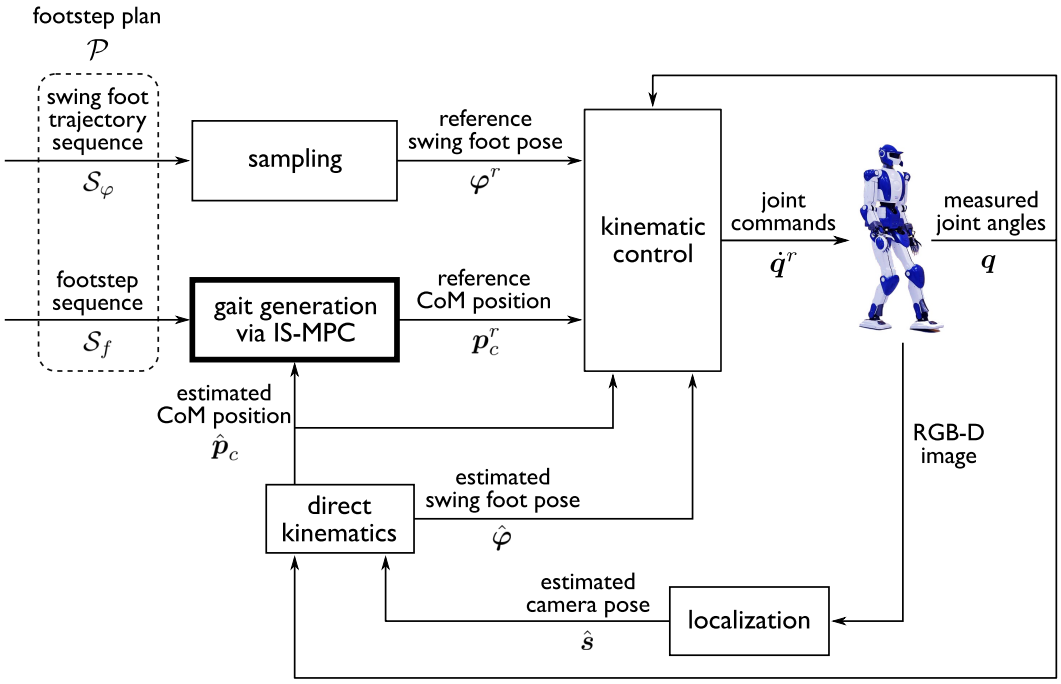

Figure 4.1. Block scheme showing the use of IS-MPC within a locomotion framework.

define

$$\mathcal{S}_f = \left\{ \boldsymbol{f}^1, \dots, \boldsymbol{f}^n \right\}$$
$$\mathcal{S}_\varphi = \left\{ \boldsymbol{\varphi}^1, \dots, \boldsymbol{\varphi}^{n-2} \right\},$$

with

$$\boldsymbol{f}^j = \left(x_f^j, y_f^j, z_f^j, \theta_f^j, T_{\text{ss}}^j, T_{\text{ds}}^j \right)$$

collecting the position (x_f^j, y_f^j, z_f^j) and the orientation θ_f^j of the j-th footstep[1], the duration T_{ss}^j of the j-th single support phase, and the duration T_{ds}^j of the j-th double support phase. The swing foot trajectory $\boldsymbol{\varphi}^j$ represents the j-th step, i.e., the trajectory leading the foot from $\boldsymbol{f}^j$ to $\boldsymbol{f}^{j+2}$. Note that $(x_f^1, y_f^1, z_f^1, \theta_f^1)$ and $(x_f^2, y_f^2, z_f^2, \theta_f^2)$ respectively represent the initial swing foot and support foot pose.

Assuming a footstep plan $\mathcal{P}$ is available (it may be manually defined or computed by a footstep planner [48]), at each timestep t_k, the IS-MPC computes a reference position $\boldsymbol{p}_c^r$ of the CoM, which, together with the reference swing foot pose $\boldsymbol{\varphi}^r$, is sent to the kinematic controller which generates joint commands $\dot{\boldsymbol{q}}^r$ that are sent to the robot. In this manuscript, we assume the robot is controlled in velocity, and that the kinematic controller is QP-based (such as [56]). The robot is then localized (using sensors mounted on the robot, such as an RGD-D camera as shown in the block scheme), the estimated CoM position $\hat{\boldsymbol{p}}_c$ is fed back to to IS-MPC, and the estimated swing foot pose is fed back to the kinematic controller.

[1]To represent the footstep orientation we only use the yaw angle, as roll and pitch are always zero thanks to the piecewise-horizontal ground assumption in the *world of stairs*.

4.2 Prediction model

As already mentioned in Sect. 3.4.2, control of the ZMP can be achieved by using the LIP model, which relates the position of the ZMP and the acceleration of the CoM. In order to obtain smoother trajectories, we dynamically extend the LIP model (3.17), obtaining a model with derivative of the ZMP $\dot{\boldsymbol{p}}_z$ as control input. Denoting the CoM as $\boldsymbol{p}_c = (x_c\ y_c\ z_c)^\top$ and the ZMP as $\boldsymbol{p}_z = (x_z\ y_z\ z_z)^\top$, the model can hence be rewritten as

$$\begin{bmatrix} \dot{x}_c \\ \ddot{x}_c \\ \dot{x}_z \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \eta^2 & 0 & -\eta^2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ \dot{x}_c \\ x_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \dot{x}_z \tag{4.1}$$

for the x component (and analogously for the y component), and

$$\begin{bmatrix} \dot{z}_c \\ \ddot{z}_c \\ \dot{z}_z \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \eta^2 & 0 & -\eta^2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_c \\ \dot{z}_c \\ z_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \dot{z}_z + \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} \tag{4.2}$$

for the z component.

The IS-MPC gait generation scheme works over discrete time-steps of duration δ, over which the input $\dot{\boldsymbol{p}}_z$ is assumed to be constant, i.e.

$$\dot{\boldsymbol{p}}_z(t) = \dot{\boldsymbol{p}}_z^k, \quad t \in [t_k, t_{k+1}).$$

This assumption makes the ZMP piecewise linear over the IS-MPC sampling intervals, i.e.,

$$x_z(t) = x_z^i + x_z^i(t - t_i), \quad t \in [t_k, t_{k+1}), \tag{4.3}$$

for the x component (an analogous relation can be written for the y and z components).

The prediction model is used to forecast the evolution of the system over a receding horizon window called the *control horizon*, spanning a time $T_c = C\delta$. The number of footsteps that are contained within this control horizon is denoted as F. In this manuscript, we consider that the available footstep plan fully covers the receding horizon window. To do so, we assume that the robot comes to a complete stop once the last element (representing the goal) is reached. When this hypothesis is not satisfied, it is possible to consider a receding window called the *preview horizon* [49].

Consider the following vectors collecting the ZMP positions:

$$\boldsymbol{X}_z^k = (x_z^k\ x_z^{k+1}\ \dots\ x_z^{k+C-1})^\top \tag{4.4}$$

$$\boldsymbol{Y}_z^k = (y_z^k\ y_z^{k+1}\ \dots\ y_z^{k+C-1})^\top \tag{4.5}$$

$$\boldsymbol{Z}_z^k = (z_z^k\ z_z^{k+1}\ \dots\ z_z^{k+C-1})^\top. \tag{4.6}$$

As explained in [57], by defining

$$\boldsymbol{P} = \begin{bmatrix} \delta & 0 & \cdots & 0 \\ \delta & \delta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \delta & \delta & \cdots & \delta \end{bmatrix}, \qquad \boldsymbol{p} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

it is possible to express ZMP positions as

$$\begin{aligned}\boldsymbol{X}_z^{k+1} &= \boldsymbol{p}x_z^k + \boldsymbol{P}\dot{\boldsymbol{X}}_z^k \\ \boldsymbol{Y}_z^{k+1} &= \boldsymbol{p}y_z^k + \boldsymbol{P}\dot{\boldsymbol{Y}}_z^k \\ \boldsymbol{Z}_z^{k+1} &= \boldsymbol{p}z_z^k + \boldsymbol{P}\dot{\boldsymbol{Z}}_z^k.\end{aligned}$$

This definition will be used to express the stability constraint and the ZMP constraint in the following sections. Moreover, the vectors $\dot{\boldsymbol{X}}_z^k, \dot{\boldsymbol{Y}}_z^k, \dot{\boldsymbol{Z}}_z^k$ will be the optimization variables in the IS-MPC optimization problem, which, as mentioned before, is a QP problem.

4.3 Stability constraint

Model (3.17), and consequently the dynamically extended model (4.2), as already mentioned in Sect. 3.4.2, has a positive eigenvalue η, reflecting the intrinsic instability of the humanoid dynamics. Given this instability, it is not sufficient to generate a gait such that the ZMP is inside the support region (3.23), because the associated CoM trajectory might be divergent, making the motion unrealizable by the humanoid. The role of the stability constraint is to enforce a condition on theù unstable component of the dynamics in order to guarantee that the CoM trajectory does not diverge with respect to the ZMP.

Despite the instability, the evolution of the system is bounded if the following *stability condition* [58] is satisfied:

$$\boldsymbol{p}_u^k = \eta \int_{t_k}^{\infty} e^{-\eta(\tau - t_k)} \boldsymbol{p}_z(\tau) d\tau - \frac{\boldsymbol{g}}{\eta^2}, \tag{4.7}$$

where the superscript in $\boldsymbol{p}_u^k$ indicates that the variable is sampled at time t_k.

Condition (4.7) is non-causal as it requires knowledge of the future ZMP trajectory $\boldsymbol{p}_z$ up to infinity. In order to derive a causal implementation, we split the integral at t_{k+C}. Of the two separate integrals that result, the first, over $[t_k, t_{k+C})$, can be expressed in terms of the MPC decision variables. A value for the second integral, over $[t_{k+C}, \infty)$, can be obtained by conjecturing a ZMP trajectory using information coming from the footstep plan. This conjectured trajectory is called *anticipative tail* and is denoted with $\tilde{\boldsymbol{p}}_z$. In [49], the anticipative tail was used to prove recursive feasibility and stability of the MPC scheme.

The stability constraint is then written as

$$\eta \int_{t_k}^{t_{k+C}} e^{-\eta(\tau - t_k)} \boldsymbol{p}_z d\tau = \boldsymbol{p}_u^k - \tilde{\boldsymbol{c}}^k + \frac{\boldsymbol{g}}{\eta^2}. \tag{4.8}$$

where $\tilde{\boldsymbol{c}}^k$ is given by

$$\tilde{\boldsymbol{c}}^k = \eta \int_{t_{k+C}}^{\infty} e^{-\eta(\tau - t_k)} \tilde{\boldsymbol{p}}_z d\tau. \tag{4.9}$$

Enforcing constraint (4.8) allows to bound the displacement between CoM and ZMP. In fact, the value of the bound is almost identical in most practical situation, especially in view of the fact that the preview horizon is unlimited because the plan is completely known.

The stability constraint (4.8) can be rewritten in terms of $\dot{\boldsymbol{X}}_z^k, \dot{\boldsymbol{Y}}_z^k, \dot{\boldsymbol{Z}}_z^k$ by computing the integral over the piecewise linear ZMP trajectory (4.3). The final form of the constraint can be found in [49]. For the purpose of this analysis, we will use the compact expression

$$\begin{bmatrix} \boldsymbol{s}^\top & \boldsymbol{0}^\top & \boldsymbol{0}^\top \\ \boldsymbol{0}^\top & \boldsymbol{s}^\top & \boldsymbol{0}^\top \\ \boldsymbol{0}^\top & \boldsymbol{0}^\top & \boldsymbol{s}^\top \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{X}}_z^k \\ \dot{\boldsymbol{Y}}_z^k \\ \dot{\boldsymbol{Z}}_z^k \end{bmatrix} = \boldsymbol{b}^k + \boldsymbol{p}_u^k \tag{4.10}$$

where

$$\boldsymbol{s} = \frac{1 - e^{-\eta\delta}}{\eta} \begin{bmatrix} e^{-0\cdot\eta\delta} \\ e^{-1\cdot\eta\delta} \\ \dots \\ e^{-(C-1)\cdot\eta\delta} \end{bmatrix}, \quad \boldsymbol{b}^k = \begin{bmatrix} b_x^k \\ b_y^k \\ b_z^k \end{bmatrix} = -\boldsymbol{p}_z^k - \begin{bmatrix} 0 \\ 0 \\ g/\eta^2 \end{bmatrix}. \tag{4.11}$$

Note that, from 3.20, we have that

$$\boldsymbol{p}_u^k = \begin{bmatrix} x_u^k \\ y_u^k \\ z_u^k \end{bmatrix} = \boldsymbol{p}_c^k + \frac{1}{\eta}\dot{\boldsymbol{p}}_c^k. \tag{4.12}$$

4.4 ZMP constraint

As explained in Chapter 3, relating the dynamics of the CoM to those of the ZMP is essential since the latter encodes information about the realizability of ground reaction forces, and thus provides a criterion for balance. A common criterion for balance in 3D environments such as *world of stairs* is to prescribe the ZMP to be inside a polyhedral cone (Sect. 3.5). However, enforcing this condition directly would lead to a nonlinear constraint in the MPC because the vertex of the pyramid is the CoM of the robot [55]. Thus, we adopt a conservative approximation called the *moving constraint* [59].

The IS-MPC block constructs ZMP constraints from the footstep plan $\mathcal{P}^l$. The moving constraint requires for the ZMP to be at all times within a convex polyhedron of fixed shape, in our case a box of dimensions d_x, d_y and d_z centered in $\boldsymbol{p}_{\text{mc}} = (x_{\text{mc}}, y_{\text{mc}}, z_{\text{mc}})^\top$, which we call the *moving box*. Note that approximating the polyhedral cone $\mathcal{Z}$ with a box might seem overly conservative. However, we argue that the neglected portion of the pyramid region is not crucial here, because large displacements of the ZMP in the z direction would only be required to generate large vertical accelerations, which are not necessary in the considered setting (walking in a world of stairs). Clearly, less conservative approximations can still be envisaged and used for generating more dynamic motions.

Along the prediction, the moving box can translate but not rotate, and its center moves in such a way that it is always fully contained within the 3D pyramid $\mathcal{Z}$ (Fig 4.2). The vectors

$$\begin{aligned} \boldsymbol{X}_{\text{mc}}^{k+1} &= (x_{\text{mc}}^{k+1} \; x_{\text{mc}}^{k+2} \; \dots \; x_{\text{mc}}^{k+C})^\top \\ \boldsymbol{Y}_{\text{mc}}^{k+1} &= (y_{\text{mc}}^{k+1} \; y_{\text{mc}}^{k+2} \; \dots \; y_{\text{mc}}^{k+C})^\top \\ \boldsymbol{Z}_{\text{mc}}^{k+1} &= (z_{\text{mc}}^{k+1} \; z_{\text{mc}}^{k+2} \; \dots \; z_{\text{mc}}^{k+C})^\top \end{aligned}$$

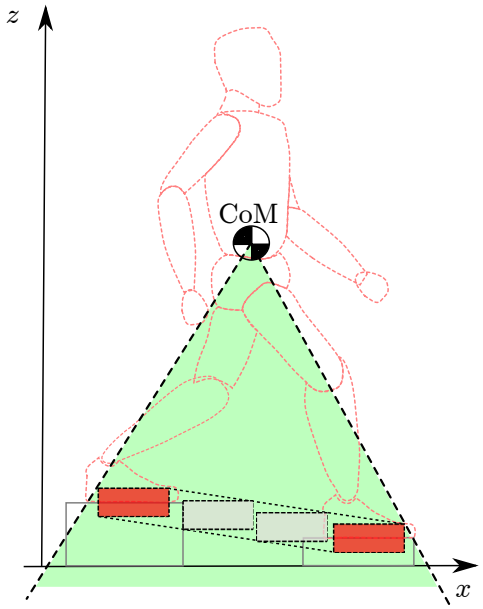

Figure 4.2. During double support, the box constraint slides from the previous to the next support foot [4].

collect the coordinates of the center of the moving box in the control horizon.

Because of its constant orientation in the prediction, at each time we can choose the orientation of the axes to align with the orientation of the moving box (taken as the orientation of the current support foot) and obtain a ZMP constraint that is decoupled along the 3 axes. We can write it as

$$\begin{aligned} \boldsymbol{X}_z^{\mathrm{m,k+1}} \leq \boldsymbol{X}_z^{k+1} \leq \boldsymbol{X}_z^{M,k+1} \\ \boldsymbol{Y}_z^{\mathrm{m,k+1}} \leq \boldsymbol{Y}_z^{k+1} \leq \boldsymbol{Y}_z^{M,k+1} \\ \boldsymbol{Z}_z^{\mathrm{m,k+1}} \leq \boldsymbol{Z}_z^{k+1} \leq \boldsymbol{Z}_z^{M,k+1}, \end{aligned} \tag{4.13}$$

where $\boldsymbol{X}_z^{\mathrm{m,k+1}}, \boldsymbol{X}_z^{\mathrm{M,k+1}}, \boldsymbol{Y}_z^{\mathrm{m,k+1}}, \boldsymbol{Y}_z^{\mathrm{M,k+1}}, \boldsymbol{Z}_z^{\mathrm{m,k+1}}, \boldsymbol{Z}_z^{\mathrm{M,k+1}}$ are the ZMP bounds along the prediction, which can be expressed as

$$\begin{aligned} \boldsymbol{X}_z^{\mathrm{m,k+1}} &= \boldsymbol{X}_{\mathrm{mc}}^{k+1} - \boldsymbol{z}\frac{d_x}{2} & \boldsymbol{X}_z^{\mathrm{M,k+1}} &= \boldsymbol{X}_{\mathrm{mc}}^{k+1} + \boldsymbol{z}\frac{d_x}{2} \\ \boldsymbol{Y}_z^{\mathrm{m,k+1}} &= \boldsymbol{Y}_{\mathrm{mc}}^{k+1} - \boldsymbol{z}\frac{d_y}{2} & \boldsymbol{Y}_z^{\mathrm{M,k+1}} &= \boldsymbol{Y}_{\mathrm{mc}}^{k+1} + \boldsymbol{z}\frac{d_y}{2} \\ \boldsymbol{Z}_z^{\mathrm{m,k+1}} &= \boldsymbol{Z}_{\mathrm{mc}}^{k+1} - \boldsymbol{z}\frac{d_z}{2} & \boldsymbol{Z}_z^{\mathrm{M,k+1}} &= \boldsymbol{Z}_{\mathrm{mc}}^{k+1} + \boldsymbol{z}\frac{d_z}{2}. \end{aligned} \tag{4.14}$$

The size of the moving box $(d_x, d_y, d_z)^\top$ is determined in such a way to always be contained inside the pyramid $\mathcal{Z}$ [4, 48]. The center of the moving box $\boldsymbol{p}_{\mathrm{mc}}$ must be expressed in terms of the subplan $\mathcal{P}^l$. First we define the *piecewise-linear sigmoid* function

$$\sigma(t, t_i, t_f) = \frac{1}{t_f - t_i}\left(\rho(t - t_i) - \rho(t - t_f)\right),$$

where $\rho(t) = t\delta_{-1}(t)$ is the unit ramp. $\sigma(t, t_i, t_f)$ is 0 before t_i, 1 after t_f, and it transitions linearly in the interval $[t_i, t_f]$. This function is useful to represent the transition between consecutive footsteps.

The vectors $\boldsymbol{X}_{\text{mc}}^{k+1}, \boldsymbol{Y}_{\text{mc}}^{k+1}, \boldsymbol{Z}_{\text{mc}}^{k+1}$ collecting the positions of the moving constraint along the horizon can be written as

$$\begin{aligned} \boldsymbol{X}_{\text{mc}}^{k+1} &= \boldsymbol{M}\boldsymbol{X}_f^l + \boldsymbol{m}x_f^l \\ \boldsymbol{Y}_{\text{mc}}^{k+1} &= \boldsymbol{M}\boldsymbol{Y}_f^l + \boldsymbol{m}y_f^l \\ \boldsymbol{Z}_{\text{mc}}^{k+1} &= \boldsymbol{M}\boldsymbol{Z}_f^l + \boldsymbol{m}z_f^l \end{aligned} \tag{4.15}$$

where

$$\begin{aligned} \boldsymbol{X}_f^l &= (x_f^l \; x_f^{l+1} \; \dots \; x_f^{l+F})^\top \\ \boldsymbol{Y}_f^l &= (y_f^l \; y_f^{l+1} \; \dots \; y_f^{l+F})^\top \\ \boldsymbol{Z}_f^l &= (z_f^l \; z_f^{l+1} \; \dots \; z_f^{l+F})^\top \end{aligned}$$

collect the footstep positions. $\boldsymbol{M} \in \mathbb{R}^{C \times F}$ is a mapping matrix whose elements M_{ij} are defined as

$$\begin{aligned} M_{ij} = \sigma(t_{k+i}, t_s^{l+j}, t_s^{l+j} + T_{\text{ds}}^{l+j}) \\ - \sigma(t_{k+i}, t_s^{l+j-1}, t_s^{l+j-1} + T_{\text{ds}}^{l+j-1}), \end{aligned} \tag{4.16}$$

and $\boldsymbol{m} \in \mathbb{R}^C$ is a vector whose elements m_i are given by

$$m_i = 1 - \sigma(t_{k+i}, t_s^l, t_s^l + T_{\text{ds}}^1),$$

where t_s^l is the starting time of the l-th step and

$$t_s^j = t_s^l + \sum_{\lambda=l}^{l+j-1} \left(T_{\text{ds}}^\lambda + T_{\text{ss}}^\lambda \right).$$

4.5 IS-MPC algorithm

IS-MPC solves, at each time t_k, the following QP problem:

$$\begin{cases} \min\limits_{\dot{\boldsymbol{X}}_z^k, \dot{\boldsymbol{Y}}_z^k, \dot{\boldsymbol{Z}}_z^k} \|\dot{\boldsymbol{X}}_z^k\|^2 + \|\dot{\boldsymbol{Y}}_z^k\|^2 + \|\dot{\boldsymbol{Z}}_z^k\|^2 + \beta\|\boldsymbol{X}_z^{k+1} - \boldsymbol{X}_{\text{mc}}^{k+1}\|^2 \\ \qquad + \beta\|\boldsymbol{Y}_z^{k+1} - \boldsymbol{Y}_{\text{mc}}^{k+1}\|^2 + \beta\|\boldsymbol{Z}_z^{k+1} - \boldsymbol{Z}_{\text{mc}}^{k+1}\|^2 \\ \text{subject to:} \\ \quad \bullet \text{ stability constraints (4.10)} \\ \quad \bullet \text{ ZMP constraints (4.13)} \end{cases} \tag{4.17}$$

The algorithm takes as input the footstep plan $\mathcal{P}$, which is used to construct, at each timestep t_k the stability and the ZMP constraints, as explained in the previous sections. In the cost function, the first three terms act as regularization while the remaining attempt to bring the ZMP as close as possible to the center of the moving box, with a strength modulated by the weight β.

The IS-MPC iteration performs, at each timestep t_k, the following steps:

1. Solve the QP (4.17) to obtain $\dot{\boldsymbol{X}}_z^k, \dot{\boldsymbol{Y}}_z^k, \dot{\boldsymbol{Z}}_z^k$.
2. From the solutions, extract the first sample $(\dot{x}_z^k, \dot{y}_z^k, \dot{z}_z^k)$.
3. Integrate (4.2) from $(x_c^k, \dot{x}_c^k, x_z^k)$ considering $\dot{x}_z = \dot{x}_z^k$, obtaining $(x_c^{k+1}, \dot{x}_c^{k+1}, x_z^{k+1})$. Compute $(y_c^{k+1}, \dot{y}_c^{k+1}, y_z^{k+1})$ and $(z_c^{k+1}, \dot{z}_c^{k+1}, z_z^{k+1})$ similarly (using (4.2) for the z component).
4. Send the CoM position $\boldsymbol{p}_c^{k+1} = (x_c^{k+1}, y_c^{k+1}, z_c^{k+1})^\top$ to the kinematic controller.

4.6 Feasibility region

The *feasibility region* is the region of the state space in which the IS-MPC optimization problem (4.17) is feasible.

Proposition 1 *IS-MPC is feasible at time t_k if*

$$\begin{aligned}
\boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{X}_z^{\mathrm{m},k+1} - \boldsymbol{p}x_z^k) \leq x_u^k + b_x^k \leq \boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{X}_z^{\mathrm{M},k+1} - \boldsymbol{p}x_z^k),\\
\boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{Y}_z^{\mathrm{m},k+1} - \boldsymbol{p}y_z^k) \leq y_u^k + b_y^k \leq \boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{Y}_z^{\mathrm{M},k+1} - \boldsymbol{p}y_z^k),\\
\boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{Z}_z^{\mathrm{m},k+1} - \boldsymbol{p}z_z^k) \leq z_u^k + b_z^k \leq \boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{Z}_z^{\mathrm{M},k+1} - \boldsymbol{p}z_z^k).
\end{aligned} \tag{4.18}$$

Proof. We focus the proof on the inequalities for the x component, as the logic, for the other components is identical. The bounds of the feasibility region along x are given by

$$\begin{aligned}
x_u^{k,b1} &= \boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{X}_z^{\mathrm{m},k+1} - \boldsymbol{p}x_z^k) - b_x^k,\\
x_u^{k,b2} &= \boldsymbol{s}^\top \boldsymbol{P}^{-1}(\boldsymbol{X}_z^{\mathrm{M},k+1} - \boldsymbol{p}x_z^k) - b_x^k.
\end{aligned}$$

Then, if x_u^k is inside the feasibility region, it is possible to express it as a convex combination of the two bounds, i.e.,

$$x_u^k = \alpha x_u^{k,b1} + (1-\alpha) x_u^{k,b2}, \alpha \in [0,1]. \tag{4.19}$$

Consider the following ZMP velocity trajectory:

$$\dot{\boldsymbol{X}}_z^k = \alpha \boldsymbol{P}^{-1}(\boldsymbol{X}_z^{\mathrm{m},k+1} - \boldsymbol{p}x_z^k) + (1-\alpha)\boldsymbol{P}^{-1}(\boldsymbol{X}_z^{\mathrm{M},k+1} - \boldsymbol{p}x_z^k). \tag{4.20}$$

We will show that this particular trajectory satisfies both the stability constraint and the ZMP constraints. As for the stability constraint, multiply both sides of (4.20) by $\boldsymbol{s}^\top$ and plug in the definitions of $x_u^{k,b1}$ and $x_u^{k,b2}$ to obtain

$$\boldsymbol{s}^\top \dot{\boldsymbol{X}}_z^k = (\alpha(x_u^{k,b1} + b_x^k)) + (1-\alpha)(x_u^{k,b2} + b_x^k)).$$

Using (4.19), this is equivalent to the stability constraint (4.10).

To prove satisfaction of the ZMP constraint. Left-multiplying (4.4) by $\boldsymbol{Z}$, the chosen ZMP velocity trajectory can be rewritten as

$$\boldsymbol{X}_z^k - \boldsymbol{p}x_z^k = \alpha(\boldsymbol{X}_z^{\mathrm{m},k+1} - \boldsymbol{p}x_z^k) + (1-\alpha)(\boldsymbol{X}_z^{\mathrm{M},k+1} - \boldsymbol{p}x_z^k),$$

which simplifies to $\boldsymbol{X}_z^k = \alpha \boldsymbol{X}_z^{\mathrm{m},k+1} + (1-\alpha)\boldsymbol{X}_z^{\mathrm{M},k+1}$, and therefore the ZMP constraint (4.13) is satisfied. ∎

Note that, because $\boldsymbol{P}$ is a lower-triangular matrix filled with δ, its inverse is simply

$$\boldsymbol{P}^{-1} = \begin{bmatrix} 1/\delta & 0 & 0 & \dots & 0 & 0 \\ -1/\delta & 1/\delta & 0 & \dots & 0 & 0 \\ 0 & -1/\delta & 1/\delta & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1/\delta & 0 \\ 0 & 0 & 0 & \dots & -1/\delta & 1/\delta \end{bmatrix}.$$

Chapter 5

Humanoid motion generation in a world of stairs

In this chapter we consider the problem of generating a humanoid motion in a *world of stairs*, a specific kind of uneven ground consisting of horizontal patches located at different heights. In order to successfully achieve this, it is necessary to plan a footstep sequence and a whole body motion for the humanoid realizing such sequence. We choose to approach the problem by keeping these two stages separate. The reason for this choice is that in this way we can better control the quality of the produced motions. In fact, the quality of the footstep plan can be evaluated based on global requirements, such as the number of steps taken, or a different performance index. As for the quality of the motion itself, this should be mainly addressed by its capability to satisfy all the required constraints, in order to avoid falls and instabilities. Maintaining a separation between planning and control also greatly simplifies the tuning because the domain in which any given parameter has effect is reduced.

Keeping planning and control separated is relatively straightforward when the planning is done off-line. However, it might be more difficult to realize if one wants to allow for on-line planning and replanning. In this chapter, we will discuss an on-line architecture that achieves this by having short planning stages interleaved with the motion, in such a way that the planning can fully utilize sensor information gathered by the robot as it moves, and that the robot itself can take advantage of the on-line planner to find more efficient and safe footstep sequences. Moreover, we consider planning both in off-line situations, in which the environment is completely known, and on-line situations, in which the geometry of the environment is not known in advance and must be reconstructed by the robot itself during motion using on-board sensors. In this case, planning and execution are interdependent: the former clearly requires the latter in order to determine where to place the footsteps, but the converse is also true, as the real-time motion of the robot is necessary to acquire information about the environment.

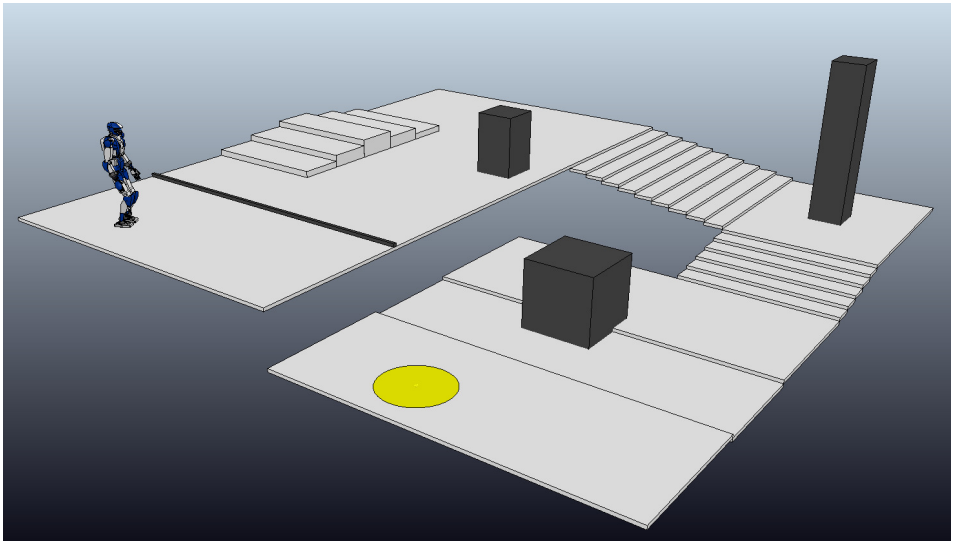

Figure 5.1. An instance of the considered problem. The robot must reach the goal region (in yellow) by traversing a *world of stairs*. Patches that are not visible correspond to infinitely deep holes or trenches.

5.1 Problem formulation

In the situation of interest (Fig. 5.1), a humanoid robot moves in a *world of stairs*, a specific kind of uneven ground consisting of horizontal patches that *(i)* are located at different heights, and *(ii)* constitute a partition[1] of $\mathbb{R}^2$. Depending on its elevation with respect to the neighboring areas, a patch may be accessible for the humanoid to climb on from an appropriate direction; otherwise, it actually represents an *obstacle* to be avoided. Any accessible patch may be stepped on, stepped over or even circumvented, depending on the generated motion.

The mission of the robot is to reach a certain goal region $\mathcal{G}$, which will belong in general to a single patch (Fig. 5.1). In particular, this locomotion task is accomplished as soon as the robot places a footstep inside $\mathcal{G}$.

We want to devise a complete framework enabling the humanoid to plan and execute a motion to fulfill the assigned task in the world of stairs. This requires addressing two fundamental problems: finding a 3D footstep plan and generating a variable-height gait that is consistent with such plan. Footstep planning consists in finding both footstep placements and swing foot trajectories between them; overall, the footstep plan must be feasible (in a sense to be formally defined later) for the humanoid, given the characteristics of the environment. Gait generation consists in finding a CoM trajectory which realizes the footstep plan while guaranteeing dynamic balance of the robot at all time instants. From the trajectories of the CoM and the feet, a whole-body motion may be computed using inverse kinematics methods.

In particular, we will address two versions of the above problem, henceforth referred to as the *off-line* and *on-line* case. In the first, the geometry of the environment is completely known in advance, while in the second it is reconstructed

[1]This implies that the whole volume of space below any horizontal patch is occupied.

by the robot itself as it moves. In the following we describe the architectures designed for the off-line and on-line case supposing that the environment is static. However, we will also show that the latter can be used effectively in dynamic environments, thanks to its fast replanning capabilities.

The problem will be solved under the following assumptions.

A1 Information about the environment is maintained in an *elevation map* $\mathcal{M}_z$, i.e., a 2.5D grid map of equally-sized cells that, whenever needed, can be queried as $z = \mathcal{M}_z(x, y)$, to provide the height of the ground at the cell having coordinates (x, y) [60]. In the off-line case, $\mathcal{M}_z$ is available a priori, while in the on-line case it must be incrementally built by the humanoid based on sensory information.

A2 The humanoid is equipped with a head-mounted RGB-D camera, which is used for localization in both the off-line and on-line cases, and also updating the elevation map $\mathcal{M}_z$ in the latter.

A3 The humanoid is endowed with a localization module which provides an estimate of the camera pose at each time instant, based on information gathered by the RGB-D camera. This is used in both the off-line and on-line cases.

A4 The friction between the robot feet and the ground is sufficiently large to avoid slipping at the contact surfaces[2].

In the off-line case, a complete footstep plan leading to the goal region $\mathcal{G}$ will be computed before the humanoid starts to move. In the on-line case, the footstep plan will instead be updated during motion, based on new information added to $\mathcal{M}_z$.

In the following, we first address in full detail the off-line case and then proceed to extending the proposed approach to the on-line case.

5.2 The off-line case

We start this section by describing the general structure of the proposed method. Then we will describe the footsep planner and present some simulations. The gait generation scheme used is the one presented in Chapter 4.

5.2.1 General architecture

To solve the described problem in the off-line case, we adopt the architecture shown in Fig. 5.2, in which the main components are the footstep planning, gait generation and localization modules.

In the following, we denote by $\boldsymbol{f} = (x_f, y_f, z_f, \theta_f)$ the *pose* of a certain footstep, with x_f, y_f, z_f representing the coordinates of a representative point, henceforth

[2]Since our objective is to generate walking gaits in a world of stairs, we expect that the horizontal components of the ground reaction forces will be rather limited with respect to the vertical components, making this assumption reasonable. Nevertheless, the constraint that ground reaction forces must always be contained in the friction cone can be incorporated in our formulation; this extension would be required in the case of non-horizontal contact surfaces.

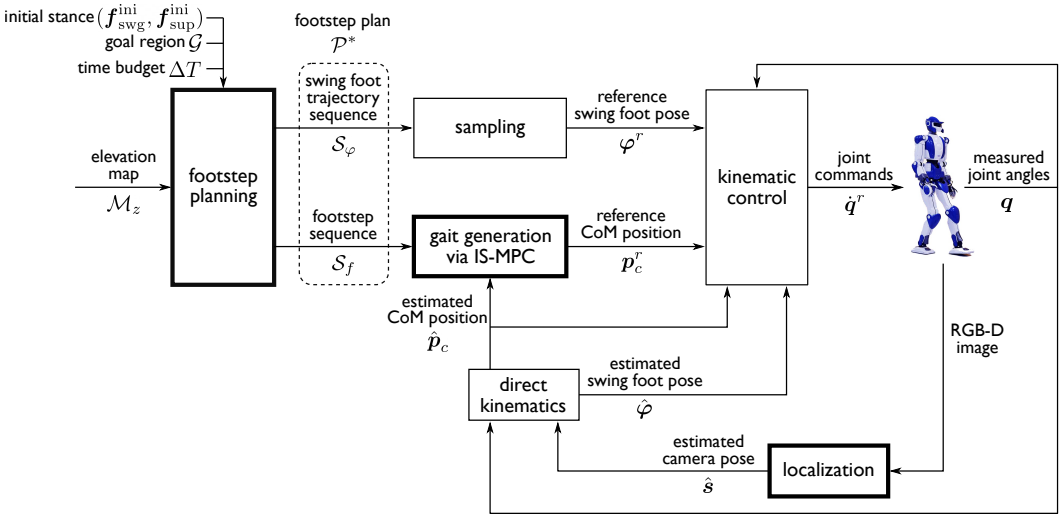

Figure 5.2. Block scheme of the proposed solution approach for the off-line case.

collectively denoted as $\boldsymbol{p}_f$, and θ_f its orientation[3]. Moreover, a pair $(\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}})$ defines a *stance*, i.e., the feet poses during a double support phase after which a step is performed by moving the swing foot from $\boldsymbol{f}_{\text{swg}}$ while keeping the support foot at $\boldsymbol{f}_{\text{sup}}$.

The footstep planner receives in input the initial humanoid stance[4] $(\boldsymbol{f}^{\text{ini}}_{\text{swg}}, \boldsymbol{f}^{\text{ini}}_{\text{sup}})$ at $t = 0$, the goal region $\mathcal{G}$, a time budget ΔT, and the elevation map $\mathcal{M}_z$ representing the environment.

The time budget represents the time given to the planner to find a solution. When this time runs over, the algorithm either returns a solution or ends with a failure. Explicitly specifying this time as input to the algorithm allows us to evaluate the performance of the planning module, but also proves to be useful for the extension to the on-line case, where the time budget is set equal to the duration of a step in order to meet the real-time requirement.

The planner works off-line to find, within ΔT, an optimal *footstep plan* $\mathcal{P}^* = \{\mathcal{S}_f, \mathcal{S}_\varphi\}$ leading to the desired goal region $\mathcal{G}$. In $\mathcal{P}^*$, we denote by

$$\mathcal{S}_f = \{\boldsymbol{f}^1, \ldots, \boldsymbol{f}^n\}$$

the sequence of footstep placements, whose generic element $\boldsymbol{f}^j$ is the pose of the j-th footstep, with $\boldsymbol{f}^1 = \boldsymbol{f}^{\text{ini}}_{\text{swg}}$ and $\boldsymbol{f}^2 = \boldsymbol{f}^{\text{ini}}_{\text{sup}}$. Also, we denote by

$$\mathcal{S}_\varphi = \{\boldsymbol{\varphi}^1, \ldots, \boldsymbol{\varphi}^{n-2}\}$$

the sequence of associated swing foot trajectories, whose generic element $\boldsymbol{\varphi}^j$ is the j-th *step*, i.e., the trajectory leading the foot from $\boldsymbol{f}^j$ to $\boldsymbol{f}^{j+2}$. Its duration T^j_s is split in T^j_{ss} and T^j_{ds}, respectively the single and double support phases. The timestamp of a generic footstep indicates the beginning of the j-th step, i.e., of the j-th single support phase; thus, $\boldsymbol{f}^j$ has an associated timestamp $t^j_s = t^{j-1}_s + T^{j-1}_s$, with $t^1_s = 0$.

[3]To represent the footstep orientation we only use the yaw angle, as roll and pitch are always zero thanks to the piecewise-horizontal ground assumption.

[4]The initial support foot can be chosen arbitrarily.

Once the footstep plan has been generated, the sequence of footsteps $\mathcal{S}_f$ is sent to a gait generator based on IS-MPC (Chapter 4), which computes in real time a variable-height CoM trajectory that is compatible with $\mathcal{S}_f$ and guaranteed to be *stable* (i.e., bounded with respect to the ZMP). In particular, we denote by $\boldsymbol{p}_{\mathrm{c}}^r$ the current reference position of the CoM produced by IS-MPC. Also, let $\boldsymbol{\varphi}^r$ be the current reference pose of the swing foot, obtained by sampling the appropriate subtrajectory in $\mathcal{S}_\varphi$. Then, $\boldsymbol{p}_{\mathrm{c}}^r$ and $\boldsymbol{\varphi}^r$ are passed to a kinematic controller which computes the joint commands $\dot{\boldsymbol{q}}^r$ for the robot.

Visual information, gathered by the head-mounted camera in the form of RGB-D images, is provided to the localization module, which continuously updates the estimate $\hat{\boldsymbol{s}}$ of the pose of the camera frame. From this, and using joint encoder data, it is possible to obtain estimates for the CoM position and swing foot pose ($\hat{\boldsymbol{p}}_c$ and $\hat{\boldsymbol{\varphi}}$, respectively) through kinematic computations. Finally, these estimates are used to provide feedback to both the gait generation and kinematic control modules.

5.2.2 Footstep planning

The input data for this module are the initial robot stance $(\boldsymbol{f}_{\mathrm{swg}}^{\mathrm{ini}}, \boldsymbol{f}_{\mathrm{sup}}^{\mathrm{ini}})$, the goal region $\mathcal{G}$, the time budget ΔT and the elevation map $\mathcal{M}_z$. Given an optimality criterion, the footstep planner returns the best footstep plan $\mathcal{P}^*$ leading to $\mathcal{G}$ found within ΔT.

The planning algorithm builds a tree $\mathcal{T}$, where each vertex $v = (\boldsymbol{f}_{\mathrm{swg}}, \boldsymbol{f}_{\mathrm{sup}})$ specifies a stance, and an edge between two vertexes v and $v' = (\boldsymbol{f}'_{\mathrm{swg}} = \boldsymbol{f}_{\mathrm{sup}}, \boldsymbol{f}'_{\mathrm{sup}})$ indicates a step between the two stances, i.e., a collision-free trajectory such that one foot swings from $\boldsymbol{f}_{\mathrm{swg}}$ to $\boldsymbol{f}'_{\mathrm{sup}}$ and the other is fixed at $\boldsymbol{f}_{\mathrm{sup}}$.

The expansion process makes use of a catalogue U of *primitives*, which allows to generate new footsteps by selecting them from a finite set of displacements with respect to the support foot. The catalogue is split in two subsets, one for the case of left support and the other for the case of right support; at each instant, the appropriate subset is used. An example catalogue is shown in Fig. 5.3.

Each branch joining the root of the tree to a generic vertex v represents a footstep plan $\mathcal{P}$. The sequences $\mathcal{S}_f$ and $\mathcal{S}_\varphi$ for this plan are respectively obtained by taking along the branch the support foot poses of all vertexes and the steps corresponding to all edges.

Footstep feasibility

Footstep $\boldsymbol{f}^j = (x_f^j, y_f^j, z_f^j, \theta_f^j) \in \mathcal{S}_f$ is *feasible* if it satisfies the following requirements:

R1 $\boldsymbol{f}^j$ is fully in contact within a single horizontal patch.

To guarantee this, each cell of $\mathcal{M}_z$ belonging to, or overlapping with, the footprint at $\boldsymbol{f}^j$ must have the same height z_f^j. In practice, one typically uses an enlarged footprint to ensure that this requirement will still be satisfied in the presence of small positioning errors.

R2 $\boldsymbol{f}^j$ is kinematically admissible from the previous footstep $\boldsymbol{f}^{j-1}$ (this is actually *stance feasibility*).

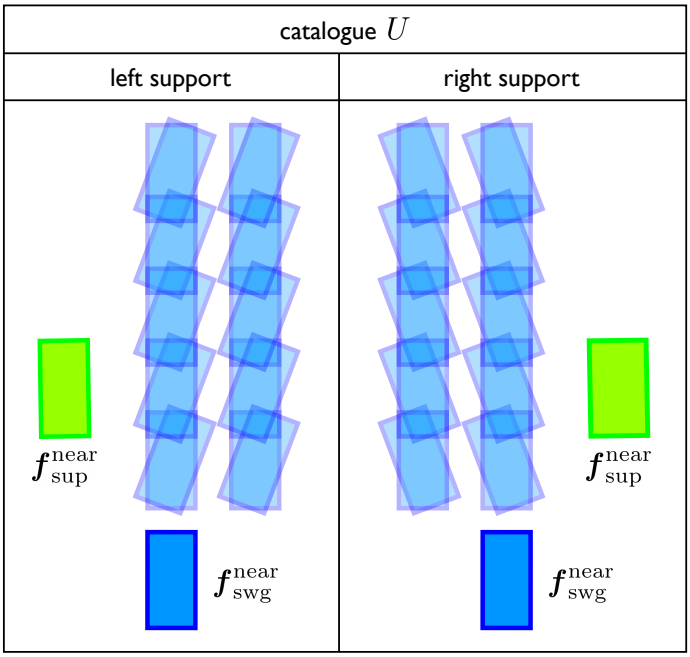

Figure 5.3. An example catalogue U of primitives, containing a finite set of landings for the swing foot with respect to the left or right support foot.

The admissible region (a submanifold of $\mathbb{R}^3 \times SO(2)$) for placing $\boldsymbol{f}^j$ next to $\boldsymbol{f}^{j-1}$ is defined by the following constraints:

$$-\begin{pmatrix}\Delta_x^-\\ \Delta_y^-\end{pmatrix} \leq R_{j-1}^\top \begin{pmatrix}x_f^j - x_f^{j-1}\\ y_f^j - y_f^{j-1}\end{pmatrix} \pm \begin{pmatrix}0\\ \ell\end{pmatrix} \leq \begin{pmatrix}\Delta_x^+\\ \Delta_y^+\end{pmatrix} \tag{5.1}$$

$$-\Delta_z^- \leq z_f^j - z_f^{j-1} \leq \Delta_z^+ \tag{5.2}$$

$$-\Delta_\theta^- \leq \theta_f^j - \theta_f^{j-1} \leq \Delta_\theta^+. \tag{5.3}$$

Here, R_{j-1} is the planar rotation matrix associated with θ_f^{j-1} and the Δ symbols are lower and upper maximum increments, see Fig. 5.4.

R3 $\boldsymbol{f}^j$ is reachable from $\boldsymbol{f}^{j-2}$ through a collision-free motion (this is actually *step feasibility*).

Since information about the whole-body motion of the robot is not yet available during footstep planning (it will only be defined in the subsequent gait generation phase), this requirement can only be tested conservatively. In particular, we say that R3 is satisfied if (*i*) there exist a collision-free swing foot trajectory $\boldsymbol{\varphi}^{j-2}$ from $\boldsymbol{f}^{j-2}$ to $\boldsymbol{f}^j$ generated by the engine of Sect. 5.2.2, and (*ii*) a suitable volume $\mathcal{B}$ accounting for the maximum occupancy of the humanoid upper body at stance $(\boldsymbol{f}^{j-1}, \boldsymbol{f}^j)$ is collision-free. More precisely, $\mathcal{B}$ is a vertical cylinder whose base has radius r_b and center at $(x_m, y_m, z_m + z_b)$, where x_m, y_m, z_m are the coordinates of the midpoint $\boldsymbol{m}$ between the footsteps and z_b is a vertical offset representing the average distance between the ground and the

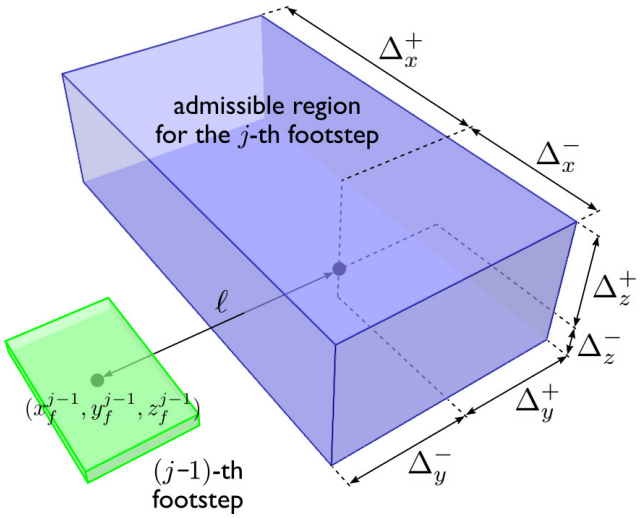

Figure 5.4. The 3D admissible region identified by the first two kinematic constraints of requirement R2, i.e., eqs. (5.1– 5.2). Footstep orientation is not represented.

hip (Fig. 5.5). Note that any nonzero height can be used for the cylinder in view of the world of stairs assumption, which implies that $\mathcal{B}$ is collision-free [5] if each cell of $\mathcal{M}_z$ belonging to, or overlapping with, its ground projection has height smaller than $z_m + z_b$.

Vertex identity, neighbors and cost

The *identity* of a vertex $v = (\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}})$ indicates whether $\boldsymbol{f}_{\text{swg}}$ refers to the left (L) or the right (R) foot:

$$\text{id}(v) = \begin{cases} L & \text{if} \quad \text{id}(v^{\text{parent}}) = R \\ R & \text{if} \quad \text{id}(v^{\text{parent}}) = L, \end{cases}$$

where v^{parent} is the parent vertex of v. This definition determines the identity of each vertex, once the identity of the root is assigned.

We also define the set of *neighbors* of $v = (\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}})$

$$\mathcal{N}(v) = \{v' = (\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}) \in \mathcal{T} : \gamma(\boldsymbol{f}_{\text{sup}}, \boldsymbol{f}'_{\text{sup}}) \leq r_{\text{neigh}}\}$$

where r_{neigh} is a threshold distance and

$$\gamma(\boldsymbol{f}, \boldsymbol{f}') = \|\boldsymbol{p}_f - \boldsymbol{p}'_f\| + k_\gamma |\theta_f - \theta'_f| \tag{5.4}$$

is a footstep-to-footstep metric, with $k_\gamma \geq 0$.

[5] Even though the volume for checking the upper body collision is chosen conservatively, this does not guarantee obstacle avoidance because the lower body is not considered. However, whole-body collision avoidance can be obtained by including appropriate constraints in the kinematic controller [56].

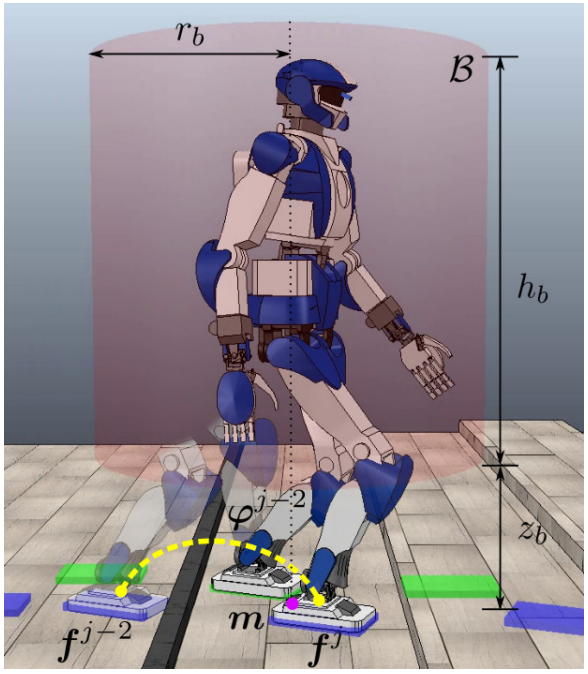

Figure 5.5. Visual representation of the process for checking requirement R3. The red cylinder accounts for the maximum occupancy of the humanoid upper body, while the yellow line represents the swing foot trajectory.

Assume that the edge between two vertexes v^a and v^b of $\mathcal{T}$ has a cost $l(v^a, v^b)$. The *cost* of a vertex v is defined as

$$c(v) = c(v^{\text{parent}}) + l(v^{\text{parent}}, v)$$

and represents the cost of reaching v from the root of $\mathcal{T}$. Then, the cost of a plan $\mathcal{P}$ ending at a vertex v is $c(\mathcal{P}) = c(v)$.

In particular, we will consider three possibilities for the cost of an edge. The first is

$$l_1(v^a, v^b) = 1, \tag{5.5}$$

for all edges in $\mathcal{T}$. The corresponding vertex cost will be denoted by $c_1(v)$ and represents the length of the corresponding plan in terms of number of edges (i.e., steps).

The second edge cost represents the net height variation of the swing foot during a step:

$$l_2(v^a, v^b) = |z^b_{f,\text{sup}} - z^a_{f,\text{swg}}|, \tag{5.6}$$

where $z^a_{f,\text{swg}}$ and $z^b_{f,\text{sup}}$ are the z-component of, respectively, the swing foot at v^a and the support foot at v^b. The corresponding vertex cost will be denoted by $c_2(v)$ and represents the cumulative height variation along the corresponding plan.

Finally, we also consider as edge cost

$$l_3(v^a, v^b) = \frac{1}{\sigma(\boldsymbol{f}^b_{\text{sup}})}, \tag{5.7}$$

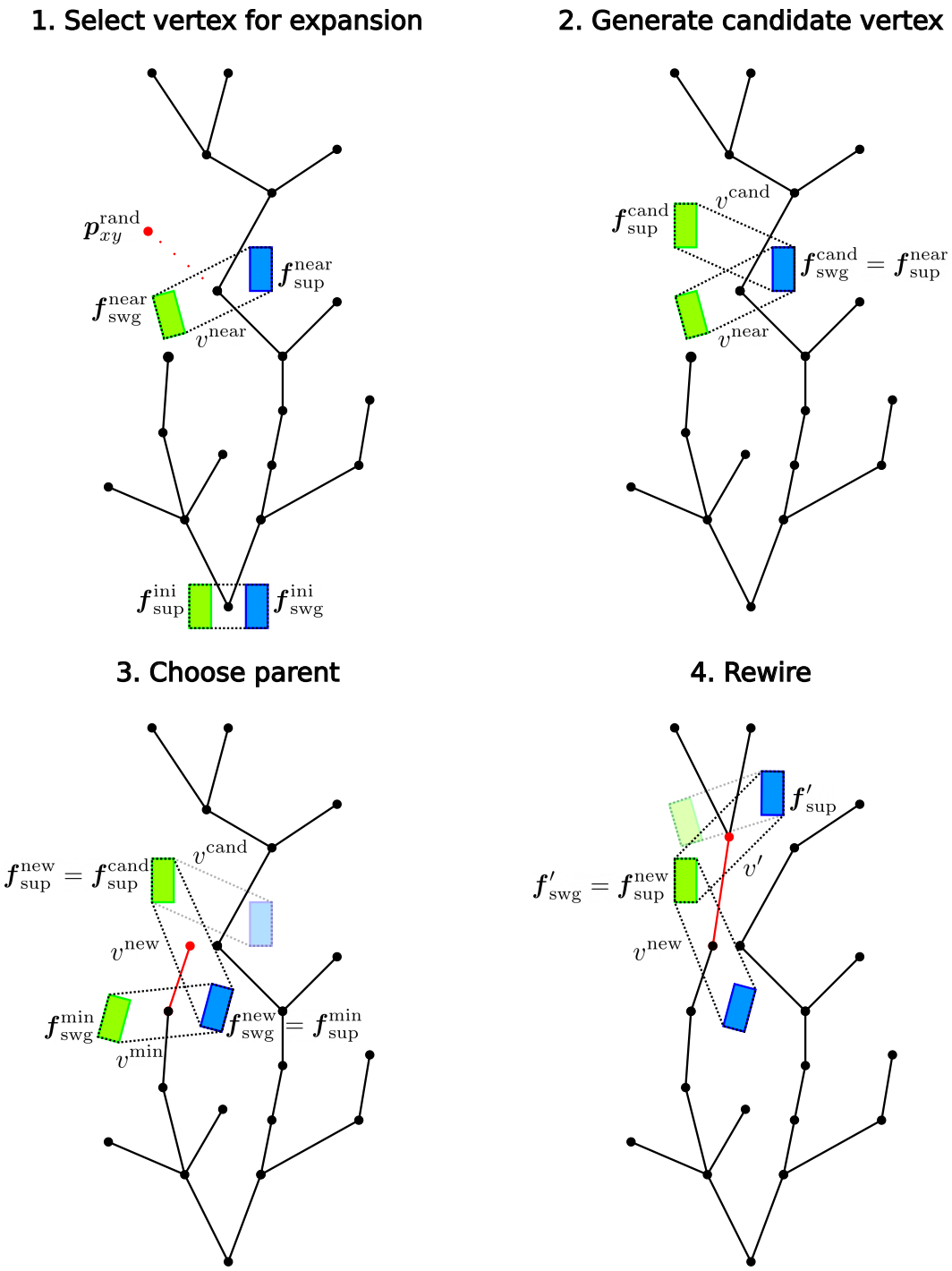

Figure 5.6. The four steps of a generic iteration of the footstep planner. First, a sampled point is used to select a vertex for expansion. Then, the selected vertex is used to generate a candidate vertex to be inserted in the tree. In the third step, a parent for the candidate vertex is chosen. Finally, a rewiring procedure is called to update the tree, guaranteeing that the cost of the nodes is optimal.

where $\sigma(\boldsymbol{f}^b_{\text{sup}})$ is the *clearance* of the support foot $\boldsymbol{f}^b_{\text{sup}}$ at v^b, defined as the distance between the representative point of $\boldsymbol{f}^b_{\text{sup}}$ and the closest point in $\mathcal{M}_z$ w.r.t. which the absolute height variation is larger than $\max\{\Delta_z^-, \Delta_z^+\}$[6]. This cost penalizes steps that bring the swing foot too close to a drop or to a vertical surface leading to a contiguous higher patch, while still allowing to approach accessible patches such as staircases. The corresponding vertex cost, denoted by $c_3(v)$, represents the cumulative inverse clearance along the corresponding plan.

Other kinds of cost functions can be considered. For example, one could penalize unnecessary rotations of the next footstep with respect to the support footstep in order to obtain smoother plans. In general, it may be advisable to use a weighted combination of several optimality criteria for better practical performance.

Algorithm

We now describe the footstep planning algorithm for the off-line case (Algorithm 1).

At the beginning, $\mathcal{T}$ is rooted at $(\boldsymbol{f}^{\text{ini}}_{\text{swg}}, \boldsymbol{f}^{\text{ini}}_{\text{sup}})$, the initial stance of the humanoid. Then, $\mathcal{T}$ is expanded using an RRT*-like strategy. The generic iteration consists of: selecting a vertex for expansion, generating a candidate vertex, choosing a parent for the new vertex and rewiring the tree. These individual steps are described in the following, see also Fig. 5.6.

Algorithm 1: FootstepPlanner

Input: $(\boldsymbol{f}^{\text{ini}}_{\text{swg}}, \boldsymbol{f}^{\text{ini}}_{\text{sup}}), \mathcal{G}, \Delta T, \mathcal{M}_z$
Output: $\mathcal{P}^*$

```
1 v^ini ← (f^ini_swg, f^ini_sup);
2 AddVertex(T, ∅, v^ini, ∅);
3 ExpandTree(T, ΔT, M_z);
4 P* ← RetrieveBestPlan(T, G);
5 return P*;
```

Selecting a vertex for expansion: A point $\boldsymbol{p}^{\text{rand}}_{xy}$ is randomly selected on the xy-plane, and the vertex v^{near} that is closest to $\boldsymbol{p}^{\text{rand}}_{xy}$ is identified. To this end, we define a vertex-to-point metric as

$$\mu(v, \boldsymbol{p}_{xy}) = \|\boldsymbol{m}_{xy}(v) - \boldsymbol{p}_{xy}\| + k_\mu |\psi(v, \boldsymbol{p}_{xy})|,$$

where $\boldsymbol{m}_{xy}(v)$ represents the planar position of the midpoint between the feet at stance v, k_μ is a positive scalar, and $\psi(v, \boldsymbol{p}_{xy})$ is the angle between the robot sagittal axis (whose orientation is the average of the orientations of the two footsteps) and the line joining $\boldsymbol{m}_{xy}$ to $\boldsymbol{p}_{xy}$. This step corresponds to the functions SamplePoint and NearestVertex in Procedure 1. ◀

Generating a candidate vertex: After identifying the vertex $v^{\text{near}} = (\boldsymbol{f}^{\text{near}}_{\text{swg}}, \boldsymbol{f}^{\text{near}}_{\text{sup}})$, a candidate footstep is first generated using the catalogue U (see Fig. 5.3). In particular, we set the support foot to $\boldsymbol{f}^{\text{near}}_{\text{sup}}$ and randomly select

[6] This information may be precomputed from the elevation map $\mathcal{M}_z$ and stored in a clearance map.

Procedure 1: ExpandTree

Input: $\mathcal{T}, \Delta T, \mathcal{M}_z$

Output: none

1 **while not** TimeExpired(ΔT) **do**
2 $\quad \boldsymbol{p}_{xy}^{\text{rand}} \leftarrow$ SamplePoint();
3 $\quad v^{\text{near}} \leftarrow$ NearestVertex($\mathcal{T}, \boldsymbol{p}_{xy}^{\text{rand}}$);
4 $\quad \boldsymbol{f}_{\text{sup}}^{\text{cand}} \leftarrow$ GenerateCandidateFootstep($\boldsymbol{f}_{\text{sup}}^{\text{near}}, U, \mathcal{M}_z$);
5 $\quad$ **if** R1($\boldsymbol{f}_{\text{sup}}^{\text{cand}}$) **and** R2($\boldsymbol{f}_{\text{sup}}^{\text{cand}}, \boldsymbol{f}_{\text{sup}}^{\text{near}}$) **then**
6 $\quad\quad \boldsymbol{\varphi}^{\text{near}} \leftarrow$ SwingTrajectoryEngine($\boldsymbol{f}_{\text{swg}}^{\text{near}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}}$);
7 $\quad\quad$ **if** R3($\boldsymbol{\varphi}^{\text{near}}, (\boldsymbol{f}_{\text{sup}}^{\text{near}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$) **then**
8 $\quad\quad\quad v^{\text{cand}} \leftarrow (\boldsymbol{f}_{\text{sup}}^{\text{near}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$;
9 $\quad\quad\quad \mathcal{N} \leftarrow$ Neighbors($\mathcal{T}, v^{\text{cand}}$);
10 $\quad\quad\quad (v^{\text{min}}, \boldsymbol{\varphi}^{\text{min}}) \leftarrow$ ChooseParent($\mathcal{T}, \mathcal{N}, v^{\text{near}}, v^{\text{cand}}, \boldsymbol{\varphi}^{\text{near}}$);
11 $\quad\quad\quad v^{\text{new}} \leftarrow (\boldsymbol{f}_{\text{sup}}^{\text{min}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$;
12 $\quad\quad\quad$ AddVertex($\mathcal{T}, v^{\text{min}}, v^{\text{new}}, \boldsymbol{\varphi}^{\text{min}}$);
13 $\quad\quad\quad$ ReWire($\mathcal{T}, \mathcal{N}, v^{\text{new}}$);
14 $\quad\quad$ **end**
15 $\quad$ **end**
16 **end**
17 **return**;

one element from the subset of U associated to the identity of v^{near}, which may be L (left) or R (right). Note that all elements of U are chosen so as to automatically satisfy conditions (5.1–5.3) of requirement R2. Call $\boldsymbol{f}_{\text{sup}}^{\text{cand}}$ the chosen candidate footstep. The z coordinate to be associated to the footstep $\boldsymbol{f}_{\text{sup}}^{\text{cand}}$ is then retrieved from $\mathcal{M}_z$, and both the requirement R1 and the last condition (5.2) of R2 can now be checked.

To test the last requirement R3, we invoke an engine (Procedure 2) that generates a swing foot trajectory $\boldsymbol{\varphi}^{\text{near}}$ from $\boldsymbol{f}_{\text{swg}}^{\text{near}}$ to $\boldsymbol{f}_{\text{sup}}^{\text{cand}}$. Such engine uses a parameterized trajectory which, given the endpoints, can be deformed[7] by varying the maximum height h along the motion. Using the elevation map and increments of Δh, the engine tries growing values of h in a certain range $[h^{\text{min}}, h^{\text{max}}]$ looking for a collision-free trajectory.

If all requirements have been satisfied, a candidate vertex is generated as $v^{\text{cand}} = (\boldsymbol{f}_{\text{swg}}^{\text{cand}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$, with $\boldsymbol{f}_{\text{swg}}^{\text{cand}} = \boldsymbol{f}_{\text{sup}}^{\text{near}}$; however, v^{cand} is not added to $\mathcal{T}$ because the planner first needs to identify the best parent for it. If any requirement among R1–R3 is violated, the current expansion attempt is aborted and a new iteration is started. This step corresponds to lines 4-8 in Procedure 1. ◀

Choosing a parent: Although v^{cand} was generated from v^{near}, there might be a different vertex in the tree that leads to the same vertex with a lower cost. To find it, the planner checks for each vertex $v' = (\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}) \in \mathcal{N}(v^{\text{cand}})$ whether setting v' as parent of v^{cand} satisfies requirements R2-R3, and whether this connection reduces

[7] As a deformable trajectory we used a polynomial, but other choices (e.g., B-splines and Bezier curves) are possible.

Procedure 2: SwingTrajectoryEngine

Input: $\boldsymbol{f}^a$, $\boldsymbol{f}^b$
Output: $\boldsymbol{\varphi}^a$

$h \leftarrow h^{\min}$;
while $h \leq h^{\max}$ **do**
  $\boldsymbol{\varphi}^a \leftarrow \text{DeformTrajectory}(\boldsymbol{f}^a, \boldsymbol{f}^b, h)$;
  if $\text{CollisionFree}(\boldsymbol{\varphi}^a)$ **then**
    return $\boldsymbol{\varphi}^a$;
  end
  $h \leftarrow h + \Delta h$;
end
return $\emptyset$;

the cost of v^{cand}, that is

$$c(v') + l(v', v^{\text{cand}}) < c(v^{\text{near}}) + l(v^{\text{near}}, v^{\text{cand}}).$$

The vertex $v^{\min} = (\boldsymbol{f}^{\min}_{\text{swg}}, \boldsymbol{f}^{\min}_{\text{sup}})$ that allows to reach v^{cand} with minimum cost is chosen as its parent. If $v^{\min} = v^{\text{near}}$, then v^{cand} can be added to the tree together with the edge joining it to v^{near}. However, if a different parent $v^{\min} \neq v^{\text{near}}$ is chosen, the candidate vertex v^{cand} must be modified by relocating its swing footstep to the support footstep of $v^{\min}$. To this end, a new vertex $v^{\text{new}} = (\boldsymbol{f}^{\text{new}}_{\text{swg}}, \boldsymbol{f}^{\text{new}}_{\text{sup}})$ with $\boldsymbol{f}^{\text{new}}_{\text{swg}} = \boldsymbol{f}^{\min}_{\text{sup}}$ and $\boldsymbol{f}^{\text{new}}_{\text{sup}} = \boldsymbol{f}^{\text{cand}}_{\text{sup}}$ is generated and added to $\mathcal{T}$ as child of $v^{\min}$. The edge between $v^{\min}$ and v^{new} corresponds to the swing foot trajectory $\boldsymbol{\varphi}^{\min}$. This step corresponds to the function ChooseParent in Procedure 3. ◀

Procedure 3: ChooseParent

Input: $\mathcal{T}, \mathcal{N}, v^{\text{near}}, v^{\text{cand}}, \boldsymbol{\varphi}^{\text{cand}}$
Output: $v^{\min}, \boldsymbol{\varphi}^{\min}$

$v^{\min} \leftarrow v^{\text{near}}$;
$\boldsymbol{\varphi}^{\min} \leftarrow \boldsymbol{\varphi}^{\text{cand}}$;
$c^{\min} \leftarrow c(v^{\text{near}}) + l(v^{\text{near}}, v^{\text{cand}})$;
foreach $v' \in \mathcal{N}$ **do**
  if $\text{R2}(\boldsymbol{f}^{\text{cand}}_{\text{sup}}, \boldsymbol{f}'_{\text{sup}})$ **then**
    $\boldsymbol{\varphi}' \leftarrow \text{SwingTrajectoryEngine}(\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}^{\text{cand}}_{\text{sup}})$;
    if $\text{R3}(\boldsymbol{\varphi}', (\boldsymbol{f}'_{\text{sup}}, \boldsymbol{f}^{\text{cand}}_{\text{sup}}))$ **and** $c(v') + l(v', v^{\text{cand}}) < c^{\min}$ **then**
      $v^{\min} \leftarrow v'$;
      $\boldsymbol{\varphi}^{\min} \leftarrow \boldsymbol{\varphi}'$;
      $c^{\min} \leftarrow c(v') + l(v', v^{\text{cand}})$;
    end
  end
end
return $(v^{\min}, \boldsymbol{\varphi}^{\min})$;

Rewiring: This final step checks whether v^{new} allows to reach with a lower cost some vertex already in $\mathcal{T}$, and updates the tree accordingly. In particular, for each $v' = (\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}) \in \mathcal{N}(v^{\text{new}})$, the procedure checks whether setting v' as a child of

v^{new} satisfies requirements R2-R3, and whether this connection reduces the cost of v', that is

$$c(v^{\text{new}}) + l(v^{\text{new}}, v') < c(v').$$

If this is the case, v' is modified (similarly to what was done when choosing a parent) by relocating its swing footstep to $\boldsymbol{f}_{\text{sup}}^{\text{new}}$, and then reconnected to $\mathcal{T}$ as a child of v_{new}. The edge between v^{new} and v' corresponds to the swing foot trajectory $\boldsymbol{\varphi}^{\text{new}}$. Finally, for each child $v'' = (\boldsymbol{f}''_{\text{swg}}, \boldsymbol{f}''_{\text{sup}})$ of v', the swing foot trajectory $\boldsymbol{\varphi}'$ from the relocated $\boldsymbol{f}'_{\text{swg}}$ to $\boldsymbol{f}''_{\text{sup}}$ is generated and the edge between v' and v'' is accordingly updated[8]. This step corresponds to the function ReWire in Procedure 4.

Note that, although $\mathcal{N}(v^{\text{new}})$ can contain ancestors of v^{new}, no cycle will be generated by rewiring. In fact, it can be easily shown that any ancestor of v^{new} will have a cost lower or equal than $c(v^{\text{new}})$, so that it will never be set as its child. ◀

Procedure 4: ReWire

Input: $\mathcal{T}, \mathcal{N}, v^{\text{new}}$

Output: none

```
1  foreach v' ∈ N do
2      if R2(f'_sup, f^new_sup) then
3          φ^new ← SwingTrajectoryEngine(f^new_swg, f'_sup);
4          if R3(φ^new, (f'_swg, f'_sup)) and c(v^new) + l(v^new, v') < c(v') then
5              UpdateVertex(T, v', (f^new_sup, f'_sup));
6              UpdateEdge(T, v^new, v', φ^new);
7              V_child ← ChildVertexes(T, v');
8              foreach v'' ∈ V_child do
9                  φ' ← SwingTrajectoryEngine(f'_swg, f''_sup);
10                 if φ' ≠ ∅ then
11                     UpdateEdge(T, v', v'', φ');
12                 else
13                     RemoveSubtree(T, v'');
14                 end
15             end
16         end
17     end
18 end
19 return;
```

When the assigned time budget ΔT runs out, tree expansion is stopped. The set $\mathcal{V}_{\text{goal}}$ of vertexes v such that $\boldsymbol{p}_{f,\text{sup}} \in \mathcal{G}$ is retrieved. The vertex v^* with minimum cost is selected as

$$v^* = \underset{v \in \mathcal{V}_{\text{goal}}}{\text{argmin}}\ c(v) \tag{5.8}$$

and the corresponding footstep plan $\mathcal{P}^*$ is retrieved from the branch of $\mathcal{T}$ joining the root to v^*.

[8] In case the engine fails to find such trajectory, the subtree rooted at vertex v'' (including v'' itself) is simply removed from $\mathcal{T}$.

Figure 5.7. The considered scenarios, from left to right: *Rod*, *Ditch*, *Corridor*, *Maze*, *Spacious*.

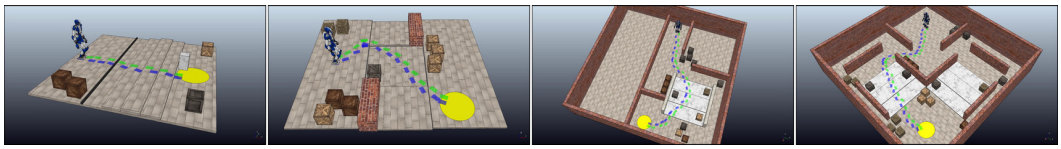

Figure 5.8. Examples of footstep plans found in the scenarios *Rod*, *Ditch*, *Corridor* and *Maze*, respectively, when minimizing the number of steps.

Clearly, the larger the time budget, the better the quality of the obtained footstep plan. We conjecture that our planner inherits the asymptotic optimality property of the general RRT* algorithm [61], although we do not have a formal proof yet.

Planning results

To assess the performance of the proposed footstep planner, we performed a campaign of planning experiments through our C++ implementation on an Intel Core i7-8700K CPU running at 3.7 GHz. The tree constructed by the planner is stored in a k-d tree structure [62], which allows to efficiently perform search and insertion operations. The used robot is HRP-4, a 1.5 m tall humanoid with 34 degrees of freedom by Kawada Robotics.

We considered the five different scenarios (see Fig. 5.7) of different complexity described in the following.

- *Rod.* A thin straight obstacle, which does not provide a large enough surface to step on, must be overcome before ascending and descending a staircase.
- *Ditch.* A ditch can only be entered from the left and exited from the right, because the platform in the middle of it is too low to be accessed directly.
- *Corridor.* A corridor must be exited before ascending and descending a staircase.
- *Maze.* A maze must be navigated, including ascending and descending a staircase, to reach the goal region.
- *Spacious.* The goal region can be reached either by traversing a flat ground or ascending and descending a staircase.

The height of each step is 8 cm for all scenarios except *Ditch* where the height is 10 cm.

In all scenarios, the robot has to reach a circular goal region of radius 0.5 m. The catalogue of primitives U is generated by listing all possible combinations of the following parameters: longitudinal displacement $\{-0.08, 0.00, 0.08, 0.16, 0.2\}$ [m],

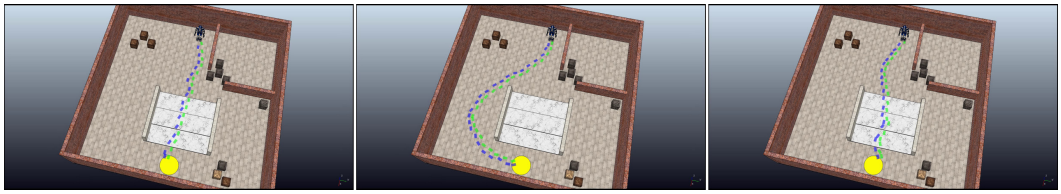

Figure 5.9. Examples of footstep plans found in the scenario *Spacious* when minimizing the number of steps, minimizing the height variation and maximizing the clearance, respectively.

lateral displacement $\{0.20, 0.30\}$ [m] for right support and $\{-0.20, -0.30\}$ [m] for left support, angular displacement $\{0.00, 0.40\}$ [rad] for right support and $\{0.00, -0.40\}$ [rad] for left support (see Fig. 5.3). In the off-line footstep planner we have set $k_\mu = 1$, $k_\gamma = 0$, $h_{\min} = 0.02$ m, $h_{\max} = 0.24$ m, $\Delta h = 0.02$ m, $\Delta_x^- = 0.08$ m, $\Delta_x^+ = 0.24$ m, $\Delta_y^- = 0.07$ m, $\Delta_y^+ = 0.07$ m, $\Delta_z^- = 0.16$ m, $\Delta_z^+ = 0.16$ m, $\Delta_\theta^- = -0.3$ rad, $\Delta_\theta^+ = 0.3$ rad, $\ell = 0.25$ m, $z_b = 0.3$ m, $h_b = 1.2$ m, and $r_b = 0.25$ m. The elevation map $\mathcal{M}_z$ has a resolution of 0.02 m. The three quality criteria described in Sect. 5.2.2 are considered in each scenario.

Tables 5.1–5.3 show the performance of the planner in each scenario, for different values of the time budget, when choosing the three optimality criteria described in Sect. 5.2.2, respectively. In each table, each row reports the results obtained over 100 runs on a combination of scenario and time budget. A total of six performance indexes are tracked and averaged over the total number of successful runs. In particular, a run is considered unsuccessful if the planner terminates without placing any footstep in the goal region. Note that all unsuccessful cases are due to inappropriate time budget. Examination of the table confirms that increasing the time budget both solves this problem by ensuring a high success rate, and improves the quality of the plan in terms of the average cost (Avg Cost). This result supports our conjecture about the asymptotic optimality of the proposed footstep planner.

Figure 5.8 shows the plans generated by minimizing the number of steps in the scenarios *Rod*, *Ditch*, *Corridor* and *Maze*. In particular, in *Rod* the plan allows to correctly pass over the thin obstacle and walk the stairway, eventually reaching the goal region; in *Ditch* the plan reaches the left patch before traversing the low central patches; in *Corridor* the plan manages to exit the first room, reaching the stairway and avoiding the obstacles; in *Maze* the plan takes the left path among the two available, which is the optimal one. Figure 5.9 compares the plans generated in the scenario *Spacious* for each considered cost function. In particular, when minimizing the number of steps the plan goes straight towards the goal region; when minimizing the height variation, the plan avoids the stairway; when maximizing the clearance the plan first moves away from the wall placed on the left flank of the robot at its starting configuration, and then moves towards the goal region while keeping the other obstacles at a safe distance.

5.2.3 Localization

The localization module is continuously fed with the RGB-D images gathered by the head-mounted camera. Based on such information, it is in charge of updating in real

Scenario	Time Budget [s]	Avg Cost	Min Cost	Max Cost	Iters	Tree Size	Successes
Rod	1	22.938	15.000	35.000	6393.8	2948.7	96/100
	5	19.810	15.000	28.000	21537.8	9863.0	100/100
	10	18.050	15.000	25.000	34758.2	15705.5	100/100
	25	16.600	15.000	24.000	62862.0	27944.5	100/100
Ditch	1	40.364	30.000	51.000	5966.7	2119.5	33/100
	5	36.450	27.000	47.000	18632.4	7100.4	100/100
	10	33.420	25.000	42.000	29195.2	11503.3	100/100
	25	30.940	25.000	38.000	52090.5	20755.6	100/100
Corridor	1	57.823	51.000	68.000	6131.4	1880.7	17/100
	5	60.213	46.000	86.000	21589.9	5068.1	89/100
	10	55.687	42.000	81.000	36426.8	8068.1	99/100
	25	49.700	42.000	60.000	70242.7	14415.3	100/100
Maze	1	74.773	62.000	89.000	5813.2	2264.9	22/100
	5	70.949	54.000	94.000	21482.0	8695.5	99/100
	10	65.520	53.000	80.000	35986.2	15327.2	100/100
	25	58.240	50.000	76.000	67507.5	28891.7	100/100
Spacious	1	47.156	37.000	68.000	5749.5	2971.2	96/100
	5	41.700	35.000	55.000	20899.7	10630.8	100/100
	10	39.290	33.000	55.000	34665.6	17412.8	100/100
	25	36.570	31.000	46.000	65308.0	31889.3	100/100

Table 5.1. Performance of the off-line footstep planner when minimizing the number of steps.

Scenario	Time Budget [s]	Avg Cost	Min Cost	Max Cost	Iters	Tree Size	Successes
Rod	1	0.450	0.420	0.480	6634.0	3186.5	96/100
	5	0.431	0.420	0.480	20559.8	9883.2	100/100
	10	0.425	0.420	0.480	31652.4	15140.0	100/100
	25	0.423	0.420	0.480	53598.0	25297.7	100/100
Ditch	1	0.640	0.640	0.640	6218.1	2294.4	34/100
	5	0.640	0.640	0.640	17250.0	6750.3	100/100
	10	0.640	0.640	0.640	25333.2	10171.4	100/100
	25	0.640	0.640	0.640	42419.5	17402.7	100/100
Corridor	1	0.400	0.400	0.400	6437.1	2035.2	26/100
	5	0.400	0.400	0.400	20834.9	5288.8	92/100
	10	0.400	0.400	0.400	33405.4	8035.9	99/100
	25	0.400	0.400	0.400	61279.5	13747.2	100/100
Maze	1	0.480	0.480	0.480	6170.1	2455.4	24/100
	5	0.480	0.480	0.480	20751.2	8991.1	99/100
	10	0.480	0.480	0.480	33049.3	14808.7	100/100
	25	0.480	0.480	0.480	57441.5	26592.6	100/100
Spacious	1	0.346	0.000	0.400	5961.4	3254.5	97/100
	5	0.248	0.000	0.400	20470.8	11047.0	100/100
	10	0.212	0.000	0.400	32485.0	17641.2	100/100
	25	0.168	0.000	0.400	57866.5	30630.2	100/100

Table 5.2. Performance of the off-line footstep planner when minimizing the height variation.

Scenario	Time Budget [s]	Avg Cost	Min Cost	Max Cost	Iters	Tree Size	Successes
Rod	1	30.060	20.556	58.287	4925.5	2143.6	92/100
	5	25.480	18.266	42.102	16219.7	6785.2	100/100
	10	23.142	17.586	36.173	26342.3	10602.8	100/100
	25	20.331	17.627	27.902	48444.6	18275.1	100/100
Ditch	1	78.095	58.308	98.540	4200.6	1446.2	10/100
	5	72.377	54.725	99.004	13681.3	4456.7	96/100
	10	64.840	49.282	83.708	21817.8	7302.6	100/100
	25	55.466	44.438	71.903	39632.2	13139.3	100/100
Corridor	1	94.234	72.764	112.454	4538.3	1376.9	12/100
	5	97.323	73.304	139.995	15477.9	3418.1	82/100
	10	88.866	66.266	139.700	26354.5	5314.1	98/100
	25	78.178	64.615	102.587	52076.3	9236.0	100/100
Maze	1	107.971	95.340	127.139	4374.8	1744.1	8/100
	5	106.620	81.630	149.408	16037.0	5919.7	88/100
	10	99.264	73.844	133.529	27154.9	10394.0	100/100
	25	84.433	65.234	119.569	52035.2	19752.6	100/100
Spacious	1	52.476	32.652	85.874	4467.9	2286.9	97/100
	5	47.138	30.160	70.100	15879.2	7689.2	100/100
	10	43.535	28.001	66.777	26325.6	12425.1	100/100
	25	38.048	27.357	57.544	49981.9	22038.9	100/100

Table 5.3. Performance of the off-line footstep planner when maximizing the minimum clearance.

time the estimate $\hat{\boldsymbol{s}}$ of the pose of the camera frame. To this end, it uses RTAB-Map [63], an open source visual SLAM library. In particular, the visual odometry and graph optimization tool are employed. The first tracks the features automatically extracted from the RGB-D images, while the second minimizes the odometry error through a graph-SLAM algorithm and a loop closure detector. It is worth mentioning that our architecture is independent from the specific implementation of the localization module, hence any off-the-shelf visual SLAM method can be employed in place of RTAB-Map (see assumption A3 in Sect. 5.1).

Given the pose $\hat{\boldsymbol{s}}$ of the camera frame estimated through visual SLAM and the measured joint positions, the direct kinematics module produces the estimates $\hat{\boldsymbol{p}}_{\mathrm{c}}$ and $\hat{\boldsymbol{\varphi}}$ of, respectively, the CoM position and swing foot pose. These are then provided to the gait generation and kinematic control module to achieve closed-loop control.

5.2.4 Simulations

We performed simulations on the HRP-4 humanoid robot in the CoppeliaSim environment. We tested our off-line framework in multiple environments (Fig. 5.7). For the gait generation module we have set $\eta = 3.6\ \mathrm{s}^{-1}$, the single support duration $T_{ss} = 0.6$ s, the double support duration $T_{ds} = 0.4$ s, the size of the moving box $\tilde{d}_x^z = \tilde{d}_y^z = d_z^z = 0.05$ m, $\beta = 1000$, $C = 100$ and $\delta = 0.01$ s. To solve the QP problems we used `hpipm`, which requires less than 1 ms to solve each QP and is thus able to run in real-time with an ample margin.

Figure 5.10 shows the robot traversing the scenario *Ditch*. The robot starts by moving to its left (first snapshot), approaching the accessible patch (second snapshot). It then accesses the platform in the middle correctly avoiding the obstacle (third and fourth snapshots), eventually reaching the goal region by climbing the final two patches (fifth and sixth snapshots). Figure 5.11 shows the robot moving

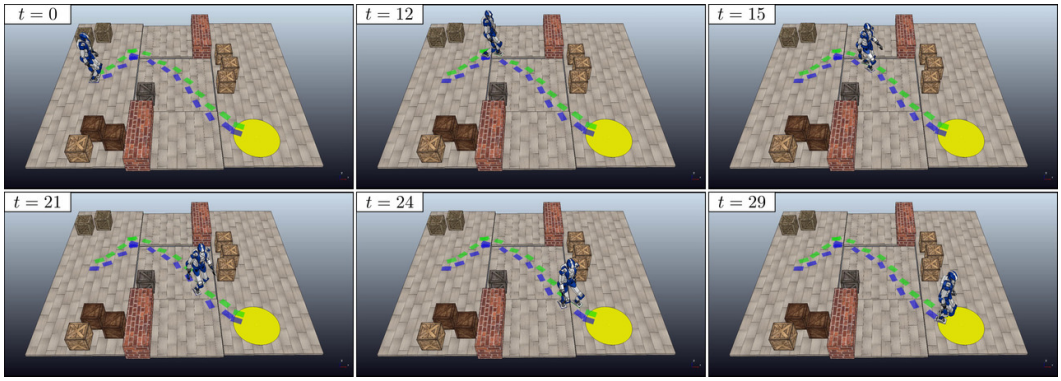

Figure 5.10. The robot reaches the goal going through the ditch, which can only be accessed from the left and exited from the right.

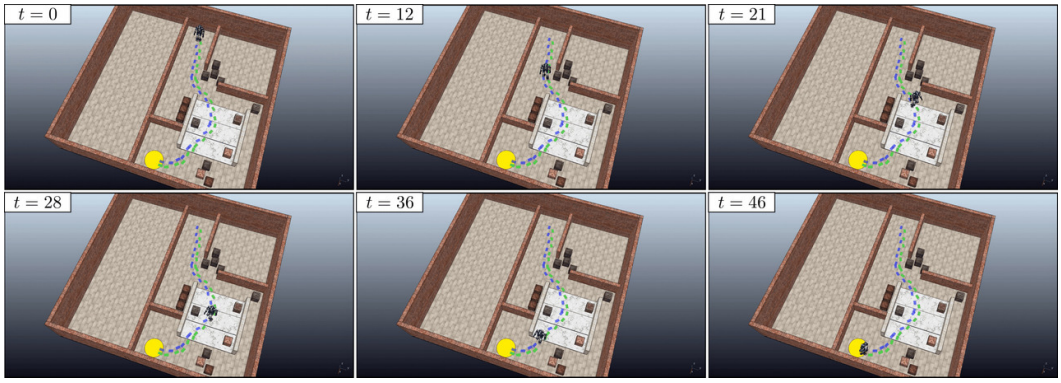

Figure 5.11. The robot reaches the goal avoiding the corridor, climbing and descending the staircase while avoiding the obstacles.

inside the scenario *Corridor*. The robot first exits the room in which it starts (first and second snapshots), approaching the stairway (third snapshot). Then, it goes up and down the staircases avoiding the obstacles (fourth and fifth snapshots), finally reaching the goal region (sixth snapshot).

We invite the reader to watch the video, available at `https://youtu.be/BF43qUcx4gY`, which includes clips related to the above simulations as well as additional cases.

5.3 The on-line case

We now extend the proposed method to the on-line case. This section starts with a description of the general architecture which, compared to that proposed for the off-line case, includes two additional modules, i.e., the mapping and visual task generation module, and employs a sensor-based version of the footstep planner, which will now work on-line; all the other modules, in particular gait generation, remain instead identical. Then, we describe the mentioned components and present some simulation results.

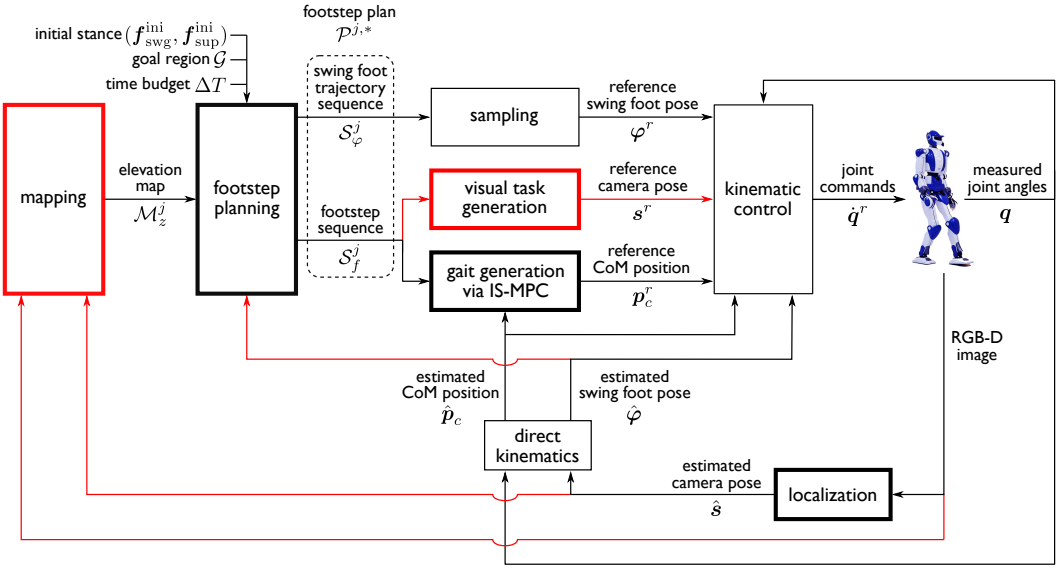

Figure 5.12. Block scheme of the on-line case. The red blocks and arrows highlight the additional modules and signals compared to the off-line case.

5.3.1 General architecture

The proposed architecture for the on-line case is given in Fig. 5.12, where the additional modules and feedback signals are shown in red.

At the beginning, the map $\mathcal{M}_z$ is initialized combining some limited exogenous knowledge about the starting location of the robot and information available by the head-mounted camera at its initial pose. Such initial map $\mathcal{M}^0_z$, together with the initial humanoid stance $(\boldsymbol{f}^{\text{ini}}_{\text{swg}}, \boldsymbol{f}^{\text{ini}}_{\text{sup}})$, the goal region $\mathcal{G}$ and a preassigned time budget ΔT, is provided to the footstep planner to find a first (possibly partial) footstep plan $\mathcal{P}^{1,*} = \{\mathcal{S}^1_f, \mathcal{S}^1_\varphi\}$.

After this initial off-line phase, all the modules run in parallel, generating the humanoid motions in a sensor-based, closed-loop fashion. The mapping module incrementally builds the elevation map $\mathcal{M}_z$ using the RGB-D images acquired by the humanoid while walking and the estimate $\hat{\boldsymbol{s}}$ of the camera pose produced by the localization module. To account for changes in $\mathcal{M}_z$ and take advantage of newly acquired information, the footstep plan is on-line updated and/or extended by repeatedly invoking the footstep planner at every step of the humanoid, with the ultimate objective of reaching $\mathcal{G}$.

More precisely, consider the generic timestamp t^j_s, i.e., the beginning of the j-th step. Let $(\hat{\boldsymbol{f}}^j_{\text{swg}}, \hat{\boldsymbol{f}}^j_{\text{sup}})$ be the current stance, with $\hat{\boldsymbol{f}}^j_{\text{swg}}$ and $\hat{\boldsymbol{f}}^j_{\text{sup}}$ the estimates of the swing and support foot poses at t^j_s, and $\mathcal{P}^{j,*} = \{\mathcal{S}^j_f, \mathcal{S}^j_\varphi\}$ be the current footstep plan – computed during the previous $((j-1)$-th) step – where the sequences of footstep placements and associated swing trajectories are defined as

$$\mathcal{S}^j_f = \{\boldsymbol{f}^{j|j}, \ldots, \boldsymbol{f}^{j+n|j}\},$$
$$\mathcal{S}^j_\varphi = \{\boldsymbol{\varphi}^{j|j}, \ldots, \boldsymbol{\varphi}^{j+n-2|j}\}$$

with their generic elements $\boldsymbol{f}^{j+i|j}$ and $\boldsymbol{\varphi}^{j+i|j}$ denoting, respectively, the $(j+i)$-

th footstep and trajectory produced by the j-th planner invocation, $\boldsymbol{f}^{j|j} \approx \hat{\boldsymbol{f}}^{j}_{\text{swg}}$, $\boldsymbol{f}^{j+1|j} \approx \hat{\boldsymbol{f}}^{j}_{\text{sup}}$ and the last footstep $\boldsymbol{f}^{j+n|j}$ henceforth referred to as *subgoal*. Also, let $(\boldsymbol{f}^{j+1}_{\text{swg}}, \boldsymbol{f}^{j+1}_{\text{sup}})$ be the stance that the humanoid is supposed to achieve at $t^{j+1}_s = t^j_s + T^j_s$, with $\boldsymbol{f}^{j+1}_{\text{swg}} = \hat{\boldsymbol{f}}^{j}_{\text{sup}}$ and $\boldsymbol{f}^{j+1}_{\text{sup}} = \boldsymbol{f}^{j+2|j}$, after performing the swing trajectory $\boldsymbol{\varphi}^{j|j}$ having duration T^j_s. Then, during the time interval $[t^j_s, t^{j+1}_s)$, motion execution and footstep planning take place simultaneously as follows.

- At any time instant $t \in [t^j_s, t^{j+1}_s)$, the current reference position $\boldsymbol{p}^r_{\text{c}}$ of the CoM is produced by the gait generator, based on the sequence $\mathcal{S}^j_f$, similarly to the off-line case; the current reference pose $\boldsymbol{\varphi}^r$ of the swing foot is obtained by sampling the trajectory $\boldsymbol{\varphi}^{j|j}$; moreover, the visual task generator produces the reference pose $\boldsymbol{s}^r$ of the camera frame, given its current estimate $\hat{\boldsymbol{s}}$ and the sequence $\mathcal{S}^j_f$, that allows to direct the gaze towards the subgoal extracted from $\mathcal{S}^j_f$, and then to enlarge $\mathcal{M}_z$ in the area of the current destination. References $\boldsymbol{p}^r_{\text{c}}$, $\boldsymbol{\varphi}^r$, $\boldsymbol{s}^r$, together with their estimates $\hat{\boldsymbol{p}}_{\text{c}}$, $\hat{\boldsymbol{\varphi}}$, $\hat{\boldsymbol{s}}$, are passed to the kinematic controller to compute the joint commands $\dot{\boldsymbol{q}}^r$ for the robot.

- At t^j_s, the footstep planner is invoked providing in input the stance $(\boldsymbol{f}^{j+1}_{\text{swg}}, \boldsymbol{f}^{j+1}_{\text{sup}})$, the goal region $\mathcal{G}$, a time budget equal to T^j_s, and the elevation map $\mathcal{M}^j_z$ currently available by the mapping module. At t^{j+1}_s, the planner returns a new footstep plan $\mathcal{P}^{j+1,*} = \{\mathcal{S}^{j+1}_f, \mathcal{S}^{j+1}_\varphi\}$, where the sequences $\mathcal{S}^{j+1}_f$ and $\mathcal{S}^{j+1}_\varphi$ are defined similarly to $\mathcal{S}^j_f$ and $\mathcal{S}^j_\varphi$, $\boldsymbol{f}^{j+1|j+1} = \boldsymbol{f}^{j+1}_{\text{swg}}$ and $\boldsymbol{f}^{j+2|j+1} = \boldsymbol{f}^{j+1}_{\text{sup}}$. The first element $\boldsymbol{\varphi}^{j+1|j+1}$ of $\mathcal{S}^{j+1}_\varphi$ will define the next ($(j+1)$-th) step of the humanoid.

Note that, while the footstep planner will make use of a fixed map $\mathcal{M}^j_z$ during the time interval $[t^j_s, t^{j+1}_s)$, the map $\mathcal{M}_z$ will continuously be updated by the mapping module during the same time interval, which will generally provide a different map $\mathcal{M}^{j+1}_z$ for the next invocation of the planner.

Clearly, in the on-line case, only the quality of the partial footstep plans can be accounted for, ultimately leading to an overall plan that is globally suboptimal.

5.3.2 Mapping

At the generic time instant, the mapping module receives in input the last RGB-D image acquired by the head-mounted camera and the current estimate $\hat{\boldsymbol{s}}$ of the camera pose produced by the localization module. It is responsible for integrating such newly acquired information into the elevation map $\mathcal{M}_z$.

First, the depth data extracted from the RGB-D image are used to construct a point cloud. Then, the latter is given in input, together with the estimate $\hat{\boldsymbol{s}}$ and a sensor noise model, to Elevation Mapping [64], an open source framework designed for rough terrain mapping; this accordingly updates a local (limited around the robot) representation of the environment in the form of a 2.5D grid map (see Assumption A1). Finally, such local map is integrated into $\mathcal{M}_z$ in order to maintain a global representation of the explored area of the environment.

The mapping module, at the time t^j_s of the generic j-th invocation of the footstep planner, provides it with a copy $\mathcal{M}^j_z$ of the available map $\mathcal{M}_z$. Meanwhile, during

the planner operation, the map $\mathcal{M}_z$ is continuously updated through the process described above.

5.3.3 Sensor-based footstep planning

This module consists in a sensor-based version of the footstep planner proposed in Sect. 5.2.2 which works using the knowledge about the environment incrementally acquired by the robot during motion. We now describe the footstep planning algorithm for the on-line case (Algorithm 2).

Algorithm 2: SensorBasedFootstepPlanner

Input: $(\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1}), \mathcal{G}, \Delta T^j, \mathcal{M}_z^j$

Output: $\mathcal{P}^{j+1,*}$

$(\mathcal{T}^{j+1}, v^{\text{root}}) \leftarrow \text{InitializeTree}(\mathcal{T}^j, (\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1}))$;
$\mathcal{V}_{\text{child}} \leftarrow \text{ChildVertexes}(\mathcal{T}^{j+1}, v^{\text{root}})$;
foreach $v' \in \mathcal{V}_{\text{child}}$ **do**
  $\text{UpdateTree}(\mathcal{T}^{j+1}, v', \mathcal{M}_z^j)$;
end
$\text{ExpandTree}(\mathcal{T}^{j+1}, \Delta T^j - \Delta T_{\text{e}}, \mathcal{M}_z^j)$;
$\mathcal{P}^{j+1,*} \leftarrow \text{RetrieveBestPlan}(\mathcal{T}^{j+1}, \mathcal{G})$;
return $\mathcal{P}^{j+1,*}$;

The input data for the j-th invocation of the footstep planner are the next robot stance $(\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1})$, the goal region $\mathcal{G}$, the time budget ΔT^j and the elevation map $\mathcal{M}_z^j$. Given an optimality criterion, the footstep planner returns the best footstep plan $\mathcal{P}^{j+1,*}$, found within ΔT^j, either leading to $\mathcal{G}$ or terminating in proximity of the frontier of $\mathcal{M}_z^j$. The latter case is typical whenever $\mathcal{G}$ is not included in $\mathcal{M}_z^j$, e.g., due to occlusions or simply being placed far from the robot.

The planning algorithm builds a tree $\mathcal{T}^{j+1}$ reusing portions of the tree $\mathcal{T}^j$ built up to the previous invocation. In this tree, vertexes and edges are defined as described in Sect. 5.2.2, with the only difference that a vertex $v = (\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}})$ can contain a support footstep $\boldsymbol{f}_{\text{sup}}$ whose z-coordinate is unspecified, indicating that $\mathcal{M}_z^j$ does not provide enough information (in a sense formally defined in the following) about the ground under the foot at $\boldsymbol{f}_{\text{sup}}$. Vertexes with this characteristic represent stances located on the frontier of $\mathcal{M}_z^j$ and thus indicate possible direction for further exploration of the environment. The generic invocation consists of: initializing, updating and expanding the tree. These individual steps are described in the following.

Initializing: The vertex $v^{\text{root}} = (\boldsymbol{f}_{\text{swg}}^{\text{root}}, \boldsymbol{f}_{\text{sup}}^{\text{root}})$ of $\mathcal{T}^j$ that is closest to $(\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1})$ is identified. To this end, we define a stance-to-stance metric as

$$\zeta(v, v') = \gamma(\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}'_{\text{swg}}) + \gamma(\boldsymbol{f}_{\text{sup}}, \boldsymbol{f}'_{\text{sup}}) \tag{5.9}$$

where $\gamma(\cdot)$ is the footstep-to-footstep metric defined in (5.4). The subtree of $\mathcal{T}^j$ rooted at v^{root} is extracted (including v^{root} itself) and represents the initial version of $\mathcal{T}^{j+1}$. To match the stance that the humanoid is supposed to reach at the end of the simultaneously executed step, v^{root} is modified by relocating $\boldsymbol{f}_{\text{swg}}^{\text{root}}$ to $\boldsymbol{f}_{\text{swg}}^{j+1}$ and $\boldsymbol{f}_{\text{sup}}^{\text{root}}$ to $\boldsymbol{f}_{\text{sup}}^{j+1}$. This step corresponds to Procedure 5. ◀

Procedure 5: InitializeTree

Input: $\mathcal{T}^j, (\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1}))$

Output: $\mathcal{T}^{j+1}, v^{\text{root}}$

1 $v^{\text{root}} \leftarrow \text{NearestVertex}(\mathcal{T}^j, (\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1}))$;
2 $\mathcal{T}^{j+1} \leftarrow \text{ExtractSubtree}(\mathcal{T}^j, v^{\text{root}})$;
3 $\text{UpdateVertex}(\mathcal{T}^{j+1}, v^{\text{root}}, (\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1}))$;
4 **return** $(\mathcal{T}^{j+1}, v^{\text{root}})$;

Updating: At this point, requirements R1–R3 are satisfied by construction in $\mathcal{T}^{j+1}$ according to the previous map $\mathcal{M}_z^{j-1}$. Then, R1–R3 must now be checked in $\mathcal{T}^{j+1}$ using the most recent map $\mathcal{M}_z^j$, consequently updating vertexes and edges in order to satisfy them. To this end, we perform a pre-order traversal of $\mathcal{T}^{j+1}$ as described in the following.

When a vertex $v = (\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}})$ is visited, it is modified[9] by relocating its swing footstep to the support footstep $\boldsymbol{f}_{\text{sup}}^{\text{parent}}$ of its parent $v^{\text{parent}} = (\boldsymbol{f}_{\text{swg}}^{\text{parent}}, \boldsymbol{f}_{\text{sup}}^{\text{parent}})$, and setting the z coordinate $z_{f,\text{sup}}$ of $\boldsymbol{f}_{\text{sup}}$ according to $\mathcal{M}_z^j$. In particular, consider the cells of $\mathcal{M}_z^j$ belonging to, or overlapping with, the footprint at $\boldsymbol{f}_{\text{sup}}$; let n_{k} and n_{u} be the number of these cells whose height is known and unknown, respectively. If the rate of cells with known height is larger than a predefined threshold $\bar{n}$, i.e.,

$$\frac{n_{\text{k}}}{n_{\text{k}} + n_{\text{u}}} > \bar{n},$$

$z_{f,\text{sup}}$ is set to the average value of the n_{k} known heights. Otherwise, $z_{f,\text{sup}}$ is left unspecified.

Once v has been updated, requirements R1–R3 are checked similarly to what was done in the off-line case, with the only two differences described in the following.

- If $z_{f,\text{sup}}$ is unspecified, requirements R1–R3 are checked conjecturing that it is equal to the z-component $z_{f,\text{swg}}$ of $\boldsymbol{f}_{\text{swg}}$.
- Requirement R1 is considered satisfied if for each of the n_{k} known heights, the net variation from $z_{f,\text{sup}}$ does not exceed a predefined threshold $\bar{z}$, i.e.,

  $$|z_{\text{k}} - z_{f,\text{sup}}| \leq \bar{z}.$$

  with z_{k} the generic known height among the n_{k} available.

If any requirement among R1–R3 is violated, vertex v is removed from $\mathcal{T}^{j+1}$, along with its descendants. Otherwise, the edge connecting v to v^{parent} is replaced by the trajectory $\boldsymbol{\varphi}^{\text{parent}}$ generated while checking R3; the set $\mathcal{V}_{\text{child}}$ of child vertexes of v is retrieved, and the procedure is recursively invoked on them. To guarantee on-line performance and save time to be used for expanding $\mathcal{T}^{j+1}$, recursion is stopped on vertexes having a maximum depth $\bar{\kappa}$. This step corresponds to Procedure 6. ◀

Expanding: Once $\mathcal{T}^{j+1}$ has been updated, it can be further expanded in the map $\mathcal{M}_z^j$. Let ΔT_{e} be the time elapsed since the beginning of the current

[9] This modification is not made on v^{root} as it corresponds to the stance that the robot must reach at the end of the simultaneously executed step.

Procedure 6: UpdateTree

Input: $\mathcal{T}^{j+1}, v, \mathcal{M}_z^j$
Output: none

$v^{\text{parent}} \leftarrow \text{ParentVertex}(\mathcal{T}^{j+1}, v)$;
$z_{f,\text{sup}} \leftarrow \text{DetermineFootstepHeight}(\boldsymbol{f}_{\text{sup}}, \mathcal{M}_z^j)$;
$\text{UpdateVertex}(\mathcal{T}^{j+1}, v, (\boldsymbol{f}_{\text{sup}}^{\text{parent}}, (x_{f,\text{sup}}, y_{f,\text{sup}}, z_{f,\text{sup}})))$;
if $\text{R1}(\boldsymbol{f}_{\text{sup}})$ **and** $\text{R2}(\boldsymbol{f}_{\text{sup}}, \boldsymbol{f}_{\text{sup}}^{\text{parent}})$ **then**
  $\boldsymbol{\varphi}^{\text{parent}} \leftarrow \text{SwingTrajectoryEngine}(\boldsymbol{f}_{\text{swg}}^{\text{parent}}, \boldsymbol{f}_{\text{sup}})$;
  if $\text{R3}(\boldsymbol{\varphi}^{\text{parent}}, (\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}}))$ **then**
    $\text{UpdateEdge}(\mathcal{T}^{j+1}, v^{\text{parent}}, v, \boldsymbol{\varphi}^{\text{parent}})$;
    if $\text{Depth}(\mathcal{T}^{j+1}, v)) = \bar{\kappa}$ **then**
      return;
    end
    $\mathcal{V}_{\text{child}} \leftarrow \text{ChildVertexes}(\mathcal{T}^{j+1}, v)$;
    foreach $v' \in \mathcal{V}_{\text{child}}$ **do**
      $\text{UpdateTree}(\mathcal{T}^{j+1}, v', \mathcal{M}_z^j)$;
    end
  else
    $\text{RemoveSubtree}(\mathcal{T}^{j+1}, v)$;
  end
else
  $\text{RemoveSubtree}(\mathcal{T}^{j+1}, v)$;
end
return;

invocation of the footstep planner, i.e., the time spent in initializing and updating $\mathcal{T}^{j+1}$. The expansion of $\mathcal{T}^{j+1}$ works iteratively as described in Sect. 5.2.2 using the remaining portion of the time budget $\Delta T^j - \Delta T_{\text{e}}$ and the map $\mathcal{M}_z^j$, with the following modifications.

- The choice of the z-coordinate for a candidate footstep $\boldsymbol{f}_{\text{sup}}^{\text{cand}}$ and the check of requirements R1–R3 are done exactly as when updating a generic vertex.
- A vertex whose support footstep has unspecified z-coordinate cannot be set as parent of another vertex. Then, such vertexes are excluded both when selecting the vertex v^{near} for an expansion attempt and when choosing a parent for a candidate vertex v^{cand}. ◀

Similarly to the off-line case, when the assigned time budget ΔT^j runs out, tree expansion is stopped and the set $\mathcal{V}_{\text{goal}}$ of vertexes v such that $\boldsymbol{p}_{f,\text{sup}} \in \mathcal{G}$ is retrieved. If $\mathcal{V}_{\text{goal}}$ is not empty, the vertex v^* with minimum cost is selected as in (5.8). Otherwise, if $\mathcal{V}_{\text{goal}}$ is empty, the planner retrieves the set $\mathcal{V}_{\text{fron}}$ containing all vertexes of $\mathcal{T}^{j+1}$ having at least one child vertex whose support footstep has unspecified z-coordinate. In practice, vertexes in $\mathcal{V}_{\text{fron}}$ contain stances located in proximity of the frontier of the current map $\mathcal{M}_z^j$. Then, the vertex v^* is selected as

$$v^* = \underset{v \in \mathcal{V}_{\text{fron}}}{\text{argmin}}\ c(v) + g(v) \tag{5.10}$$

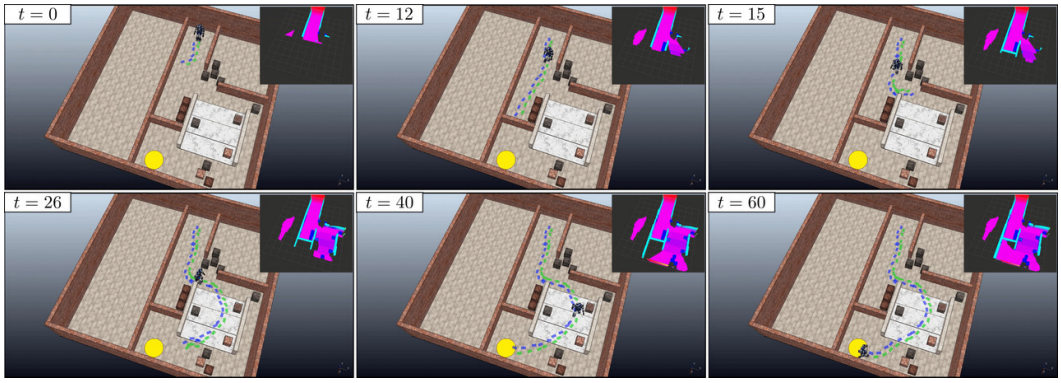

Figure 5.13. The on-line footstep planner in the scenario *Corridor* minimizing the number of steps. Here the planner finds a footstep sequence of 54 steps.

where $g(v)$ represents the *cost-to-go* of vertex v, i.e., a lower-bound on the minimum cost to reach $\mathcal{G}$ from v. A possible choice for $g(v)$ when minimizing the number of steps along the plan will be described in Sect. 5.3.5.

Finally, the footstep plan $\mathcal{P}^{j+1,*}$ is retrieved from the branch of $\mathcal{T}^{j+1}$ joining the root to v^*. Clearly, if $\mathcal{V}_{\text{goal}}$ is not empty, $\mathcal{P}^{j+1,*}$ will be a complete footstep plan leading to $\mathcal{G}$; otherwise, $\mathcal{P}^{j+1,*}$ will be a partial footstep plan leading in the direction of an unknown area of the environment whose exploration is considered useful – according to the adopted cost-to-go – to proceed towards $\mathcal{G}$.

5.3.4 Visual task generation

At any time instant during the execution of the generic j-th step, given the current estimate $\hat{\boldsymbol{s}}$ of the camera pose and the subgoal $\boldsymbol{f}^{j+n|j}$, which is readily extracted from the current sequence $\mathcal{S}_f^j$ of footstep placements, the visual task generator is in charge of producing a suitable reference $\boldsymbol{s}^r$ of the camera pose which aims at directing the gaze towards the current destination of the robot. The rationale beyond this choice is that, since the current footstep plan terminates in an area on the frontier of the map $\mathcal{M}_z$ that is considered promising for goal-oriented exploration of the environment, looking in the direction of $\boldsymbol{f}^{j+n|j}$ allows to enlarge the map in that particular area. In principle, whenever possible, this will privilege further extension of the footstep plan in that promising direction.

To compute $\boldsymbol{s}^r$, one possibility consists in adopting an image-based visual servoing scheme [65]. In particular, one may define a virtual feature in the image plane of the camera at $\hat{\boldsymbol{s}}$ associated to the representative point $\boldsymbol{p}_f^{j+n|j}$ of the subgoal footstep $\boldsymbol{f}^{j+n|j}$. Then, the reference pose $\boldsymbol{s}^r$ of the camera frame can be computed so as to keep such feature at the center of the image plane.

The produced reference pose $\boldsymbol{s}^r$ is passed to the kinematic controller which, in practice, only controls the camera yaw and pitch angles.

5.3.5 Simulations

In this section we present simulations obtained with the discussed architecture for the on-line case. Parameters are set to the same values of Sect. 5.2.4, with the

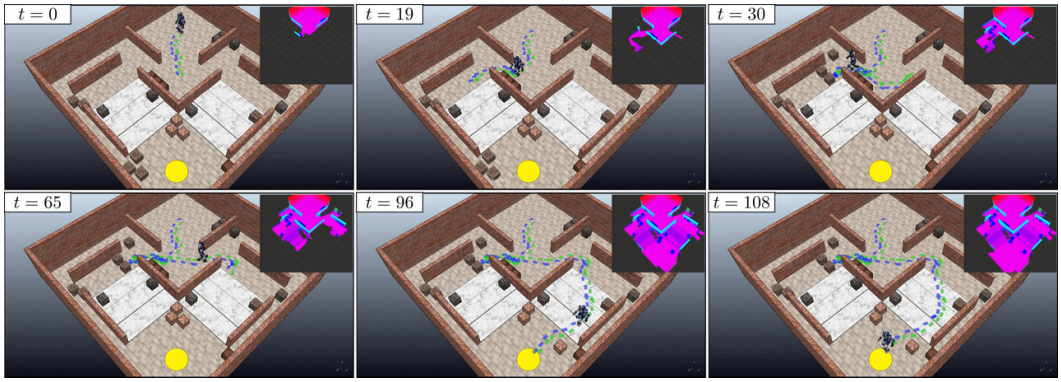

Figure 5.14. The on-line footstep planner in the environment *Maze* minimizing the number of steps. The environment is dynamic, namely the elevation map can be changed by moving obstacles around. Here the planner finds a footstep sequence of 106 steps.

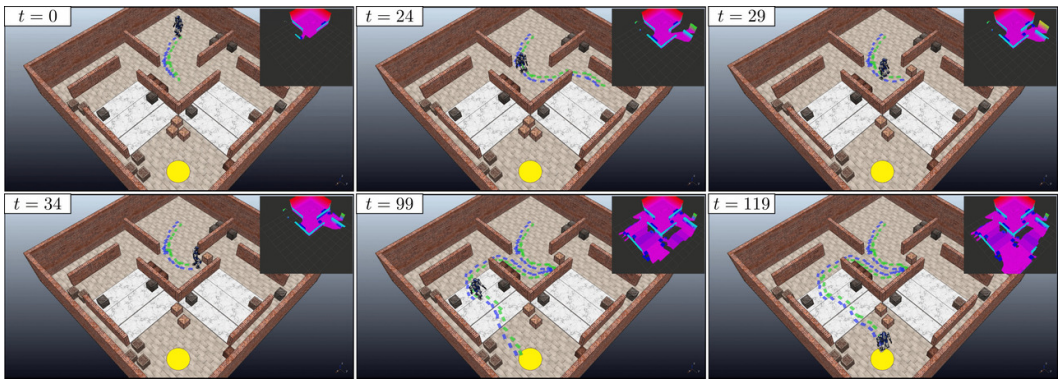

Figure 5.15. The on-line footstep planner in the dynamic environment *Maze* minimizing the number of steps. Here the planner finds a footstep sequence of 103 steps.

only exception of setting $k_\gamma = 1$ in (5.4) when used in (5.9), $\bar{n} = 0.9$, $\bar{z} = 0.02$ and $\bar{\kappa} = 5$. For each j-th invocation of the footstep planner the time budget ΔT^j is set to $T_{ss} + T_{ds}$. In all performed simulations, the quality criteria considered by the footstep planner is the number of steps, while the *cost-to-go* of each vertex v is an underestimation of the number of steps needed to reach the goal region $\mathcal{G}$ from the double support configuration specified by v. This value is computed as the distance between the position of the support foot specified by v and $\mathcal{G}$, divided by the longest step among the catalogue of primitives U.

Figure 5.13 shows the robot walking in the scenario *Corridor* together with the reconstructed elevation map. Here, the planner continuously receives an updated version of the map, which is built while the robot moves. Initially (first snapshot) the robot starts exploring its surrounding environment, moving towards the end of the corridor (second snapshot). As soon as the footstep planner realizes that the room is closed, it replans a sequence of footsteps which brings the robot outside the corridor (third snapshot). The robot keeps exploring the environment, going up and down the stairs and avoiding the obstacles placed along the path (fourth and fifth snapshot). Finally, the robot reaches the desired goal region (sixth snapshot).

Figure 5.14 shows the robot accomplishing the locomotion task in the scenario *Maze*. In this case the scenario was rendered dynamic by manually moving the obstacles at runtime. Here, the planner is facing the additional challenge of operating under continuous changes in the elevation map, which reflects the new locations of the obstacles. The robot starts by moving outside the initial room (first snapshot), choosing the path on its right (second snapshot). The footstep plan is invalidated by placing obstacles in front of the robot, forcing the robot to choose the other direction (third snapshot). The footstep planner correctly drives the robot towards the other area of the maze (fourth snapshot), making it go up and down the staircase (fifth snapshot), avoiding another obstacle which is placed in front of the robot right before reaching the goal region (sixth snapshot).

Figure 5.15 shows a situation, again in a dynamic version of the scenario *Maze*, in which the planner reaches a point in which is not able to find a new subgoal. This occurs when, once the time budget expires, both $\mathcal{V}_{\text{goal}}$ and $\mathcal{V}_{\text{fron}}$ are empty. For example, this may happen when the humanoid must exit a long corridor or when dynamic obstacles invalidate large portion of the created tree. In this specific situation, a simple solution consists in keeping the portion of the current footstep plan that is still valid after the updating step of the planner. If this happens multiple times in a row, the robot reaches the subgoal and stops. At this point, the footstep planner is invoked with an unlimited time budget and terminates as soon as a new subgoal is found. As before, the robot starts by moving outside the initial room, moving towards its left (first and second snapshots). The footstep plan is invalidated by moving an obstacle in front of the robot (third snapshots), which stops its motion upon reaching the current subgoal (fourth snapshot). As soon as the footstep planner finds a new subgoal, the robot starts moving again (fifth snapshot), eventually reaching the desired goal region (sixth snapshot).

Clips of the described simulations are available at the following link: `https://youtu.be/BF43qUcx4gY`.

5.4 Discussion

The proposed approach integrates several components and is designed to work both off-line and on-line. Since, to the best of our knowledge, no existing method can address the same wide range of situations, we focus in the following on the two main components (footstep planning and gait generation) separately.

As a representative of the state of the art in footstep planning, we selected the algorithm in [11], which uses a weighted A^* algorithm to search for optimal footstep sequences on uneven ground[10]. At each iteration, the vertex providing the lowest estimate for the path cost is expanded. This estimate is computed by adding to the cost of the vertex a heuristic cost-to-go, given by the distance to the goal divided by the maximum step length and multiplied by a weight $w \geq 1$, which can be used to increase the bias towards the goal region. The main difference with respect to our approach lies in the expansion mechanism, which is deterministic in [11] and probabilistic in our method.

Both our scheme and the weighted A^* approach use a catalogue of primitives. In order to perform a fair comparison, we use for the weighted A^* approach the same catalogue described in Sect. 19. As for the optimality criterion, we aim to minimize the number of footsteps, corresponding to an edge cost given by (5.5). In order to allow for the possibility that our implementation might not be the most efficient, we assigned to the weighted A^* planner a time budget of 100 s, which is four times the largest budget used when testing our planner.

The results obtained showed that standard A^* search, corresponding to $w = 1$, is unable to find solutions within the allotted time budget in any of the considered scenarios. By increasing w, weighted A^* performs rather well in scenarios where the solution does not involve considerable backtracking (*Rod* and *Spacious*), but fails to find the solution in any other scenario. In particular, Table 5.4 collects the results obtained for $w = 5$. These results should be compared to those in Table 5.1, which show that our approach has a 100% success rate with a fourth of the time budget. We also ran tests with larger weights ($w = 10$, $w = 25$), obtaining results that are essentially identical, with slightly longer paths and no increase in the success rate.

Indeed, the outcome of the above comparison was rather predictable. It is well known that weighted A^* works quite well in environments where the path leading to the goal does not deviate significantly from a straight line. However, as acknowledged by the authors of [11], its performance may degrade severely in the scenarios that require even mild amounts of backtracking.

[10]The algorithm in [11] actually contemplates the possibility of tilted surfaces, but obviously works in a world of stairs as a particular case.

Scenario	Cost	Iters	Tree Size
Rod	22.0	650	4049
Ditch	Fail	11687	21911
Corridor	Fail	12343	22483
Spacious	31.0	34	546
Maze	Fail	9546	22333

Table 5.4. Performance of the off-line weighted A^* footstep planner when minimizing the number of steps ($w = 5.0$).

Chapter 6

Feasibility-aware plan adaptation in humanoid gait generation

In Chapters 4 and 5 we have seen how to design a scheme for humanoid locomotion, which allow the robot to perform walking motions in complex scenarios. One of the characteristics that marks the presented framework is the separation into a footstep planning phase (based on RRT*), and a Model Predictive Control (MPC) gait generation algorithm. Most available schemes used for humanoid walking rely on such separation, lightening the computational load on the control side, and allowing it to run in real-time. Nevertheless, with this kind of design, the planner is unaware of the underlying dynamics, and the controller is unaware of any disturbances acting on the robot.

In this chapter, we present an on-line Feasibility-Aware Plan Adaptation (FAPA) module which can locally adapt footsteps (positions, timings and orientation) in such a way that to guarantee feasibility of the subsequent Intrinsically Stable MPC (IS-MPC) stage. Indeed, because our architecture is based on IS-MPC (Chapter 4), which involves an explicit stability constraint ensuring the boundedness of the CoM trajectory with respect to the ZMP, it is possible to use its feasibility region (i.e., the state space region for which the constrained QP admits a solution) to enhance the capabilities of the scheme itself. Two examples of such strategy rely on adapting the time of the first step [25] and allowing for non-convex regions [66].

With the FAPA module, which depends on the system state and the dynamics of the chosen template dynamic model, we obtain the generality given by nonlinear constraints without sacrificing much performance as the number of variables in FAPA is much lower than that of the variables of the MPC, making it very fast and capable of working in real time. Furthermore, we explore the inclusion of integer variables, further increasing the range of situations that can be covered. Note that while modules for online footstep adaptation using nonlinear optimization have been proposed [67], they do not work in conjunction with MPC. Our approach is not only designed to work along with the MPC module, but it specifically aimed at enhancing its capabilities.

We present two versions of the proposed FAPA scheme: one with a fixed regions assignment for placing the footstep and another one where the regions are selected automatically through mixed-integer programming.

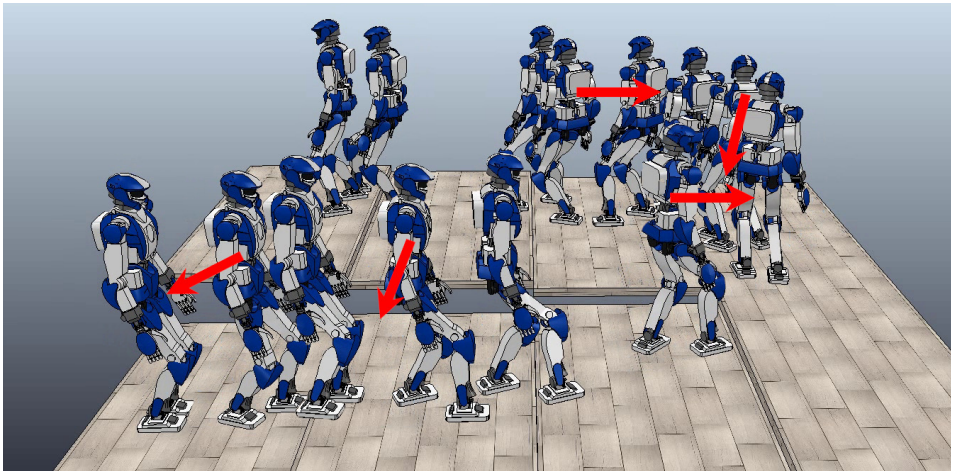

Figure 6.1. An example simulation using the proposed architecture: the robot is walking along a staircase while being subject to multiple pushes. The adaptation module modifies position, orientation and timing of the footsteps real-time to guarantee a successful execution.

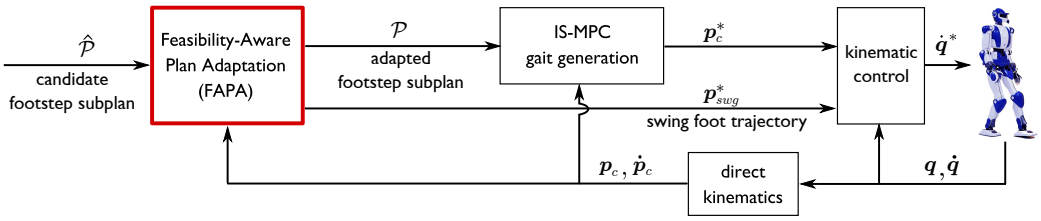

Figure 6.2. A block scheme of the proposed architecture. The candidate footstep subplan $\hat{\mathcal{P}}$ is adapted by the FAPA module, guaranteeing the feasibility of IS-MPC. The IS-MPC module receives the adapted footstep subplan $\mathcal{P}$, and generates a desired trajectory of the CoM $\boldsymbol{p}_c^*$, which is used by the kinematic controller, together with the desired trajectory of the swing foot $\boldsymbol{p}_{\text{swg}}^*$, to generate the desired joint velocities $\dot{\boldsymbol{q}}^*$.

6.1 Problem formulation

The proposed architecture is shown in Fig. 6.2. An external candidate plan is provided, which in this chapter will be either a basic plan to demonstrate simple motions, or a plan generated by randomized exploration (Chapter 5) for more complex environments. A subplan, i.e., a portion of the candidate plan, is given as input to the scheme at each timestep.

The basic components of the considered scheme are:

- a Feasibility-Aware Plan Adaptation (FAPA) block, that can modify locally the high-level footstep plan;
- an IS-MPC gait generation block (Chapter 4) that generates CoM/ZMP trajectories based on the output of FAPA;
- a kinematic controller that realizes at the joint level the generated CoM and swing foot trajectories.

While the high-level footstep plan is designed considering the humanoid's kinematic limitations, it is entirely unaware of its dynamics and is not informed by the robot state since it is fully generated off-line. To make up for this deficiency, the FAPA module performs a local adaptation of the planned footsteps before these enter the IS-MPC stage.

This adaptation is based on a *gait feasibility constraint* that guarantees feasibility of the next IS-MPC stage while trying to match the original plan. It can concurrently change the footstep positions, orientations, as well as step timings.

To formulate this constraint, we leverage the feasibility region of IS-MPC (i.e., the subset of the state space where the problem is feasible at a given time), and define it in an implicit form with the nonlinear dependency on the footstep positions, orientations, and step timings.

The fact that the feasibility of the MPC can be efficiently captured by the expression of this constraint is a crucial aspect of the formulation, because it means that the scheme can harness the power of nonlinear optimization without burdening the MPC itself, which remains linear and can run at a high rate. The nonlinear optimization part is external to the MPC, which allows the number of variables to be kept small and thus to keep the computation time manageable.

We propose two versions of the FAPA module, that differ by the optimization problem required for their implementation. In particular, the first version only uses continuous optimization, while the second one also employs discrete variables and is formulated as a Mixed Integer Nonlinear Program (MINLP). Being the latter very general it can be used to account for more adaptation scenarios, e.g., in which the footsteps can also be moved to different terrain patches than the ones assigned by the high-level planner. As will be discussed extensively in Sect. 6.4, the second version is more demanding in terms of computation time, but we present it as a proof of concept as we strongly believe it can be made to work in real time with proper code optimization.

6.2 Preliminaries

In this section we describe the environment and the structure of the footstep plan used in our scheme.

6.2.1 Environment

The considered environment is a world of stairs, i.e., constituted by flat horizontal regions. The robot is allowed to walk across different regions if these are relatively close in height, and if there is sufficient available surface to step on them, otherwise they will constitute obstacles to be avoided.

The arrangement of these regions is assumed to be known, and it is processed and encoded in the following way:

- regions are reduced in size so that they represent the collision-free area available for the center of the footprint. This is done by performing a Minkowski difference between each flat region and the area swept by a footprint (accounting for all possible footstep orientations);

- after reduction, non-convex regions are subdivided into non-overlapping convex polytopic *patches*.

A patch P is identified by the inequality

$$\boldsymbol{A}(P)\boldsymbol{p} \leq \boldsymbol{b}(P),$$

where $\boldsymbol{A}(P) \in \mathbb{R}^{V(P)\times 2}$ and $\boldsymbol{b}(P) \in \mathbb{R}^{V(P)}$ define a polytope (with $V(P)$ vertices) and $\boldsymbol{p} = (x, y)^\top$ is a generic 2D point. In this way, non-polytopic portions of ground (e.g., round edges) are approximated, but the number of vertices can be arbitrarily large. Since each patch P is flat, its height is denoted simply as $z(P)$.

6.2.2 Footstep plan

The high-level footstep plan is a sequence of *candidate footsteps* $\hat{\boldsymbol{f}}$, each identified by the tuple $\hat{\boldsymbol{f}} = (\hat{x}_f, \hat{y}_f, \hat{z}_f, \hat{\theta}_f, \hat{T}_{\text{ss}}, \hat{T}_{\text{ds}})$. For each planned footstep $\hat{\boldsymbol{f}}$

- $\hat{x}_f$, $\hat{y}_f$ and $\hat{z}_f$ are the coordinates of its center;
- $\hat{\theta}_f$ is its orientation around the z axis;
- $\hat{T}_{\text{ds}}$ and $\hat{T}_{\text{ss}}$ are the durations of its double support and single support phases, respectively;
- we denote by $\Pi(\hat{\boldsymbol{f}})$ the patch that contains the footstep, i.e., the patch P such that[1]
$$(\hat{x}_f, \hat{y}_f)^\top \in P, \quad \hat{z}_f = z(P).$$

The footstep plan $\hat{\mathcal{P}}$ is computed off-line, and at each time t_k a subplan $\hat{\mathcal{P}}^l$ of size $F+1$ is extracted, where l is the index of the first footstep of the current subplan (at t_k), and F a fixed parameter. The subplan contains the next F candidate footsteps:

$$\hat{\mathcal{P}}^l = \left\{\hat{\boldsymbol{f}}^l, \ldots, \hat{\boldsymbol{f}}^{l+F}\right\}.$$

The FAPA block, which performs footsteps adaptation, modifies $\hat{\mathcal{P}}^l$ in the adapted subplan $\mathcal{P}^l$, i.e., in the input of the IS-MPC block

$$\mathcal{P}^l = \left\{\boldsymbol{f}^l, \ldots, \boldsymbol{f}^{l+F}\right\}.$$

After every iteration, if adaptation took place (i.e., $\mathcal{P}^l$ differs from $\hat{\mathcal{P}}^l$), the algorithm performs a *footstep plan override*, i.e., the corresponding portion of the high-level footstep plan is substituted with the adapted subplan $\mathcal{P}^l$. Note that the remaining part of the plan (after the index $l+F$) is unchanged, so if the adaptation makes the robot stray from the initial path it will later try to catch up. This behavior is often acceptable, but might sometimes be undesirable, and can be improved in future versions if we allow the high-level planner to replan on-line (see [48]).

[1] Note that this patch is unique because the environment is subdivided into non-overlapping patches.

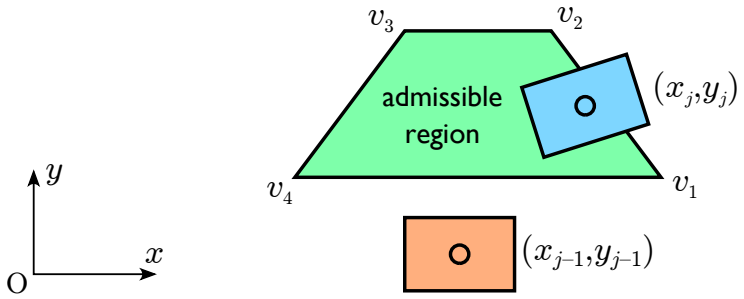

Figure 6.3. Admissible region of the kinematic constraint in the x-y plane.

6.3 Feasibility-Aware Plan Adaptation

In this following section, we describe how to use the feasibility region to formulate a constraint for the FAPA module, and thus ensure that the output of FAPA can be used by IS-MPC to construct a feasible QP. The FAPA module runs in real-time and performs a local adaptation of the subplan $\hat{\mathcal{P}}^l$, including their timing. We now describe the constraints and the optimization problems that define the adaptation procedure.

6.3.1 Kinematic constraint

The j-th footstep $\boldsymbol{f}^j$ is ensured to be kinematically feasible by limiting its displacement with respect to the previous footstep $\boldsymbol{f}^{j-1}$. In practice we constrain the geometric components of $\boldsymbol{f}^j$ to be within the *admissible region*

$$\begin{aligned} \begin{pmatrix} \boldsymbol{n}_1^\top \left(\boldsymbol{p}_{xy}^{l+j} - \boldsymbol{p}_{xy}^{l+j-1} - \boldsymbol{R}(\theta_f^{l+j-1}) \boldsymbol{v}_1 \right) \\ \vdots \\ \boldsymbol{n}_V^\top \left(\boldsymbol{p}_{xy}^{l+j} - \boldsymbol{p}_{xy}^{l+j-1} - \boldsymbol{R}(\theta_f^{l+j-1}) \boldsymbol{v}_V \right) \end{pmatrix} &\geq \boldsymbol{0}, \\ \Delta z^{\mathrm{m}} \leq z_{l+j} - z_{l+j-1} &\leq \Delta z^{\mathrm{M}}, \\ \Delta \theta^{\mathrm{m}} \leq \theta_{l+j} - \theta_{l+j-1} &\leq \Delta \theta^{\mathrm{M}}, \end{aligned} \tag{6.1}$$

with $\boldsymbol{R}(\theta_f^{l+j-1})$ a 2D rotation matrix, $\boldsymbol{n}_i$ the vector normal to the i-th segment of the convex region computed as

$$\boldsymbol{n}_i = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \boldsymbol{R}(\theta_f^{l+j-1})(\boldsymbol{v}_{i+1} - \boldsymbol{v}_i),$$

and $\boldsymbol{v}_i$ being the vertices defining the convex polygon (different depending whether the support foot is left or right), shown in Fig. 6.3. Furthermore, Δ_z^m, Δz^{M}, $\Delta \theta^{\mathrm{m}}$, and $\Delta \theta^{\mathrm{M}}$ define limits for the foot reachability over vertical displacement and relative orientation.

6.3.2 Timing constraint

Single and double support duration are subject to minimum and maximum duration constraints

$$T_{\mathrm{ds}}^{\min} \leq T_{\mathrm{ss}}^{l+j} \leq T_{\mathrm{ds}}^{\max}, \quad T_{\mathrm{ss}}^{\min} \leq T_{\mathrm{ds}}^{l+j} \leq T_{\mathrm{ss}}^{\max}, \tag{6.2}$$

where the bounds $T_{\text{ds}}^{\min}$, $T_{\text{ds}}^{\max}$, $T_{\text{ss}}^{\min}$, $T_{\text{ss}}^{\max}$ are chosen in such a way to avoid excessively fast trajectories that might be difficult to track, as well as very slow steps that could result in quasi-static motion.

6.3.3 Patch constraints

We describe alternative versions of this constraint, as we will later compare the module using either of them, both in terms of the quality of the resulting plan and of the computational load. The first version of the constraint simply restricts the $(l+j)$-th footstep to lie within its associated patch $\Pi(\boldsymbol{f}^{l+j})$, which is the one originally chosen by the high-level planner. This constraint can be written as

$$\begin{cases} \boldsymbol{A}(\Pi(\boldsymbol{f}^{l+j}))\begin{pmatrix} x_f^{l+j} & y_f^{l+j} \end{pmatrix}^\top \leq \boldsymbol{b}(\Pi(\boldsymbol{f}^{l+j})), \\ z_f^{l+j} = z(\Pi(\boldsymbol{f}^{l+j})). \end{cases} \tag{6.3}$$

The second version of the patch constraint allows the footstep to be moved to a different patch. To entertain this possibility, we introduce binary variables in order to formulate a mixed-integer constraint. This constraint defines a logical implication in which, if a certain binary variable $b_{l+j,\kappa}$ is *true*, then a linear constraint must be verified:

$$b_{l+j,\kappa} = 1 \Rightarrow \begin{cases} \boldsymbol{A}(P^\kappa)\begin{pmatrix} x_f^{l+j} & y_f^{l+j} \end{pmatrix}^\top \leq \boldsymbol{b}(P^\kappa), \\ z_f^{l+j} = z(P^\kappa). \end{cases} \tag{6.4}$$

This forces the $(l+j)$-th footstep to lie within the κ-th patch. Since each footstep can only be inside a single patch, we also impose

$$\sum_{\kappa=1}^{R} b_{l+j,\kappa} = 1. \tag{6.5}$$

In MIP, logical implications can be implemented using binary variables through the so-called *big-M* technique [68]. In this case, we rewrite (6.4) as

$$\begin{cases} \boldsymbol{A}(P^\kappa)\begin{pmatrix} x_f^{l+j} & y_f^{l+j} \end{pmatrix}^\top \leq \boldsymbol{b}(P^\kappa) + (1 - b_{l+j,\kappa})M\mathbf{1}_{V(P^\kappa)}, \\ z_{l+j} \leq z(P^\kappa) + (1 - b_{l+j,\kappa})M, \\ -z_{l+j} \leq -z(P^\kappa) + (1 - b_{l+j,\kappa})M, \end{cases} \tag{6.6}$$

where M is a constant large enough to relax the constraints if $b_{l+j,\kappa} = 0$ and $\mathbf{1}_{V(P^\kappa)}$ is a row vector with $V(P^\kappa)$ ones. We define $\hat{\kappa}_{l+j}$ as the index of $\Pi(\hat{\boldsymbol{f}}^{l+j})$. Note that this requires turning the equality constraint into two inequality constraints. Based on the patches of the candidate footsteps in $\hat{\mathcal{P}}^l$, we also define candidate binary variables as

$$\hat{b}_{l+j,\kappa} = \begin{cases} 1, & \text{if } \kappa = \hat{\kappa}_{l+j}, \\ 0, & \text{if } \kappa \neq \hat{\kappa}_{l+j}. \end{cases}$$

Finally, (6.5) and (6.6) assume that every footstep may be mapped to every patch, which requires $F \times R$ binary variables. However, since the computational load of a MIP is largely related to the number of binary variables, we employ a heuristic that allows a footstep $\boldsymbol{f}^j$ to be assigned only to the patches adjacent to $\Pi(\hat{\boldsymbol{f}}^j)$.

6.3.4 Current footstep constraints

The first footstep in the subplan $\boldsymbol{f}^l$ corresponds to the footstep currently in contact with the ground, which means that some of its components cannot be changed. In particular, its geometric components should be constrained to be equal to the corresponding components of $\hat{\boldsymbol{f}}^l$, i.e.,

$$x_f^l = \hat{x}_f^l, \quad y_f^l = \hat{y}_f^l, \quad z_f^l = \hat{z}_f^l, \quad \theta_f^l = \hat{\theta}_f^l. \tag{6.7}$$

Note that, because of the footstep plan override, the components of $\hat{\boldsymbol{f}}^l$ are not the same as in the original plan, but rather those adapted at the previous iteration.

If t_k belongs to a single support phase, the double support of the current step cannot be changed anymore because it is already passed. This is expressed by the constraint

$$t_k - t_s^l > T_{\text{ds}}^l \quad \Rightarrow \quad T_{\text{ds}}^l = \hat{T}_{\text{ds}}^l. \tag{6.8}$$

Note that the implication in (6.8) is handled at the code level and does not require introducing binary variables.

To avoid footstep changes when the swing foot is close to touching the ground, when nearing the end we add the following constraint:

$$T_{\text{ds}}^l + T_{\text{ss}}^l - t_k + t_s^l < t_{\text{change}} \quad \Rightarrow \quad \boldsymbol{f}^{l+1} = \hat{\boldsymbol{f}}^{l+1}. \tag{6.9}$$

6.3.5 Gait feasibility constraints

The gait feasibility constraints are introduced to ensure that IS-MPC is feasible. They do so by constraining the current state to be within the feasibility region (4.18).

The expression of the feasibility region (4.18) uses the ZMP bounds, that clearly depend on the motion of the moving box, and thus on the footsteps positions and timings. To derive a constraint, we simply make this dependency explicit by plugging (4.14) and (4.15) inside (4.18). This results in

$$\begin{aligned}
x_u^k + b_x^k &\leq \boldsymbol{s}^\top \boldsymbol{P}^{-1} \left(\boldsymbol{M}\boldsymbol{X}_f^l + \boldsymbol{m} x_f^l + \boldsymbol{p} \left(\frac{d_x}{2} - x_z^k \right) \right) \\
y_u^k + b_y^k &\leq \boldsymbol{s}^\top \boldsymbol{P}^{-1} \left(\boldsymbol{M}\boldsymbol{Y}_f^l + \boldsymbol{m} y_f^l + \boldsymbol{p} \left(\frac{d_y}{2} - y_z^k \right) \right) \\
z_u^k + b_z^k &\leq \boldsymbol{s}^\top \boldsymbol{P}^{-1} \left(\boldsymbol{M}\boldsymbol{Z}_f^l + \boldsymbol{m} z_f^l + \boldsymbol{p} \left(\frac{d_z}{2} - z_z^k \right) \right).
\end{aligned} \tag{6.10}$$

with $x_u^k, y_u^k, z_u^k, b_x^k, b_y^k, b_z^k, \boldsymbol{s}, \boldsymbol{p}, \boldsymbol{P}, \boldsymbol{m}, \boldsymbol{M}, d_x, d_y, d_z, x_z^k, y_z^k$ and z_z^k already defined in Chapter 4.

6.3.6 Feasibility-driven plan adaptation algorithm

We present two different versions of the FAPA algorithm. The first one is not allowed to move footsteps from a different patch to the one in the original plan, and is thus referred to as Fixed patches FAPA (F-FAPA). The second one is instead allowed to choose different patches, and goes under the name of Variables patches FAPA (V-FAPA).

The decision variable over the planning horizon are collected as

$$\begin{aligned}
\boldsymbol{X}_f^l &= (x_f^l, \ldots, x_f^{l+F}), \quad \boldsymbol{Y}_f^l = (y_f^l, \ldots, y_f^{l+F}),\\
\boldsymbol{Z}_f^l &= (z_f^l, \ldots, z_f^{l+F}), \quad \boldsymbol{\Theta}_f^l = (\theta_f^l, \ldots, \theta_f^{l+F}),\\
\boldsymbol{T}_{\text{ds}}^l &= (T_{\text{ds}}^l, \ldots, T_{\text{ds}}^{l+F}), \quad \boldsymbol{T}_{\text{ss}}^l = (T_{\text{ss}}^l, \ldots, T_{\text{ss}}^{l+F}),
\end{aligned}$$

$$\boldsymbol{B}^l = \begin{pmatrix} b_{l,1} & \ldots & b_{l,L} \\ \vdots & \ddots & \vdots \\ b_{l+F,1} & \ldots & b_{l+F,L} \end{pmatrix},$$

while the corresponding candidate values are identified by the vectors $\hat{\boldsymbol{X}}_f^l$, $\hat{\boldsymbol{Y}}_f^l$, $\hat{\boldsymbol{Z}}_f^l$, $\hat{\boldsymbol{\Theta}}_f^l$, $\hat{\boldsymbol{T}}_{\text{ds}}^l$, $\hat{\boldsymbol{T}}_{\text{ss}}^l$, $\hat{\boldsymbol{B}}^l$, similarly defined.

F-FAPA solves the following problem, with decision variables $\boldsymbol{U}^l = (\boldsymbol{X}_f^l, \boldsymbol{Y}_f^l, \boldsymbol{Z}_f^l, \boldsymbol{\Theta}_f^l, \boldsymbol{T}_{\text{ds}}^l, \boldsymbol{T}_{\text{ss}}^l)$:

$$\begin{aligned}
\min_{\boldsymbol{U}^l} \quad & w_x \|\hat{\boldsymbol{X}}_f^l - \boldsymbol{X}_f^l\|^2 + w_y \|\hat{\boldsymbol{Y}}_f^l - \boldsymbol{Y}_f^l\|^2 + \\
& w_z \|\hat{\boldsymbol{Z}}_f^l - \boldsymbol{Z}_f^l\|^2 + w_\theta \|\hat{\boldsymbol{\Theta}}_f^l - \boldsymbol{\Theta}_f^l\|^2 + \\
& w_{\text{ds}} \|\hat{\boldsymbol{T}}_{\text{ds}}^l - \boldsymbol{T}_{\text{ds}}^l\|^2 + w_{\text{ss}} \|\hat{\boldsymbol{T}}_{\text{ss}}^l - \boldsymbol{T}_{\text{ss}}^l\|^2
\end{aligned}$$

subject to:

- kinematic constraints (6.1), for $j = 1, \ldots, F$
- timing constraints (6.2), for $j = 0, \ldots, F$
- fixed patch constraints (6.3), for $j = 1, \ldots, F$
- current footsteps constraints (6.7), (6.8) and (6.9)
- gait feasibility constraints (6.10)

Since F-FAPA does not have binary variables, it can be implemented using a regular nonlinear solver (i.e., `ipopt`).

V-FAPA solves the following problem, with decision variables which now include the binary variables $\boldsymbol{B}^l$ that is $\boldsymbol{W}^l = (\boldsymbol{X}_f^l, \boldsymbol{Y}_f^l, \boldsymbol{Z}_f^l, \boldsymbol{\Theta}_f^l, \boldsymbol{T}_{\text{ds}}^l, \boldsymbol{T}_{\text{ss}}^l, \boldsymbol{B}^l)$:

$$
\begin{aligned}
\min_{\boldsymbol{W}^l} \quad & w_x \|\hat{\boldsymbol{X}}_f^l - \boldsymbol{X}_f^l\|_2^2 + w_y \|\hat{\boldsymbol{Y}}_f^l - \boldsymbol{Y}_f^l\|_2^2 + \\
& w_z \|\hat{\boldsymbol{Z}}_f^l - \boldsymbol{Z}_f^l\|_2^2 + w_\theta \|\hat{\boldsymbol{\Theta}}_f^l - \boldsymbol{\Theta}_f^l\|_2^2 + \\
& w_{\mathrm{ds}} \|\hat{\boldsymbol{T}}_{\mathrm{ds}}^l - \boldsymbol{T}_{\mathrm{ds}}^l\|_2^2 + w_{\mathrm{ss}} \|\hat{\boldsymbol{T}}_{\mathrm{ss}}^l - \boldsymbol{T}_{\mathrm{ss}}^l\|_2^2 + \\
& w_{\mathrm{b}} \|\hat{\boldsymbol{B}}^l - \boldsymbol{B}^l\|_2^2
\end{aligned}
$$

subject to:

- kinematic constraints (6.1), for $j = 1, \ldots, F$
- timing constraints (6.2), for $j = 0, \ldots, F$
- variable patch constraints (6.5) and (6.6), for $j = 1, \ldots, F$
- current footsteps constraints (6.7), (6.8) and (6.9)
- gait feasibility constraints (6.10)

Since V-FAPA contains the binary variables $\boldsymbol{B}$ it is implemented as a MINLP.

6.4 Simulations

We ran four simulations in MATLAB, using CoppeliaSim to kinematically visualize the resulting motions. The system is an AMD Ryzen 9 5900X (4.8 GHz, 12 core) with 16 GB DDR4 3600 MHz running Ubuntu 22.04 LTS. IS-MPC runs at 100 Hz and is solved using `quadprog`, while FAPA runs at 10 Hz and is solved using the CasADi interface. In CasADi, we used `ipopt` for F-FAPA, and `bonmin` for V-FAPA. We also ran tests with the commercial solver `knitro`, to compare the performance (see Table 6.2).

All the simulations use the parameters of Table 6.1. Simulation videos are available at `https://youtu.be/4_QYsZH1E7Y`.

Simulations take place in 3 different scenarios: *empty*, which is completely flat with no obstacles, and is represented using a single patch; *2-patches* is constituted by two patches at different heights (0 and 0.06 [m]); *stairs* has a total of 7 patches of increasing height. While walking, the robot is subject to impulsive pushes (lasting 0.01 [s]), transformed in equivalent acceleration imparted on the CoM.

In the first simulation, the robot is walking forward in the *empty* scenario. At 4.5 [s] it receives a 15.6 [m/s^2] push in the direction $(-2, -1, 0)$, that without FAPA would make the MPC infeasible. F-FAPA reacts by adapting footstep positions, orientations and timings concurrently, allowing the MPC to recover feasibility. Figure 6.4 shows nominal and adapted footsteps, trajectories and step timings. Figure 6.5 shows a sequence of snapshots of the HRP-4 humanoid robot executing the motion.

In the second simulation (shown in Fig. 6.6), the scenario is *2-patches*, and the robot must climb a step. Upon receiving the push, the footsteps do not change significantly, because the F-FAPA algorithm is not allowed to move the footstep to the other patch. As a result, the tolerable push is smaller, i.e., 7.8 [m/s^2]. Figure 6.7 shows a sequence of snapshots of the HRP-4 humanoid robot executing the motion.

Symbol	Value
δ	0.01 [s]
T_c	2.0 [s]
T_p	4.0 [s]
η	3.6 [s^{-1}]
β	100
d_x, d_y, d_z	0.035 [m]
F	3
$\boldsymbol{v}_1$	$(0.28, 0.13)^\top$ [m]
$\boldsymbol{v}_2$	$(0.2, 0.43)^\top$ [m]
$\boldsymbol{v}_3$	$(-0.12, 0.43)^\top$ [m]
$\boldsymbol{v}_4$	$(-0.2, 0.13)^\top$ [m]
Δ_z^m	-0.10 [m]
Δ_z^M	0.10 [m]
Δ_θ^m	-0.4 [rad]
Δ_θ^M	0.4 [rad]
$T_{\text{ds}}^{\min}$	0.3 [s]
$T_{\text{ds}}^{\max}$	0.5 [s]
$T_{\text{ss}}^{\min}$	0.5 [s]
$T_{\text{ss}}^{\max}$	0.7 [s]
t_{change}	0.1 [s]
M	100
w_x, w_y, w_z, w_θ	1.0
$w_{\text{ds}}, w_{\text{ss}}$	1.0
w_b	0.01

Table 6.1. Hyperparameters of F-FAPA and V-FAPA used in all the experiments.

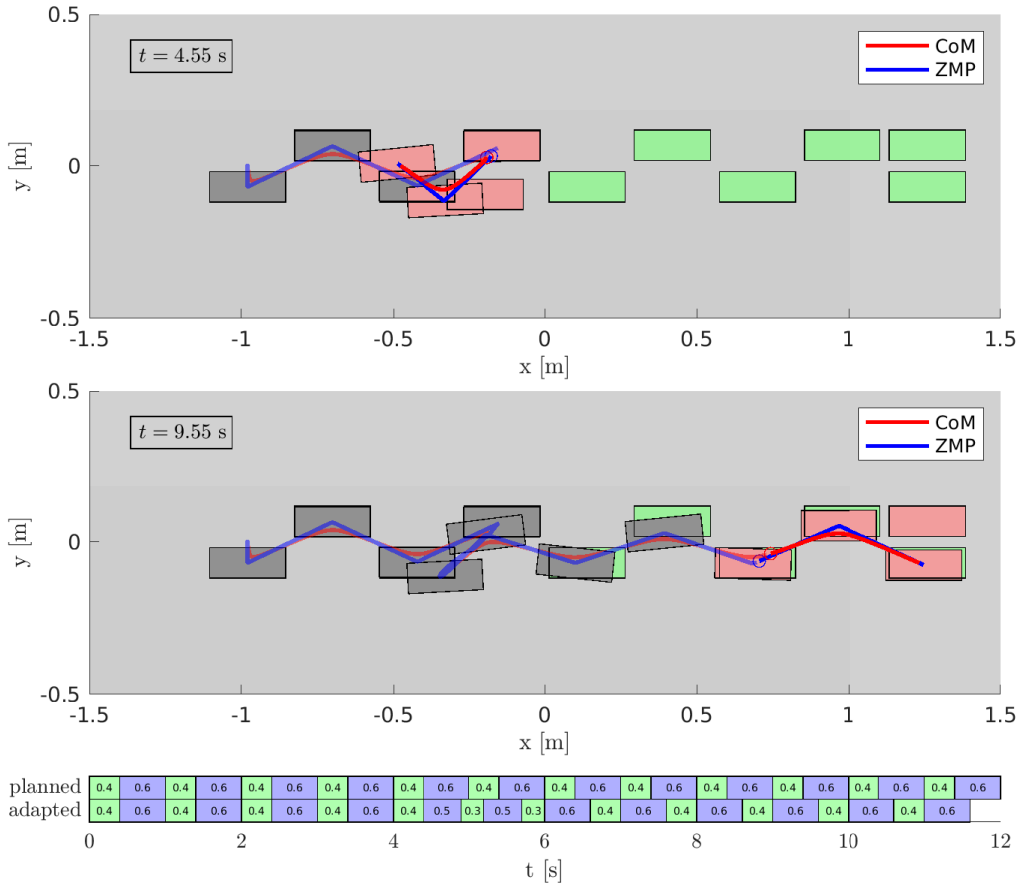

Figure 6.4. F-FAPA in the *empty* scenario. The robot is walking in a straight line and is pushed at time 4.5 [s] (slightly before the first snapshot). Green footsteps represent the original candidate plan, while the footsteps that are actually executed are shown in grey. Red footsteps represent the current adapted subplan. The two bands on the bottom show the nominal and adapted timings (green for double support and blue for single support). The same color scheme is used for the rest of the figures.

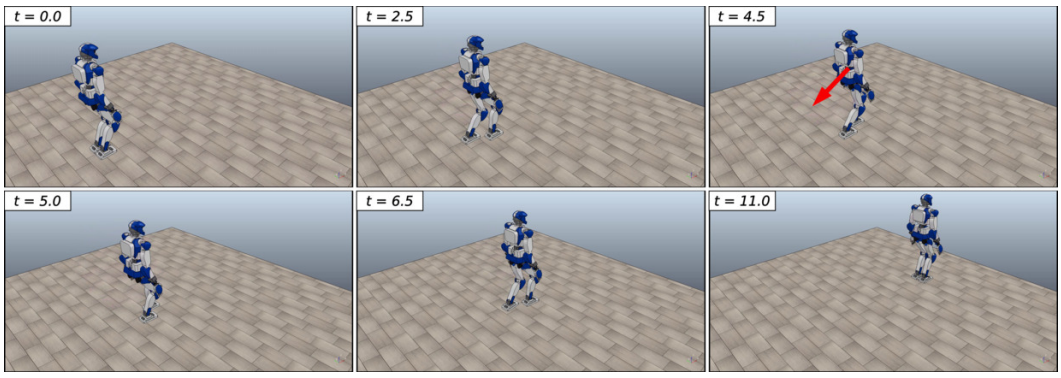

Figure 6.5. HRP-4 walking in the *empty* scenario using F-FAPA. The robot walks in a straight line and it is pushed at time 4.5 [s] (third snapshot). The robot is able to sustain the push adapting the footsteps and the duration of single and double support (fourth snapshot), eventually reaching its desired goal (last snapshot).

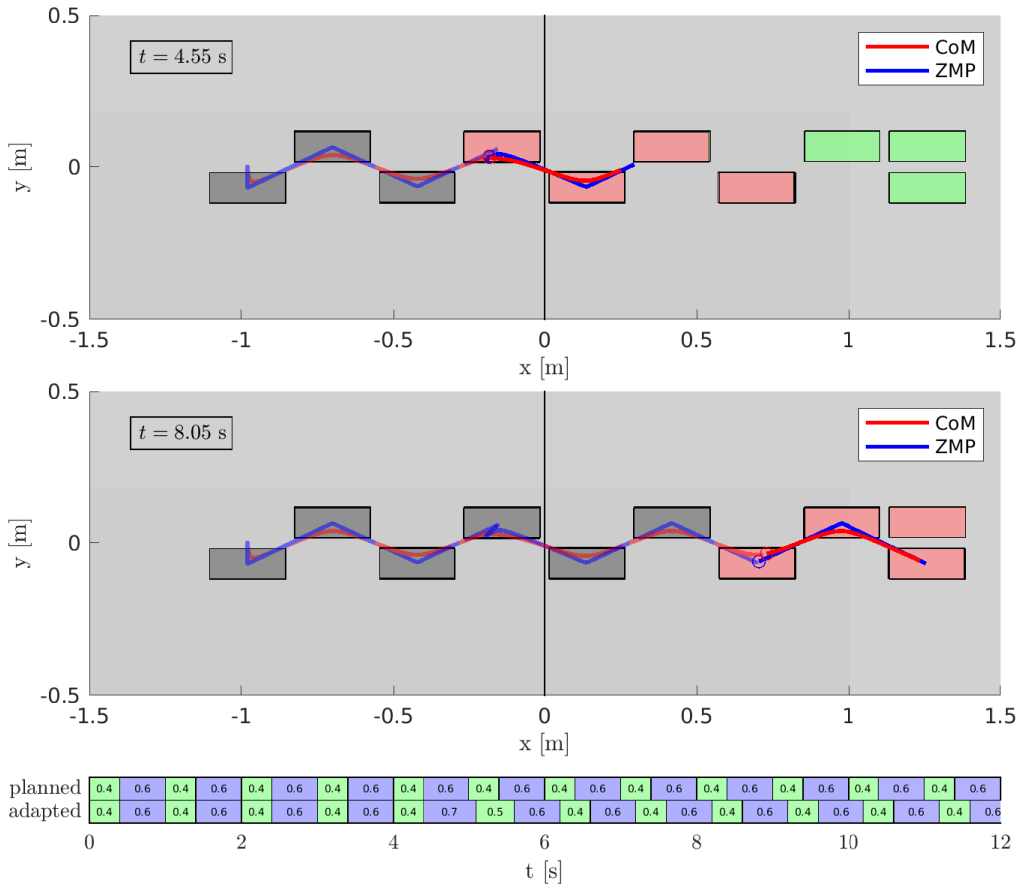

Figure 6.6. F-FAPA in the *2-patches* scenario. The robot is walking in a straight line and is pushed at time 4.5 [s] (slightly before the first snapshot). Since changing patches is not allowed, the magnitude of the push that can be tolerated is quite small, compared to that of the other simulations.

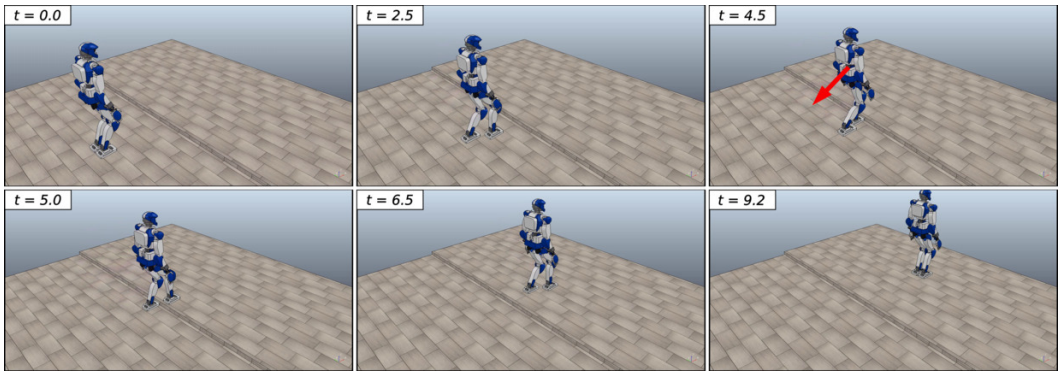

Figure 6.7. HRP-4 walking in the *2-patches* scenario using F-FAPA. The robot walks in a straight line and it is pushed at time 4.5 [s] (third snapshot). The robot is able to sustain the push adapting the duration of single and double support (fourth snapshot), eventually reaching its desired goal (last snapshot).

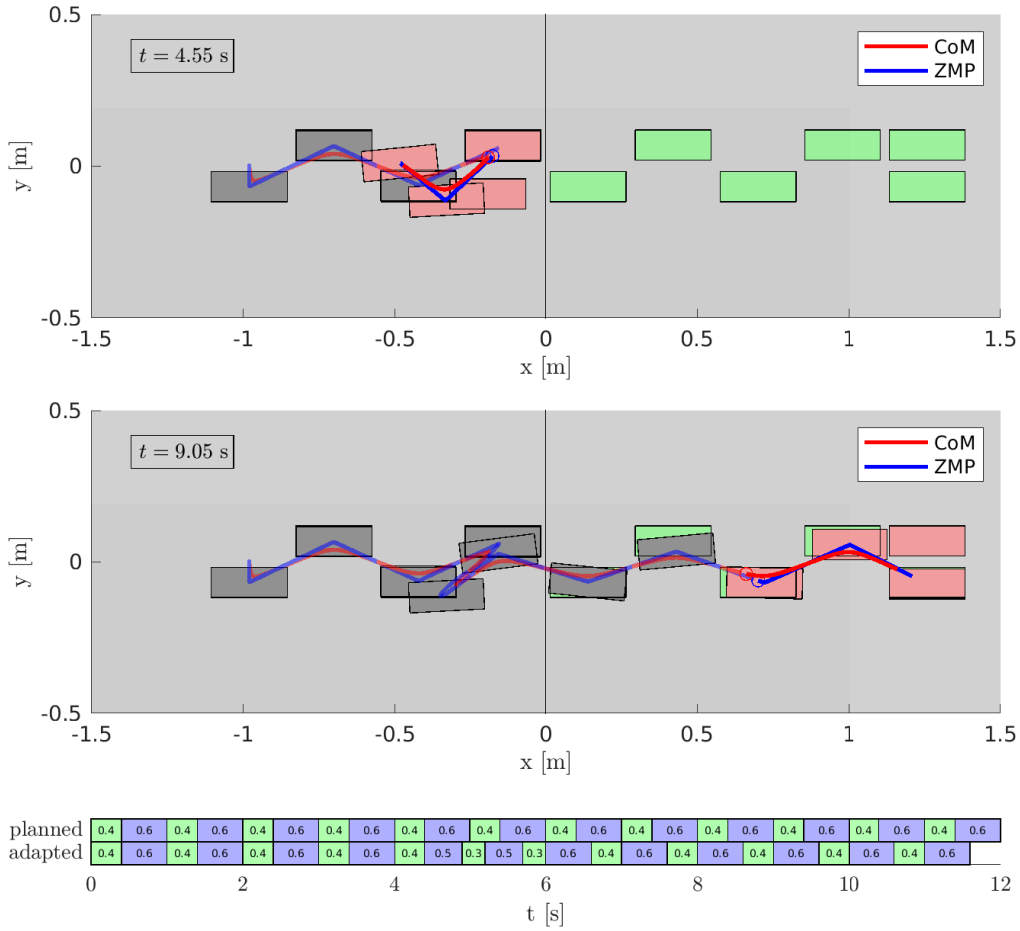

Figure 6.8. V-FAPA in the *2-patches* scenario. The robot is walking in a straight line and is pushed at time 4.5 [s] (slightly before the first snapshot). Now the robot is allowed to adapt the footstep position to the other patch, and is able to tolerate a stronger push.

In the third simulation (shown in Fig. 6.8), the scenario is still *2-patches*, but now the scheme is using V-FAPA. When the push is perceived, the first predicted footstep is moved to the lower patch, and as a result the increase of the tolerable push intensity is very significant, i.e., the same as in the *empty* scenario. Figure 6.10 shows a sequence of snapshots of the HRP-4 humanoid robot executing the motion.

In the last simulation, the robot is moving through a more complex environment constituted by a long staircase. While climbing, the robot is subject to multiple pushes, triggering several footstep adjustments. Figure 6.5 shows a sequence of snapshots of the HRP-4 humanoid robot executing the motion.

To discuss the real-time applicability of the scheme, we report performance metrics in Table 6.2. The solvers used are `ipopt` and `knitro` for F-FAPA, and `bonmin` and `knitro` for V-FAPA. `knitro` is faster overall, but `ipopt` still demonstrates good performance for F-FAPA, compatible with real-time requirements. For V-FAPA, `bonmin` is clearly too slow, while `knitro` has an average performance that is real-time on average, but some outliers violate the requirements. Since all results in this chapter are simulated, real-time performance is desirable but not critical. However,

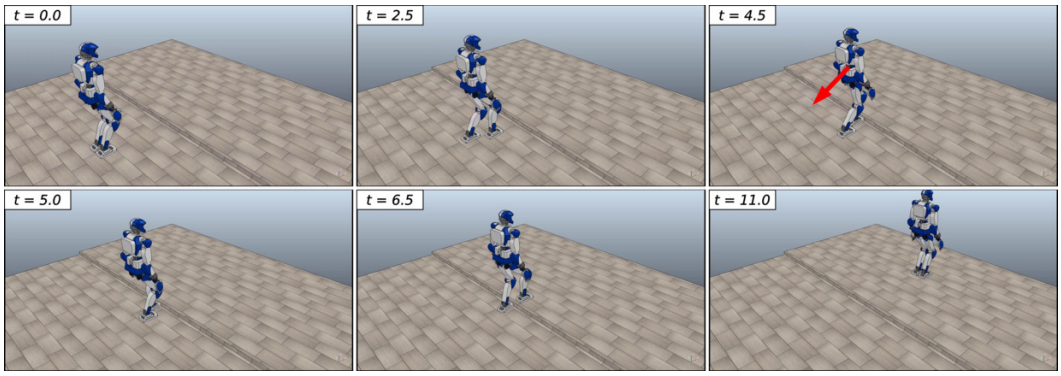

Figure 6.9. HRP-4 walking in the *2-patches* scenario using V-FAPA. The robot walks in a straight line and it is pushed at time 4.5 [s] (third snapshot). The robot is able to sustain the push adapting the footsteps and the duration of single and double support (fourth snapshot), eventually reaching its desired goal (last snapshot). Notice how the robot is able to sustain a stronger push by placing the foot in the first patch.

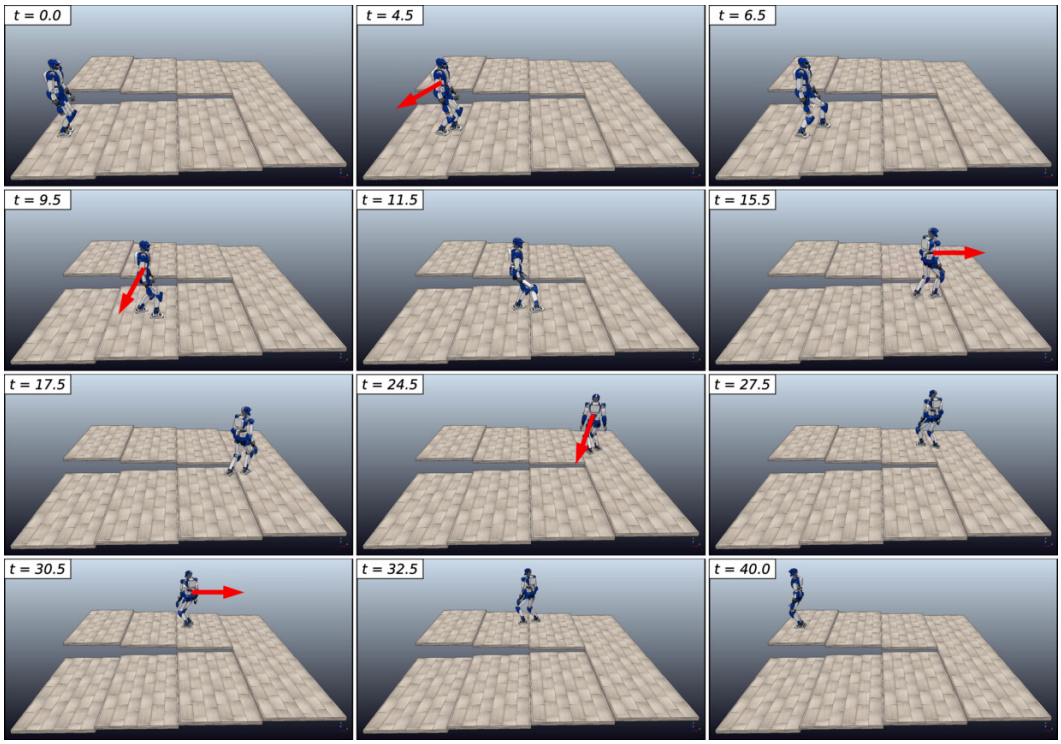

Figure 6.10. HRP-4 walking in a scenario composed of multiple staircases using V-FAPA. The robot follows a footstep plan and it is pushed multiple times during the execution of the motion (second, fourth, sixth , eight and tenth snapshot). The robot is able to sustain different pushes adapting the footsteps and the duration of single and double support, and changing the patch when necessary. Eventually, the robot is able the reach the final patch.

Algorithm	Solver	Average [s]	Std dev. [s]	Max [s]
F-FAPA	`ipopt`	0.0207	0.0041	0.0467
F-FAPA	`knitro`	0.0144	0.0032	0.0329
V-FAPA	`bonmin`	0.3164	0.2075	1.2098
V-FAPA	`knitro`	0.0316	0.0393	0.3985

Table 6.2. Performance metrics of F-FAPA in the *empty* scenario and V-FAPA in the *2-patches* scenario, using different solvers.

it is necessary for hardware implementation, which is why we will be working to guarantee real-time performance in future works.

Part II

Motion control for steerable wheeled mobile robots

Chapter 7

Nonlinear model predictive control based on real-time iteration

Model Predictive Control (MPC) is a technique used to control complex constrained systems that, in the fast few years, has seen an increased interest in both acadamia and industry. It works by solving on-line a finite horizon Optimal Control Problem (OCP), which considers a prediction model of the system, its constraints, the constraints on the control inputs, and a cost function to be minimized (which reflects the desired behavior of the system itself). At each timestep, the controller receives a new measurement of the state of the system, and it uses it to solve the OCP, obtaining an optimal control action, which is applied to the system for a small time interval [69]. These steps are repeated at each new measurement of the state, which acts as feedback of the MPC itself.

Among the categories of MPC, Linear MPC (LMPC) is the most common one. Indeed, because in LMPC the prediction model and the constraints are linear, and the function is quadratic, the associated OCP can be efficienly solved via a Quadratic Programming (QP) problem [70]. Nevertheless, most of the systems that we are interested in controlling are nonlinear. That is particularly true in robotics, where the systems exhibit both a nonlinear model, and nonlinear constraints. While, in principle, it is possible to apply a LMPC to a nonlinear system by linearizing it around a reference trajectory, it is often preferred to treat nonlinear dynamics and constraints explicitely.

Nonlinear MPC (NMPC) [71] allows to use nonlinear models, constraints, and cost functions, at the price of higher computational cost with respect to LMPC. While this makes it more complex to deploy such kind of techniques on real platforms, recent progress in efficient algorithms [72], and availability of powerful processing power, is making the adoption of NMPC more and more common.

This chapter gives a brief overview on Nonlinear MPC, focusing in particular on the real-time iteration scheme [70]. The algorithms presented here will be used in Chapter 8 to develop a Nonlinear MPC for the control of the motion of a steerable wheeled mobile robot.

7.1 Optimal control problem and MPC formulation

Consider a robotic system whose equations of motion are described by the ordinary differential equation

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{u}(t)), \tag{7.1}$$

with $\boldsymbol{q} \in \mathbb{R}^n$ state of the system, $\boldsymbol{u} \in \mathbb{R}^m$ control inputs, and $\boldsymbol{q}(t_0) = \boldsymbol{q}_0$ initial state of the system itself. Moreover, assume that the system is subject to constraints of the kind

$$\boldsymbol{h}(\boldsymbol{q}(t), \boldsymbol{u}(t)) \leq \boldsymbol{0}, \tag{7.2}$$

which collect linear constraints (e.g., hardware limitations due to actuators), and nonlinear constraints (e.g., collision avoidance constraints due to complex environments) on the state and the control inputs of the system[1].

At each time instant t_k, given the estimate $\hat{\boldsymbol{q}}_k$ of the system, the NMPC solves an OCP over a finite horizon $[t_k, t_k + T]$, taking into account the prediction model (7.1) and the constraints (7.2). The OCP can be formulated as

$$\begin{aligned} \min_{\boldsymbol{u}(\cdot)} \quad & \Phi(\boldsymbol{q}(t_k + T)) + \int_{t_k}^{t_k+T} \mathcal{L}(\boldsymbol{q}, \boldsymbol{u}) dt \\ \text{s.t.} \quad & \dot{\boldsymbol{q}}(t) = \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{u}(t)) \\ & \boldsymbol{h}(\boldsymbol{q}(t), \boldsymbol{u}(t)) \leq \boldsymbol{0} \\ & \boldsymbol{q}(t_k) = \hat{\boldsymbol{q}}_k, \end{aligned} \tag{7.3}$$

with $\Phi(\boldsymbol{q}(t_k + T))$ terminal cost, and $\mathcal{L}(\boldsymbol{q}, \boldsymbol{u})$ stage cost. The solution of the OCP gives an optimal control input $\boldsymbol{u}_k^*(t)$, which is defined over the prediction horizon $[t_k, t_k + T]$.

Given $\boldsymbol{u}_k^*(t)$, the NMPC extracts the control input $\boldsymbol{u}_k^{\text{NMPC}}$, corresponding to the subinterval $[t_k, t_k + \delta]$, with δ sampling time of the NMPC. The control action $\boldsymbol{u}_k^{\text{NMPC}}$ is then applied to the system over the same interval $[t_k, t_k + \delta]$. This steps, which describes the NMPC algorithm, summarized in Algorithm 3, are repeated each time a new estimate of the state is available.

Algorithm 3: NMPC algorithm

```
k ← 0;
while true do
    t_k ← kδ;
    q̂_k ← receive the estimated state of the system at time t_k;
    u*_k(t) ← solve the OCP (7.3) over t ∈ [t_k, t_k + T];
    u^NMPC_k ← extract from u*_k(t) the optimal control input corresponding to time
      interval [t_k, t_k + δ];
    apply u^NMPC_k to the system over the time interval [t_k, t_k + δ];
    k ← k + 1;
end
```

[1] We choose not to separate the notation on linear and nonlinear constraints to make the description of the chapter simpler. Bear in mind, however, that modern implementations take it into account in order to improve the performance of the software [73].

7.2 Numerical methods for the OCP solution

The solution of the OCP (7.3) is fundamental for the NMPC algorithm. Indeed, the method chosen determines the performance and the behavior of the NMPC itself.

In literature, there exists three classes for the numerical solution of the OCP.

The first one is based on *dynamic programming* and *Hamilton-Jacobi-Bellman* equation [74], and requires the solution of a partial differential equation. These methods, however, suffer from the curse of dimensionality and they are suitable only to system whose state dimension is small.

The second class are the *indirect methods*, which are based on Pontryagin's minimum principle [75], and consist in solving a multi-point boundary value problem for an ordinary differential equation.

The third class are *direct methods*, and consists in transcribing the OCP (which is infinite-dimesional and continuous-time) into a Nonlinear Programming Problem (NLP). In literature, there exists three transcription approaches (*direct collocation*, *single shooting* and *multiple shooting*), whose difference depends on the way in which the state and the conrol inputs are discretized.

Direct collocation discretizes both state and control inputs using piecewise continuous polynomials, resulting in a large and sparse NLP [76]. Single shooting discretizes only the control inputs, resulting in a smaller NLP (whose decision variables are the control inputs themselves), which is however sensitive to nonlinearities and instabilities of the system [77]. Multiple shooting discretizes both the state and the control inputs, obtaining a large and sparse NLP (which is smaller then the one obtained through collocation), guaranteeing continuity of the solution by adding appropriate constraints on the shooting nodes [78].

In the following, we give a brief overview on multiple shooting, which will be used in Chapter 8 for the transcription of the OCP associated to the trajectory tracking problem of a steerable wheeled mobile robot.

7.2.1 Multiple shooting

In order to discretize the OCP (7.3) using the multiple shooting method, the prediction horizon $[t_k, t_k + T]$ is partitioned into N subintervals

$$t_k < t_{k+1} < \cdots < t_{k+N} = t_k + T, \tag{7.4}$$

where, typically, each subinterval has the same duration, and it is equal to the sampling time of the NMPC, i.e. $t_{k+i+1} = t_{k+i} + \delta$. The control trajectory $\boldsymbol{u}(t)$ is discretized assuming it is piecewise constant over each subinterval, i.e.,

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

with $\boldsymbol{u}_{k|i} \in \mathbb{R}^m$.

The prediction model (7.1) is discretized over each subinterval by using a numerical integration method [79] (typically the fourth order Runge-Kutta method). Denoting by $\boldsymbol{F}(\cdot)$ the discrete-time dynamics of the system (7.1), the prediction model is discretized in the following way:

$$\boldsymbol{q}_{i+1|k} = \boldsymbol{F}(\boldsymbol{q}_{i|k}, \boldsymbol{u}_{i|k}), \ \forall i \in \mathbb{I}_0^{N-1}$$

with $\mathbb{I}_a^b = \{a, \ldots, b\} \subset \mathbb{N}$ subset of natural numbers containing all naturals from a to b. The cost function of the OCP and the constraints (7.2) are evaluated at the corresponding nodes of the subintervals (7.4).

The infinite-dimensional OCP (7.3) is hence transformed into the following finite-dimensional NLP:

$$\begin{aligned} \min_{\boldsymbol{Q}_k, \boldsymbol{U}_k} \quad & \Phi(\boldsymbol{q}_{N|k}) + \sum_{i=0}^{N-1} \mathcal{L}(\boldsymbol{q}_{i|k}, \boldsymbol{u}_{i|k}) \\ \text{s.t.} \quad & \boldsymbol{q}_{i+1|k} = \boldsymbol{F}(\boldsymbol{q}_{i|k}, \boldsymbol{u}_{i|k}),\ \forall i \in \mathbb{I}_0^{N-1} \\ & \boldsymbol{h}(\boldsymbol{q}_{i|k}, \boldsymbol{u}_{i|k}) \leq \boldsymbol{0},\ \forall i \in \mathbb{I}_0^{N-1} \\ & \boldsymbol{q}_{0|k} = \boldsymbol{q}_k, \end{aligned} \tag{7.5}$$

with

$$\begin{aligned} \boldsymbol{Q}_k &= (\boldsymbol{q}_{0|k}^\top, \boldsymbol{q}_{1|k}^\top, \ldots, \boldsymbol{q}_{N|k}^\top)^\top \\ \boldsymbol{U}_k &= (\boldsymbol{u}_{0|k}^\top, \boldsymbol{u}_{1|k}^\top, \ldots, \boldsymbol{u}_{N-1|k}^\top)^\top \end{aligned}$$

collecting the decision variables of the NLP.

Most of the optimization solvers use methods such as Interior-Point method and Sequential Quadratic Programming (SQP) [80] to solve the NLP (7.5). In this manuscript, we are going to solve the NMPC using the real-time iteration scheme [70], which is based on SQP.

7.2.2 Sequential Quadratic Programming

SQP is an iterative method for the solution of constrained nonlinear optimization. At each κ-th iteration, it solves a Quadratic Programming (QP) problem which approximates the NLP (7.5). This procedure is repeated until a convergence criterion is satisfied.

Let us simplify the notation of the NLP (7.5) by expressing it as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{f}(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}, \hat{\boldsymbol{x}}_k) = \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} \end{aligned} \tag{7.6}$$

with $\boldsymbol{x}$ collecting decision variables $\boldsymbol{Q}_\kappa$ and $\boldsymbol{U}_\kappa$, $\boldsymbol{f}(\boldsymbol{x})$ cost function, $\hat{\boldsymbol{x}}_k$ estimate of the state at t_k, $\boldsymbol{g}(\boldsymbol{x}, \hat{\boldsymbol{x}}_k)$ collecting equality constraints, and $\boldsymbol{h}(\boldsymbol{x})$ collecting inequality constraints.

The NLP (7.6), at each κ-th iteration of SQP, is approximated by the following QP:

$$\begin{aligned} \min_{\Delta \boldsymbol{x}_\kappa} \quad & \frac{1}{2} \Delta \boldsymbol{x}_\kappa^\top \boldsymbol{H}_\kappa \Delta \boldsymbol{x}_\kappa + \nabla \boldsymbol{f}(\boldsymbol{x}_\kappa)^\top \Delta \boldsymbol{x}_\kappa \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}_\kappa, \hat{\boldsymbol{x}}_k) + \nabla \boldsymbol{g}(\boldsymbol{x}_\kappa)^\top \Delta \boldsymbol{x}_\kappa = \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}_\kappa) + \nabla \boldsymbol{h}(\boldsymbol{x}_\kappa)^\top \Delta \boldsymbol{x}_\kappa \leq \boldsymbol{0} \end{aligned} \tag{7.7}$$

with $\nabla \boldsymbol{g}(\boldsymbol{x}_\kappa)$ gradient of the equality constraints, $\nabla \boldsymbol{h}(\boldsymbol{x}_\kappa)$ gradient of the inequality constraints, $\nabla \boldsymbol{f}(\boldsymbol{x}_\kappa)$ and $\boldsymbol{H}_\kappa$ gradient and Hessian of the cost function. Note that

$\nabla \boldsymbol{g}(\boldsymbol{x}_\kappa)$ does not depend on $\hat{\boldsymbol{x}}_k$. Moreover, because the computation of the exact Hessian is computationally expensive, it is typically approximated.

The solution $\Delta \boldsymbol{x}_\kappa$ of the κ-th QP gives a direcion $\Delta \boldsymbol{x}_k$, which is used to compute the iterate of the subsequent SQP step:

$$\boldsymbol{x}_{\kappa+1} = \boldsymbol{x}_\kappa + \alpha_k \Delta \boldsymbol{x}_\kappa \tag{7.8}$$

with $\alpha_k \in [0, 1]$ step size.

Algorithm 4 summarizes the SQP method. Note that the most computationally intensive steps are the computation of sensitivies and the computation of the solution of the QP. Regarding the latter, there exists off-the-shelf solvers which efficiently exploit the sparsity of the problem in order to solve the QP more efficiently (e.g., OSQP [81], qpSWIFT [82]). Another approach is called *condensing* [78], and it consists in reducing the number of variables in the QP by eliminating those corresponding to the state of the system. The resulting QP can be efficiently solved by using dense QP solvers (e.g., qpOASES [83], HPIPM [84]).

Algorithm 4: SQP algorithm at t_k

Input: estimated state of the system $\hat{\boldsymbol{x}}_k$ and an initial guess $\tilde{\boldsymbol{x}}_k$

$\kappa \leftarrow 0$;
$\boldsymbol{x}_\kappa \leftarrow \tilde{\boldsymbol{x}}_k$;
while *convergence criterion not satisfied* **do**
  compute $\boldsymbol{H}_\kappa, \boldsymbol{g}(\boldsymbol{x}_\kappa, \hat{\boldsymbol{x}}_k), \boldsymbol{h}(\boldsymbol{x}_\kappa)$ and sensitivities $\nabla \boldsymbol{f}(\boldsymbol{x}_\kappa), \nabla \boldsymbol{g}(\boldsymbol{x}_\kappa), \nabla \boldsymbol{h}(\boldsymbol{x}_\kappa)$;
  $\Delta \boldsymbol{x}_\kappa \leftarrow$ construct and solve the QP (7.7);
  $\boldsymbol{x}_{\kappa+1} \leftarrow \boldsymbol{x}_\kappa + \alpha_k \Delta \boldsymbol{x}_\kappa$;
  $\kappa \leftarrow \kappa + 1$;
end
return $\boldsymbol{x}_\kappa$;

7.3 The real-time iteration

Iterating SQP algorithm to convergence requires a significant and unspecified amount of time, which could make the computed optimal control input outdated, negatively affecting the performance of the NMPC. Choosing whether solving the NLP to convergence, applying a control input based on outdated information, or applying an approximate solution but using most recent information available, is known as *real-time dilemma* [85].

Luckily, recent advancements in optimal control methods enable NMPC to perform in real-time. In this section, we give a brief overview on the real-time iteration (RTI) scheme. For a more detailed description, please refer to [70].

The main idea behind RTI is to exploit the fact that NMPC solves OCPs which are similar from one iteration to the other. Instead of solving SQP to convergence, the NLP is approximated as a QP and solved only once. Moreover, the Newton step (7.8) is taken with $\alpha_k = 1$, increasing the convergence rate. In RTI, the structure of QP (7.7) is exploited in order to further reduce the feedback time. Indeed, the terms $\boldsymbol{H}_k, \boldsymbol{h}(\boldsymbol{x}_k)$ and the sensitivities $\nabla \boldsymbol{f}(\boldsymbol{x}_k), \nabla \boldsymbol{g}(\boldsymbol{x}_k), \nabla \boldsymbol{h}(\boldsymbol{x}_k)$ could be computed before the estimate $\hat{\boldsymbol{x}}_k$ is available. The real-time iteration scheme is, thus, composed by two

phases: a *preparation phase*, where the QP (7.7) is prepared omitting the estimate $\hat{\boldsymbol{x}}_k$, and a *feedback phase*, where the QP is solved by introducing the estimate as well. Algorithm 5 gives an overview of the RTI scheme for NMPC.

Algorithm 5: RTI for NMPC

Preparation phase performed over $t \in [t_{k-1}, t_k)$
Input: previous solution $\boldsymbol{x}_{k-1}$
1 prepare a guess $\tilde{\boldsymbol{x}}_k$ from $\boldsymbol{x}_{k-1}$;
2 compute $\boldsymbol{H}_k, \boldsymbol{h}(\boldsymbol{x}_k)$ and sensitivities $\nabla \boldsymbol{f}(\boldsymbol{x}_k), \nabla \boldsymbol{g}(\boldsymbol{x}_k), \nabla \boldsymbol{h}(\boldsymbol{x}_k)$;
3 prepare QP (7.7) omitting $\hat{\boldsymbol{x}}_k$;
4 **return** QP (7.7);

Feedback phase performed at t_k
Input: $\hat{\boldsymbol{x}}_k$, QP (7.7)
5 $\Delta \boldsymbol{x}_k \leftarrow$ solve QP (7.7) by introducing $\hat{\boldsymbol{x}}_k$;
6 $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \Delta \boldsymbol{x}_k$;
7 **return** *NMPC solution* $\boldsymbol{x}_{k+1}$;

Chapter 8

Nonlinear model predictive control for steerable wheeled mobile robots

Steerable wheeled mobile robots (SWMRs) are known to be flexibile and robust thanks to their omnidirectionality and the presence of conventional wheels. Nevertheless, their modeling and control is complex, due to the presence of singularities in their representation or in the control scheme.

In this chapter, we consider the problem of trajectory tracking for steerable wheeled mobile robots (SWMRs), equipped with two or more wheels. The robot is required to follow a user-defined reference pose trajectory in an environment free of obstacles, without violating the driving and steering velocity constraints of each wheel. Note that, in order to successfully perform this task, it is of utmost importance to take into account the kinematic singularities of the platform [35]. To solve this problem, we propose a framework which makes use of Nonlinear Model Predictive Control [86]. While many existing work use MPC on differential drive robots [87], on autonomous vehicles such as cars [88] and tractor trailers [89], and on wheeled-legged robots [90], the application of MPC to SWMRs has yet to be explored.

Our NMPC is supported by a *finite state machine*, responsible for starting and stopping the motion of the robot, while guaranteeing that it never encounters any kinematic singularity, and an *auxiliary trajectory generation scheme* based on dynamic feedback linearization [91], which generates reference configurations and reference control inputs for the NMPC itself, given the reference pose trajectory. The NMPC is formulated as a Nonlinear Programming problem, and solved using the real-time iteration scheme [70]. Our approach is validated on a Neobotix MPO-700 on trajectories of increasing difficulty.

8.1 Kinematic model

In this section, we will develop the kinematic model of a steerable wheeled mobile robot (SWMR), following the analysis presented in [30]. Note that while our mobile base is equipped with steerable wheels, its kinematic model is identical to the one

Figure 8.1. The Neobotix MPO-700 steerable wheeled mobile robot.

described in [30], which considers caster wheels.

Consider a SWMR equipped with $n_s \geq 2$ independent steerable wheels. With reference to Fig. 8.2, we will denote the vector $\boldsymbol{\xi} = [x, y, \theta]^\top \in SE(2)$ as the pose of mobile base, with (x, y) its position and θ its orientation. Let S_i be the i-th steering joint of the mobile base, and W_i the i-th wheel of the mobile base, and let $\boldsymbol{o}_{S_i}$ and $\boldsymbol{o}_{W_i}$ respectively be their positions, the latter parameterized by the steering angle β_i. Each wheel is also described by two independent velocities, the driving velocity v_{Wi} and the steering velocity v_{β_i}, which are taken as control inputs. We define the whole robot configuration via the vector $\boldsymbol{q} = [\boldsymbol{\xi}^\top, \boldsymbol{\beta}^\top]^\top$, where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_{n_s}]^\top$.

The position of the i-th steering joint S_i is defined as

$$\boldsymbol{o}_{S_i} = \begin{bmatrix} x \\ y \end{bmatrix} + \boldsymbol{R}(\theta) \begin{bmatrix} b_i \\ a_i \end{bmatrix},$$

and the position of the i-th wheel W_i is defined as

$$\boldsymbol{o}_{W_i} = \boldsymbol{o}_{S_i} + \boldsymbol{R}(\theta + \beta_i) \begin{bmatrix} 0 \\ -d \end{bmatrix}.$$

where $\boldsymbol{R} \in SO(2)$ is a rotation matrix.

Due to the assumption of no lateral skidding (i.e., the velocity of the contact point of the wheel must be orthogonal with respect to the zero motion line of the wheel itself), each wheel is subject to the Pfaffian constraint

$$\begin{bmatrix} -\sin(\theta + \beta_i) \\ \cos(\theta + \beta_i) \end{bmatrix}^\top \dot{\boldsymbol{o}}_{W_i} = 0. \tag{8.1}$$

By combining the above equations, it is possible to rearrange the n_s nonholonomic

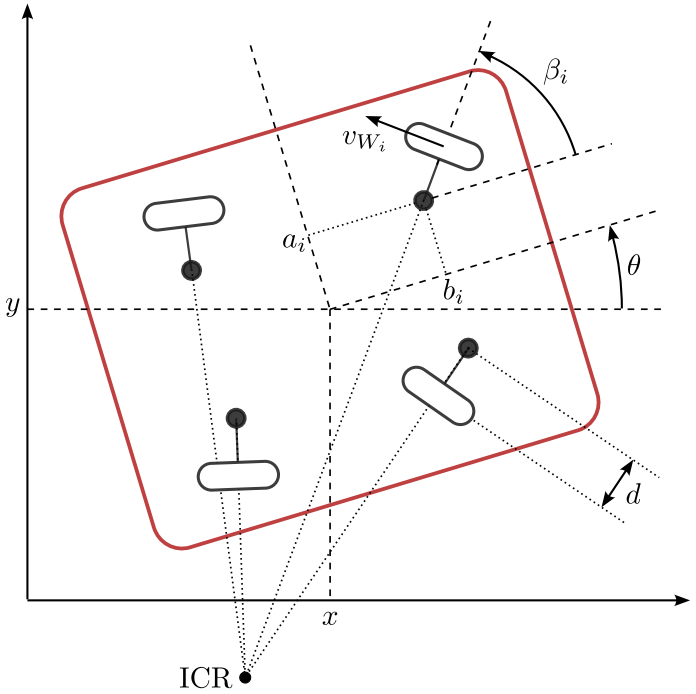

Figure 8.2. Schematic model of a SWMR. Note that, even if the figure represents a robot equipped with four wheels, our approach is generic and works with an arbitrary number of 2 or more wheels.

constraints in matrix form

$$\underbrace{\begin{bmatrix} -\sin(\theta+\beta_1) & \cos(\theta+\beta_1) & \Delta_1 & 0\dots 0 \\ -\sin(\theta+\beta_2) & \cos(\theta+\beta_2) & \Delta_2 & 0\dots 0 \\ \vdots & \vdots & \vdots & \vdots \\ -\sin(\theta+\beta_{n_s}) & \cos(\theta+\beta_{n_s}) & \Delta_{n_s} & 0\dots 0 \end{bmatrix}}_{\boldsymbol{A}^\top(\boldsymbol{q})} \dot{\boldsymbol{q}} = 0, \tag{8.2}$$

with $\Delta_i = b_i \cos\beta_i + a_i \sin\beta_i$.

For the mobile base to perform a motion, all wheel axles must instantaneously intersect at the same point, the ICR. The existence of an ICR can also be seen as a geometric constraint, which requires all wheel orientations to be coordinated. In the following, we will study how the ICR constraint affects the robot mobility.

8.1.1 ICR constraint not satisfied

Whenever the robot configuration is such that it has no instantaneous center of rotation (ICR), since from (8.2) $\dot{\boldsymbol{q}} \in \mathcal{N}(\boldsymbol{A}^\top(\boldsymbol{q}))$, the kinematic model of the robot is the following:

$$\dot{\beta}_i = v_{\beta_i},$$

with v_{β_i} steering velocities. In this case, the pose of the robot remains constant, and it is only possible to control the steering angles.

8.1.2 ICR constraint satisfied

Whenever the robot configuration is such that there exists an ICR, it is possible to simplify (8.2) through the use of *coordinating functions* for β_i [30], with $i \geq 2$. The idea is to let the ICR be defined by the trajectory of $\boldsymbol{\xi}$, namely $\boldsymbol{\xi}(t)$. Indeed, considering the i-th constraint in (8.2) and solving for β_i yields the coordinating function[1] (holonomic constraint)

$$\beta_i = h_i(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) = \arctan \frac{-\sin\theta\dot{x} + \cos\theta\dot{y} + b_i\dot{\theta}}{\cos\theta\dot{x} + \sin\theta\dot{y} - a_i\dot{\theta}}, \tag{8.3}$$

which can be used to "transform" the last $n_s - 1$ constraints of (8.2), obtaining:

$$\begin{bmatrix} -\sin(\theta+\beta_1) & \cos(\theta+\beta_1) & \Delta_1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \beta_1 \end{bmatrix} = 0, \tag{8.4}$$

$$\beta_i = h_i(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}), \quad i = 2, \ldots, n_s.$$

From (8.4), it is trivial to get the reduced kinematic model

$$\begin{aligned} \dot{x} &= v_{W_1}\cos(\theta+\beta_1) + \omega(b_1\sin\theta + a_1\cos\theta) \\ \dot{y} &= v_{W_1}\sin(\theta+\beta_1) + \omega(-b_1\cos\theta + a_1\sin\theta) \\ \dot{\theta} &= \omega \\ \dot{\beta}_1 &= v_{\beta_1}, \end{aligned} \tag{8.5}$$

which can be used together with (8.3) to express β_i as

$$\beta_i = h_i(v_{W_1}, \omega, \beta_1) = \arctan \frac{v_{W_1}\sin\beta_1 + \omega(b_i - b_1)}{v_{W_1}\cos\beta_1 + \omega(a_1 - a_i)}. \tag{8.6}$$

Note that the above equations (named *coordinating functions*) present a singularity whenever the position of the i-th steering joint S_i is constant (i.e., $\dot{\boldsymbol{o}}_{S_i} = \mathbf{0}$). This needs to be considered when designing a controller. Note that this condition is met when the platform is not moving or when the position of the ICR coincides with the position of S_i.

As a consequence, if the ICR does not lie on any of the steering joints S_i and if it does not change through time (i.e. $\dot{\beta}_i = 0$), all coordinating functions h_i are free of singularity. Two interesting cases when these hypotheses hold are when the ICR is constant at infinity (i.e., $\omega = 0$), and when the ICR is constant, not at infinity (i.e., $v_{W_1} = \omega R$, with $R > 0$ the distance between ICR and $\boldsymbol{o}_{W_1}$). Hereby, we define the reduced kinematic models and coordinating functions for these 2 cases, which we use to start/stop the robot.

[1] Note that β_i can have two values, shifted by π. We select the one closest to the current β_i.

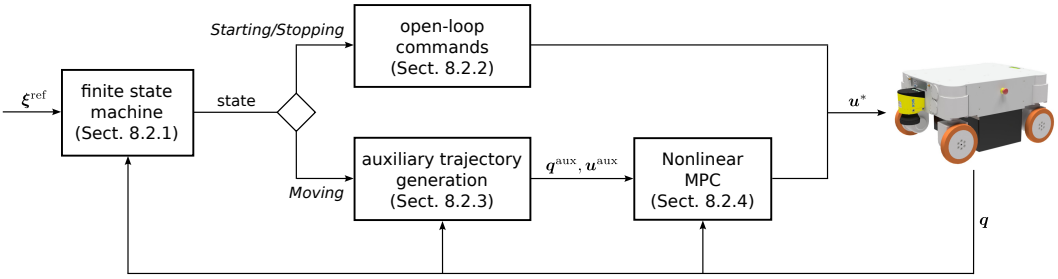

Figure 8.3. Block scheme of the proposed framework. A user-defined reference pose trajectory $\boldsymbol{\xi}^{\text{ref}}$ of the mobile base is fed to a Finite State Machine (FSM), which determines when to start/stop robot motion. As soon as a reference trajectory is available, the state of the FSM becomes *Starting*, and the mobile base is accelerated (using open-loop commands) until all wheel driving velocities are non null. When this condition is met, the state of the FSM becomes *Moving*, and the Nonlinear MPC takes full control of the robot motion. In this case, an auxiliary trajectory generation scheme based on dynamic feedback linearization computes the trajectories $\boldsymbol{q}^{\text{aux}}$ and $\boldsymbol{u}^{\text{aux}}$ (using $\boldsymbol{\xi}^{\text{ref}}$), which are used by the Nonlinear MPC to compute optimal control inputs $\boldsymbol{u}^*$. The state of the FSM becomes *Stopping* when $\boldsymbol{\xi}^{\text{ref}} = \mathbf{0}$, and the robot decelerates, then stops.

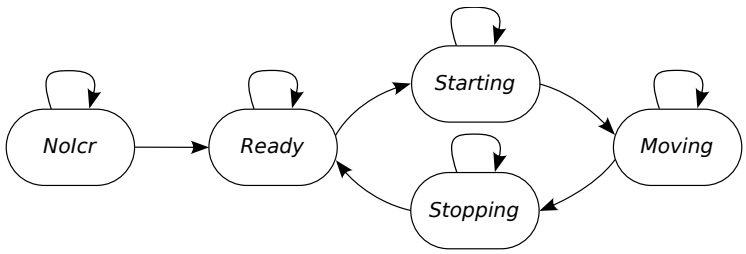

Figure 8.4. Finite state machine defining the motion of the mobile base.

ICR constant at infinity

in this case, the steering angles are the same for all wheels and $\omega = 0$. Then, the robot's reduced kinematic model (8.5) becomes

$$\begin{aligned} \dot{x} &= v_{W_1} \cos(\theta + \beta_1) \\ \dot{y} &= v_{W_1} \sin(\theta + \beta_1), \end{aligned} \tag{8.7}$$

and the coordinating functions become

$$\beta_i = h_i(\beta_1) = \beta_1.$$

The robot can only move along a line parallel to the wheels' sagittal axes.

ICR constant not at infinity

in this case, the robot's reduced kinematic model (8.5) becomes

$$\begin{aligned} \dot{x} &= \omega R \cos(\theta + \beta_1) + \omega(b_1 \sin\theta + a_1 \cos\theta) \\ \dot{y} &= \omega R \sin(\theta + \beta_1) + \omega(-b_1 \cos\theta + a_1 \sin\theta) \\ \dot{\theta} &= \omega, \end{aligned} \tag{8.8}$$

and the coordinating functions become

$$\beta_i = h_i(\beta_1) = \arctan \frac{R \sin\beta_1 + b_i - b_1}{R \cos\beta_1 + a_1 - a_i}.$$

Note that R can be determined from the ICR, which can be computed as the intersection of the wheel axles. In this case, the robot can only move along the circle centered at the ICR, and radius corresponding to the distance between the position of the robot and the ICR itself.

8.2 Proposed framework

This section describes in detail the main components of our framework (shown in Fig. 8.3), namely: the finite state machine, responsible for starting and stopping the robot while avoiding kinematic singularities, the auxiliary trajectory generation scheme, which provides trajectories to the NMPC, and the NMPC itself, which computes control inputs for the robot, while satisfying the driving and steering velocity constraints of each wheel.

8.2.1 Finite state machine

Since the ICR constraint may be not satisfied at initialization, and since the NMPC must avoid configurations in which coordinating functions (8.6) are singular, we designed a finite state machine (FSM) to move the robot towards a configuration free of singularity.

The FSM, shown in Fig. 8.4, consists of five states, described – along with the triggering events – hereby.

- *NoICR.* The configuration of the robot is such that the ICR constraint is not satisfied. In this state, the wheels are regulated to a user-defined configuration using a proportional controller. Once the ICR constraint is satisfied, the state of the FSM becomes *Ready.*
- *Ready.* The configuration of the robot is such that the ICR constraint is satisfied and the robot is not moving. Once a new trajectory is available, the state of the FSM becomes *Starting.*
- *Starting.* An open loop controller makes the robot start its motion taking into account the singularity-free kinematic models previously presented: either (8.7) or (8.8), depending on the robot's initial configuration. Once all velocities $\dot{\boldsymbol{o}}_{S_i}$ become non-null, the state of the FSM becomes *Moving.*
- *Moving.* The robot moves using the NMPC. If the trajectory tracking task is about to be be completed, the state of the FSM becomes *Stopping.*
- *Stopping.* Similarly to *Starting*, an open loop motion makes the mobile base reduce its speed until it stops. Once the robot stops its motion, the state of the FSM becomes *Ready.*

8.2.2 Open-loop commands (starting and stopping)

In this section, we present our singularity-free strategy, for handling starting and stopping motions. To this end, we first constrain the ICR to be constant, as explained in Sect. 8.1.2, and we accelerate (respectively, decelerate) the mobile base along the arc of circle defined by the initial position of the robot and the initial ICR, until all velocities $\dot{\boldsymbol{o}}_{S_i}$ are non-null (respectively, null).

If the ICR is constant at infinity, the robot evolves according to (8.7). Considering the dynamic extension $\dot{v}_{W_1} = a_{W_1}$, with a_{W_1} new control input:

- when the state of the FSM is *Starting*, we accelerate the mobile base by choosing $a_{W_1} = a_{W_1}^{\text{init}}$, where $a_{W_1}^{\text{init}}$ is a parameter;
- when the state of the FSM is *Stopping*, we decelerate the mobile base by choosing $a_{W_1} = -K_{v_{W_1}}^{\text{stop}} v_{W_1}$, with $K_{v_{W_1}}^{\text{stop}} > 0$.

Assuming that the ICR is constant, but not at infinity and not lying on any of the steering joint, the robot evolves according to (8.8). Considering the dynamic extension $\dot{\omega} = a_\omega$, with a_ω new control input:

- when the state of the FSM is *Starting*, we accelerate the mobile base via $a_\omega = a_{W_1}^{\text{init}}/R$, which, since $\dot{v}_{W_1} = a_\omega R$, is equivalent to accelerate W_1 by $a_{W_1}^{\text{init}}$;
- when the state of the FSM is *Stopping*, we decelerate the mobile base via $a_\omega = -K_{v_{W_1}}^{\text{stop}} v_{W_1}/R$, which is equivalent to decelerate W_1 by $-K_{v_{W_1}}^{\text{stop}} v_{W_1}$.

8.2.3 Auxiliary trajectory generation

Once the velocities $\dot{\boldsymbol{o}}_{S_i}$ become non-null, the state becomes *Moving*, and the robot is controlled by the NMPC. In this section, we present the auxiliary trajectory generation scheme based on dynamic feedback linearization [91], which computes auxiliary configurations and control input trajectories for the NMPC, given a reference pose trajectory $\boldsymbol{\xi}^{\text{ref}}$ of the mobile base. Note that both auxiliary trajectory generation and NMPC are only active when the state is *Moving*.

Consider the output function $\boldsymbol{z}(\boldsymbol{q}) = \boldsymbol{\xi}$ and dynamically extend (8.5) by adding the following integrators:

$$\begin{aligned} \dot{v}_{W_1} &= a_{W_1} \\ \dot{\omega} &= a_\omega, \end{aligned}$$

so that $\boldsymbol{u} = [a_{W_1}, a_\omega, v_{\beta_1}]^\top$ are the new control inputs. In the following, unless otherwise specified, we denote the robot configuration with dynamic extension as $\boldsymbol{q} = [x, y, \theta, \beta_1, v_{W_1}, \omega]^\top$. The dynamically extended kinematic model is

$$\begin{aligned} \dot{x} &= v_{W_1}\cos(\theta+\beta_1) + \omega(b_1\sin\theta + a_1\cos\theta) \\ \dot{y} &= v_{W_1}\sin(\theta+\beta_1) + \omega(-b_1\cos\theta + a_1\sin\theta) \\ \dot{\theta} &= \omega \\ \dot{\beta}_1 &= v_{\beta_1} \\ \dot{v}_{W_1} &= a_{W_1} \\ \dot{\omega} &= a_\omega, \end{aligned} \tag{8.10}$$

which, in the following, will be denoted as $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$.

By deriving twice $\boldsymbol{z}(\boldsymbol{q})$, we obtain

$$\ddot{\boldsymbol{z}}(\boldsymbol{q}) = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \boldsymbol{M}(\boldsymbol{q}) + \boldsymbol{H}(\boldsymbol{q}) \begin{bmatrix} a_{W_1} \\ a_\omega \\ v_{\beta_1} \end{bmatrix}, \tag{8.11}$$

with $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^3$ and $\boldsymbol{H}(\boldsymbol{q}) \in \mathbb{R}^{3\times 3}$ defined as

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} -\sin(\theta+\beta_1)\omega v_{W_1} + (b_1\cos\theta - a_1\sin\theta)\omega^2 \\ \cos(\theta+\beta_1)\omega v_{W_1} + (-b_1\sin\theta + a_1\cos\theta)\omega^2 \\ 0 \end{bmatrix},$$

$$\boldsymbol{H}(\boldsymbol{q}) = \begin{bmatrix} \mathrm{c}(\theta+\beta_1) & b_1\mathrm{s}\theta + a_1\mathrm{c}\theta & -\mathrm{s}(\theta+\beta_1)v_{W_1} \\ \mathrm{s}(\theta+\beta_1) & -b_1\mathrm{c}\theta + a_1\mathrm{s}\theta & \mathrm{c}(\theta+\beta_1)v_{W_1} \\ 0 & 1 & 0 \end{bmatrix},$$

where $\mathrm{c}(\cdot)$ and $\mathrm{s}(\cdot)$ respectively denote $\cos(\cdot)$ and $\sin(\cdot)$.

By choosing

$$\boldsymbol{u} = \begin{bmatrix} a_{W_1} \\ a_\omega \\ v_{\beta_1} \end{bmatrix} = \boldsymbol{H}(\boldsymbol{q})^{-1}\left(\boldsymbol{a} - \boldsymbol{M}(\boldsymbol{q})\right),$$

we can transform (8.11) into an equivalent chain of integrators

$$\ddot{\boldsymbol{z}} = \boldsymbol{a},$$

which can be easily stabilized. Indeed, exponential regulation of the trajectory tracking error $\boldsymbol{e}(t) = \boldsymbol{z}^{\text{ref}}(t) - \boldsymbol{z}(t)$, can be achieved by taking

$$\boldsymbol{a} = \ddot{\boldsymbol{z}}^{\text{ref}} + \boldsymbol{K}_P\boldsymbol{e} + \boldsymbol{K}_D\dot{\boldsymbol{e}}, \quad \boldsymbol{K}_P, \boldsymbol{K}_D > 0,$$

with $\boldsymbol{z}^{\text{ref}}(t)$ twice differentiable and persistent (i.e., $v_{W_1} \neq 0$) reference trajectory. Note that the above decoupling matrix $\boldsymbol{H}(\boldsymbol{q})$ is singular at $v_{W_1} = 0$. This kind of singularity is structural for mobile robots [91].

Given the reference trajectories $\boldsymbol{\xi}^{\text{ref}}, \dot{\boldsymbol{\xi}}^{\text{ref}}, \ddot{\boldsymbol{\xi}}^{\text{ref}}$, Algorithm 6 generates, at each timestep t_k, the auxiliary configurations $\boldsymbol{q}^{\text{aux}}_{j|k}$ $(j = 0, \dots, N)$, together with the auxiliary control inputs $\boldsymbol{u}^{\text{aux}}_{j|k}$ $(j = 0, \dots, N-1)$. These will be used by the NMPC, described in the next section, to compute control inputs $(a_{W_1,k}, a_{\omega,k}, v_{\beta_1,k})^T$ for the mobile base. In the pseudocode: function Sample discretizes a trajectory, given over a time interval $[t_k, t_k + N\delta_{\text{MPC}}]$, into $N+1$ elements, with δ_{MPC} timestep of the NMPC, and function $\boldsymbol{F}$ integrates kinematic model (8.10) using fourth-order Runge-Kutta over timestep δ_{MPC}.

Algorithm 6: AuxiliaryTrajectoryGeneration

Input: $\boldsymbol{\xi}^{\text{ref}}, \dot{\boldsymbol{\xi}}^{\text{ref}}, \ddot{\boldsymbol{\xi}}^{\text{ref}}$
Output: $\boldsymbol{q}^{\text{aux}}_{0|k}, \dots, \boldsymbol{q}^{\text{aux}}_{N|k}, \boldsymbol{u}^{\text{aux}}_{0|k}, \dots, \boldsymbol{u}^{\text{aux}}_{N-1|k}$

$\boldsymbol{\xi}^{\text{ref}}_{0|k}, \dots, \boldsymbol{\xi}^{\text{ref}}_{N|k} \leftarrow \text{Sample}(\boldsymbol{\xi}^{\text{ref}})$;
$\dot{\boldsymbol{\xi}}^{\text{ref}}_{0|k}, \dots, \dot{\boldsymbol{\xi}}^{\text{ref}}_{N|k} \leftarrow \text{Sample}(\dot{\boldsymbol{\xi}}^{\text{ref}})$;
$\ddot{\boldsymbol{\xi}}^{\text{ref}}_{0|k}, \dots, \ddot{\boldsymbol{\xi}}^{\text{ref}}_{N|k} \leftarrow \text{Sample}(\ddot{\boldsymbol{\xi}}^{\text{ref}})$;
$\boldsymbol{q}^{\text{aux}}_{0|k} \leftarrow \boldsymbol{q}^{\text{ref}}_{0|k}$;
for $j \leftarrow 0$ **to** $N-1$ **do**
  $\boldsymbol{a}_{j|k} \leftarrow \ddot{\boldsymbol{z}}^{\text{ref}}_{j|k} + \boldsymbol{K}_P(\boldsymbol{z}^{\text{ref}}_{j|k} - \boldsymbol{z}_{j|k}) + \boldsymbol{K}_D(\dot{\boldsymbol{z}}^{\text{ref}}_{j|k} - \dot{\boldsymbol{z}}_{j|k})$;
  $\boldsymbol{u}^{\text{aux}}_{j|k} \leftarrow \boldsymbol{H}(\boldsymbol{q}^{\text{aux}}_{j|k})^{-1}(\boldsymbol{a}_{j|k} - \boldsymbol{M}(\boldsymbol{q}^{\text{aux}}_{j|k}))$;
  $\boldsymbol{q}^{\text{aux}}_{j+1|k} \leftarrow \boldsymbol{F}(\boldsymbol{q}^{\text{aux}}_{j+1|k}, \boldsymbol{u}^{\text{aux}}_{j|k})$;
end
return $\boldsymbol{q}^{\text{aux}}_{0|k}, \dots, \boldsymbol{q}^{\text{aux}}_{N|k}, \boldsymbol{u}^{\text{aux}}_{0|k}, \dots, \boldsymbol{u}^{\text{aux}}_{N-1|k}$;

8.2.4 Nonlinear Model Predictive Control

The Nonlinear MPC solves, at each control cycle, a finite horizon constrained Optimal Control Problem (OCP), taking into account the kinematic model (8.5), wheel velocity and control inputs constraints, singularities of the coordinating functions (8.6), and singularity of the decoupling matrix $\boldsymbol{H}(\boldsymbol{q})$ in the auxiliary trajectory generation scheme. In the following, we will denote as $\mathbb{I}_a^b = \{a, \dots, b\} \subset \mathbb{N}$ the subset of natural numbers containing all naturals from a to b.

The OCP can be defined as

$$\begin{aligned}
\min_{\boldsymbol{u}(\cdot)} \quad & \Phi(\boldsymbol{q}(t_k + T_{\text{MPC}})) + \int_{t_k}^{t_k+T_{\text{MPC}}} \mathcal{L}(\boldsymbol{q}, \boldsymbol{u}) dt \\
\text{s.t.} \quad & \dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}) \\
& v_{W_1} \neq 0 \\
& v_W^- \leq v_{W_i} \leq v_W^+, \ \forall i \in \mathbb{I}_2^{n_s} \\
& \dot{\boldsymbol{o}}_{S_i} \neq \boldsymbol{0}, \ \forall i \in \mathbb{I}_1^{n_s} \\
& a_W^- \leq a_{W_1} \leq a_W^+ \\
& v_\beta^- \leq v_{\beta_i} \leq v_\beta^+, \ \forall i \in \mathbb{I}_1^{n_s} \\
& \boldsymbol{q}(t_k) = \boldsymbol{q}_k,
\end{aligned}$$

with T_{MPC} duration of the prediction horizon, stage and terminal cost respectively defined as

$$\begin{aligned}
\mathcal{L}(\boldsymbol{q}, \boldsymbol{u}) &= \|\boldsymbol{q}^{\text{aux}} - \boldsymbol{q}\|^2_{\boldsymbol{W_q}} + \|\boldsymbol{u}^{\text{aux}} - \boldsymbol{u}\|^2_{\boldsymbol{W_u}} \\
\Phi(\boldsymbol{q}) &= \|\boldsymbol{q}^{\text{aux}} - \boldsymbol{q}\|^2_{\boldsymbol{W_q}},
\end{aligned}$$

$\boldsymbol{W_q}, \boldsymbol{W_u}$ positive semi-definite weighting matrices, v_W^- and v_W^+ min/max wheel driving velocity, a_W^- and a_W^+ min/max wheel driving acceleration, v_β^- and v_β^+ min/max wheel steering velocity and $\boldsymbol{q}_k$ initial configuration.

Note that the velocity constraints are linear for the coordinating wheel (since v_{W_1} and v_{β_1} are part of $\boldsymbol{q}$) and nonlinear for the coordinated wheels. In particular, because of the assumption of no lateral skidding (8.1), the driving velocity of the coordinated wheels can be computed as

$$v_{W_i} = \begin{bmatrix} \cos(\theta + \beta_i) \\ \sin(\theta + \beta_i) \end{bmatrix}^\top \dot{\boldsymbol{o}}_{W_i}.$$

Since the steering angles of the coordinated wheels are defined as $\beta_i = h_i(v_{W_1}, \omega, \beta_1)$, the steering velocities can simply be computed as their time derivatives.

As already mentioned, since the coordinating function h_i is singular when $\dot{\boldsymbol{o}}_{S_i} = \boldsymbol{0}$, it is important to carefully design the control scheme. A simple strategy to make the NMPC free of singularities, is to never let the position of the i-th steering joint be at rest. Since the NMPC is activated only when changing the FSM state from *Starting* to *Moving*, it is possible to constrain $\dot{\boldsymbol{o}}_{S_i}$ so that it is never null. Indeed, the constraint $\dot{\boldsymbol{o}}_{S_i} \neq \boldsymbol{0}$, with a proper change of coordinates, can be rewritten as

$$\boldsymbol{R}(\theta + \beta_i)\dot{\boldsymbol{o}}_{S_i} = \begin{bmatrix} v_{S_i} \\ 0 \end{bmatrix} \neq \boldsymbol{0},$$

with

$$v_{S_i} = \begin{bmatrix} \cos(\theta + \beta_i) \\ \sin(\theta + \beta_i) \end{bmatrix}^\top \dot{\boldsymbol{o}}_{S_i}.$$

To satisfy the above inequality, we need to have $v_{S_i} \neq 0$, which is equivalent to imposing constant $\text{sgn}(v_{S_i})$. Note that, because of the starting motion described in

Sect. 8.2.2, v_{S_i} is either positive or negative when the NMPC is activated. This implies that the constraint will simply be

$$\begin{cases} v_{S_i} > 0, & \text{if } v_{S_i}(t_0) > 0 \\ v_{S_i} < 0, & \text{otherwise} \end{cases},$$

with t_0 time of activation of the NMPC. A similar reasoning can be done for the constraint $v_{W_1} \neq 0$, which guarantees that subsequent calls of the auxiliary trajectory generation scheme are free of singularities.

We can transcribe the above OCP into the following nonlinear programming (NLP) problem by using multiple shooting [78]:

$$\begin{aligned}
\min_{\boldsymbol{Q}_k, \boldsymbol{U}_k} \; & \Phi(\boldsymbol{q}_{N|k}) + \sum_{j=0}^{N-1} \mathcal{L}(\boldsymbol{q}_{j|k}, \boldsymbol{u}_{j|k}) \\
\text{s.t. } & \boldsymbol{q}_{j+1|k} = \boldsymbol{F}(\boldsymbol{q}_{j|k}, \boldsymbol{u}_{j|k}),\ \forall j \in \mathbb{I}_0^{N-1} \\
& \operatorname{sgn}(v_{W_1,j|k}) = \operatorname{sgn}(v_{W_1}(t_0)),\ \forall j \in \mathbb{I}_0^{N} \\
& v_W^- \leq v_{W_{i,j|k}}(\cdot) \leq v_W^+,\ \forall i \in \mathbb{I}_2^{n_s}, \forall j \in \mathbb{I}_0^{N-1} \\
& \operatorname{sgn}(v_{S_{1,j|k}}(\cdot)) = \operatorname{sgn}(v_{S_1}(t_0)),\ \forall j \in \mathbb{I}_0^{N} \\
& \operatorname{sgn}(v_{S_{i,j|k}}(\cdot)) = \operatorname{sgn}(v_{S_i}(t_0)),\ \forall i \in \mathbb{I}_2^{n_s}, \forall j \in \mathbb{I}_0^{N-1} \\
& a_W^- \leq a_{W_1,j|k} \leq a_W^+,\ \forall j \in \mathbb{I}_0^{N-1} \\
& v_\beta^- \leq v_{\beta_1,j|k} \leq v_\beta^+,\ \forall j \in \mathbb{I}_0^{N-1} \\
& v_\beta^- \leq v_{\beta_i,j|k}(\cdot) \leq v_\beta^+,\ \forall i \in \mathbb{I}_2^{n_s}, \forall j \in \mathbb{I}_0^{N-1} \\
& \boldsymbol{q}_{0|k} = \boldsymbol{q}_k,
\end{aligned}$$

with vectors

$$\begin{aligned}
\boldsymbol{Q}_k &= (\boldsymbol{q}_{0|k}^\top, \boldsymbol{q}_{1|k}^\top, \ldots, \boldsymbol{q}_{N|k}^\top)^\top \\
\boldsymbol{U}_k &= (\boldsymbol{u}_{0|k}^\top, \boldsymbol{u}_{1|k}^\top, \ldots, \boldsymbol{u}_{N-1|k}^\top)^\top
\end{aligned}$$

collecting the decision variables of the NMPC at t_k, $T_{\text{MPC}} = N\delta_{\text{MPC}}$, δ_{MPC} timestep of the NMPC, and the cost function evaluated using $\boldsymbol{q}_{j|k}^{\text{aux}}$ $(j = 0, \ldots, N)$ and $\boldsymbol{u}_{j|k}^{\text{aux}}$ $(j = 0, \ldots, N-1)$, computed by the auxiliary trajectory generation scheme. Note that, within the constraints, we used $(\cdot)$ to denote the use of nonlinear functions.

Once the NLP problem is solved, the control sample $\boldsymbol{u}_{0|k}$ is extracted rom $\boldsymbol{U}_k$, and used to compute the driving velocities and the steering angles, which are sent to the robot.

8.3 Experiments

The proposed framework has been implemented in Python, using the acados library [73] to solve the aforementioned NLP problem with real-time iteration scheme [70]. We use the robot Neobotix MPO-700, which has $n_s = 4$ steerable wheels (Fig. 8.1). The scheme runs at 75 Hz on an Intel Core i5-10210U (1.6 GHz, 8 cores) with Ubuntu 20.04 LTS.

Symbol	Value
$\boldsymbol{a}$	$(-0.19, 0.19, 0.19, -0.19)$ [m]
$\boldsymbol{b}$	$(0.24, 0.24, -0.24, -0.24)$ [m]
d	0.045 [m]
$a_{W_1}^{\text{init}}$	0.1 [m/s^2]
$K_{v_{W_1}}^{\text{stop}}$	1.0
$\boldsymbol{K}_P$	$\text{diag}(4.0, 4.0, 2.0)$
$\boldsymbol{K}_D$	$\text{diag}(2.0, 2.0, 1.0)$
N	5
δ_{MPC}	0.1 [s]
$\boldsymbol{W_q}$	$\boldsymbol{I}_9$
$\boldsymbol{W_u}$	$\boldsymbol{I}_3$
v_W^-	-0.9 [m/s]
v_W^+	0.9 [m/s]
a_W^-	-0.5 [m/s^2]
a_W^+	0.5 [m/s^2]
v_β^-	-2.0 [rad/s]
v_β^+	2.0 [rad/s]

Table 8.1. Hyperparameters used in all our experiments.

We validate our implementation on a series of trajectory tracking experiments of increasing complexity. We define the trajectories, using a geometric path $\boldsymbol{\xi}^{\text{ref}}(s)$ and a timing law $s(t) \in [0, 1]$. Naming t_0 and t_f respectively initial and final time, all $s(t)$ are 5-th order polynomials, such that:

$$s(t_0) = \dot{s}(t_0) = \dot{s}(t_f) = \ddot{s}(t_0) = \ddot{s}(t_f) = 0, s(t_f) = 1,$$

In the following, we will use $t_0 = 0$ [s], and we will assume that the initial position of the robot is $x_0 = 0$ [m], $y_0 = 0$ [m].

The hyperparameters used are listed in Table 8.1.

8.3.1 Straight line motions

The *forward motion* trajectory, consists in the robot moving along a straight line, without changing its orientation. It is defined by

$$\begin{aligned} x^{\text{ref}}(s) &= x_0 + s(x_f - x_0) \\ y^{\text{ref}}(s) &= y_0 + s(y_f - y_0) \\ \theta^{\text{ref}}(s) &= \theta_0 \\ x_f &= x_0 + v^{\text{ref}} \cos(\theta^{\text{dir}})(t_f - t_0) \\ y_f &= y_0 + v^{\text{ref}} \sin(\theta^{\text{dir}})(t_f - t_0), \end{aligned}$$

with $\theta_{\text{dir}} = \theta_0$ and $v^{\text{ref}} > 0$. The initial orientation of the robot is $\theta_0 = 0.0$ [rad], and the initial configuration of the steering angles is given by $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = 0.0$ [rad]. Moreover, $t_f = 25.0$ [s] and $v^{\text{ref}} = 0.2$ [m/s]. Figure 8.5 shows a sequence of snapshots of the mobile base moving while tracking the considered

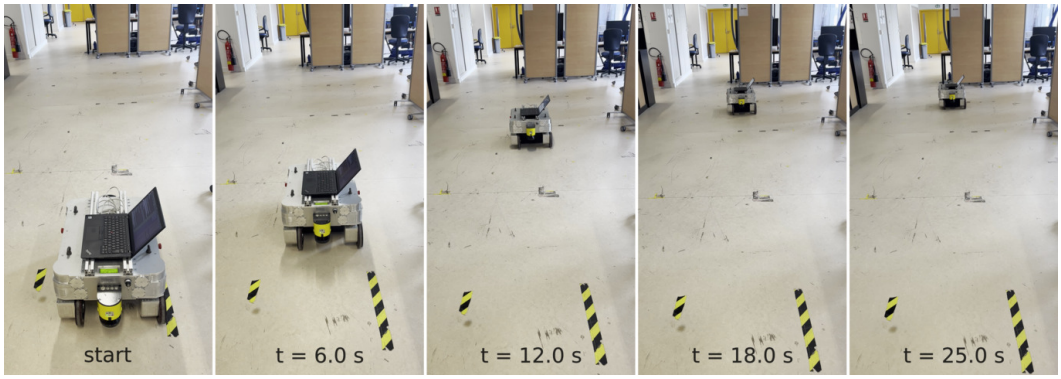

Figure 8.5. Snapshots of the mobile base tracking a *forward motion* trajectory. The robot accelerates for half the duration of the motion, eventually decelerating and stopping.

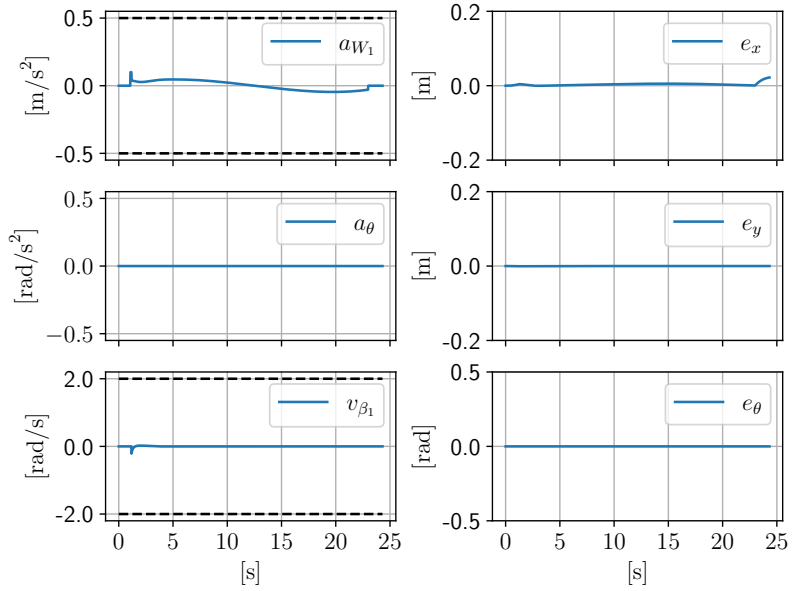

Figure 8.6. Control inputs and trajectory tracking errors for the *forward motion* trajectory.

trajectory. Here, the robot starts moving, it accelerates until completing the first half of the trajectory, it then decelerate, finally stopping its motion. Figure 8.6 show the control inputs computed by the NMPC and the trajectory tracking error. Notice that both driving acceleration a_{W_1} and steering velocity v_{β_1} are within their boundaries. Figure 8.7 shows the corresponding driving and steering velocities. Note that the initial spike in the steering velocity is due to the change of status of the FSM from *Starting* to *Moving*.

The *backward motion* trajectory is defined similarly to the *forward motion* trajectory, with the difference that $v^{\text{ref}} < 0$. The initial orientation of the robot is $\theta_0 = 0.0$ [rad]. Moreover, $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = \pi$ [rad], $t_f = 25.0$ [s] and $v^{\text{ref}} = 0.2$ [m/s]. Figure 8.8shows a sequence of snapshots of the mobile base moving while tracking the considered trajectory. Figure 8.9 show the control inputs computed by the NMPC and the trajectory tracking error. Figure 8.10 shows the

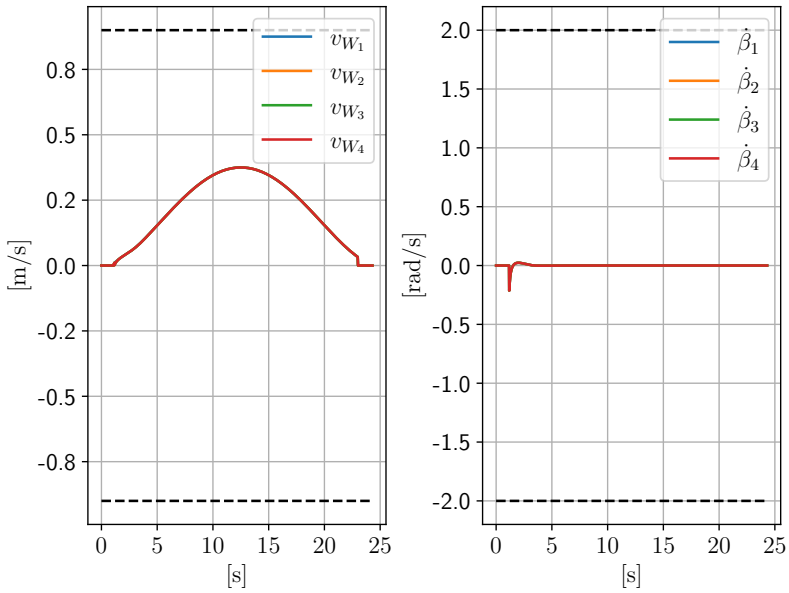

Figure 8.7. Driving and steering velocities of the four wheels for the *forward motion* trajectory. The driving velocities are increasing for the first part of the motion, and decreasing until the robots stops for the second half. The steering velocities, apart for the first spike (due to numerical approximations), are always close to zero, indicating that the robot is not steering.

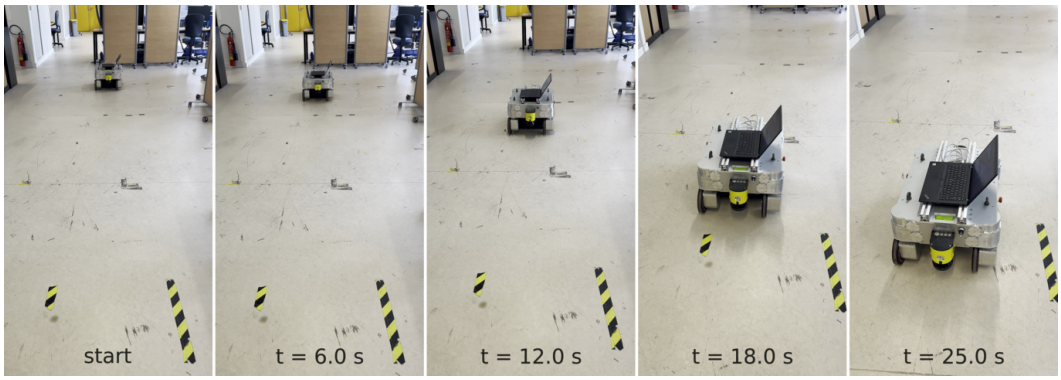

Figure 8.8. Snapshots of the mobile base tracking a *backward motion* trajectory.

corresponding driving and steering velocities. Note again, how constraints are always satisfied during the motion of the robot. From the snapshots, it is possible to see that the robot is slightly deviating from the straight line. This behavior is due the odometric localization module, which is not as precise as a Kalman filter or SLAM [92]. The integration of a better localization system will be part of future works.

The *diagonal motion* trajectory is defined similarly to the *forward motion* trajectory, but with $\theta^{\text{dir}} \neq \theta_0$. The initial orientation of the robot is $\theta_0 = \pi/4$ [rad]. Moreover, $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = -\pi/4$ [rad], $t_f = 25.0$ [s] and $v^{\text{ref}} = 0.2$ [m/s]. Figure 8.11 shows a sequence of snapshots of the mobile base moving while tracking the considered trajectory. Figure 8.12 show the control inputs computed by the NMPC and the trajectory tracking error. Figure 8.13 shows the corresponding

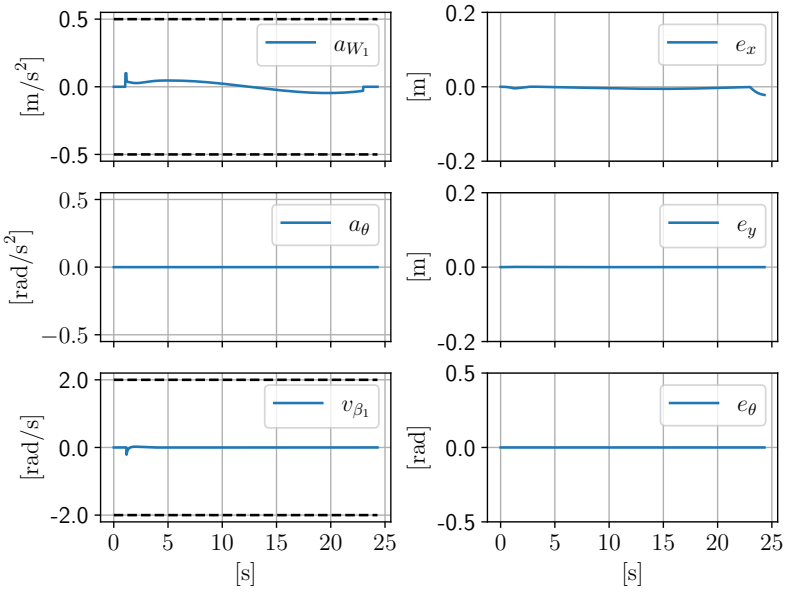

Figure 8.9. Control inputs and trajectory tracking errors for the *backward motion* trajectory.

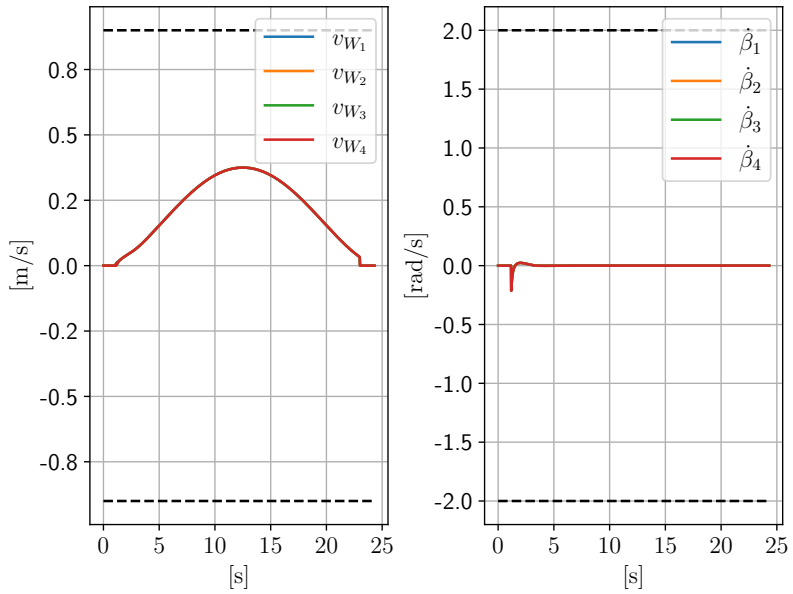

Figure 8.10. Driving and steering velocities of the four wheels for the *backward motion* trajectory.

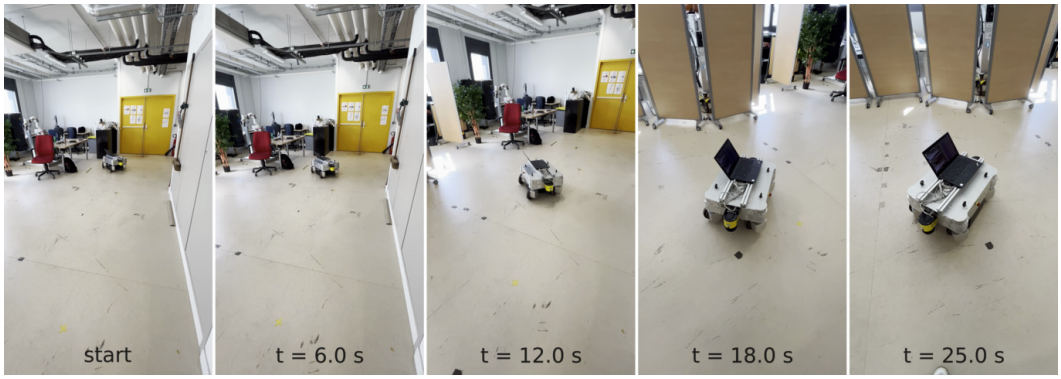

Figure 8.11. Snapshots of the mobile base tracking a *diagonal motion* trajectory.

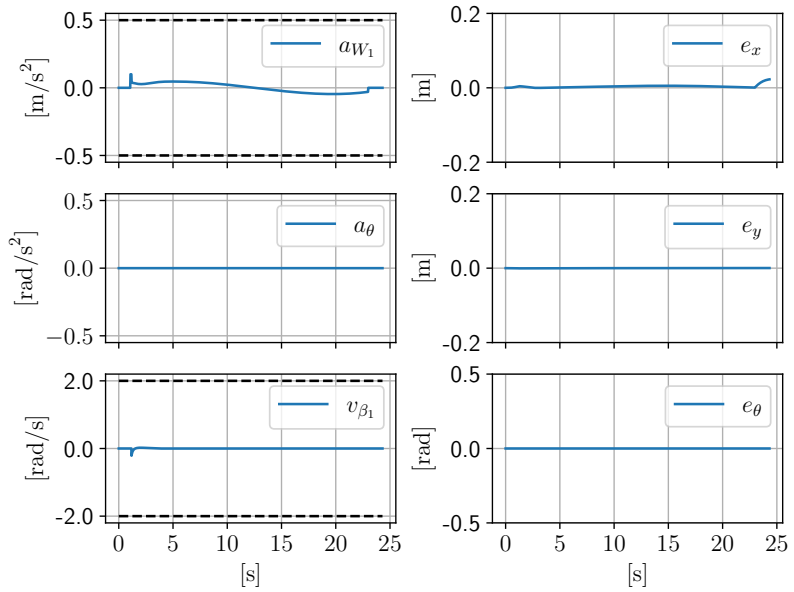

Figure 8.12. Control inputs and trajectory tracking errors for the *diagonal motion* trajectory.

driving and steering velocities. Once again, the trajectory is tracking while keeping the error close to zero, and satisfying all constraints of the system.

In all these experiments (*forward*, *backward* and *diagonal motion*), the reference position of the robot is the same. The reference orientation, on the other hand, is different. Note that the robot is able to track diagonal trajectories because of the steerable wheels, which allow the mobile base behave as an omnidirectional robot.

8.3.2 Circular motions

In this section, we consider tasks in which the robot is required to follow a circle with center (x_C, y_C) and radius $R > 0$. The reference position, which is in common

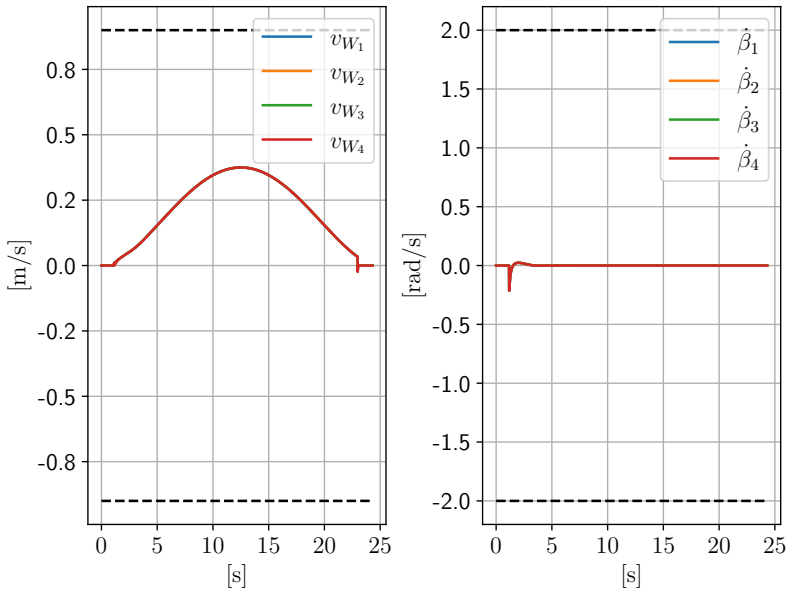

Figure 8.13. Driving and steering velocities of the four wheels for the *diagonal motion* trajectory. The profile is similar to the case of moving forward because the initial configuration of the steerable joints are displaced by π [rad] with respect to the previous experiment.

among all circular trajectories (defined below), is given by

$$x^{\text{ref}}(s) = x_C + R\cos(\phi + 2\pi s)$$
$$y^{\text{ref}}(s) = y_C + R\sin(\phi + 2\pi s).$$

In the *circle with constant orientation* trajectory, the reference orientation is defined by

$$\theta^{\text{ref}}(t) = \theta_0,$$

with $\theta_0 = \pi$ [rad]. The initial configuration of the steering angles are given by$\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = 0.0$ [rad]. Moreover, $R = 0.5$ [m], $\phi = \pi/2$ [rad], and $t_f = 15.7$ [s].

Similarly to the *diagonal motion*, the robot is able to track a circle while keeping its orientation constant because of the steerable wheels. This kind of motion, indeed, would not be possible with a differential drive robot. Figure 8.14 shows a sequence of snapshots of the mobile base moving while tracking the considered trajectory. Figure 8.15 shows the control inputs computed by the NMPC and the trajectory tracking error. Figure 8.16 shows the corresponding driving and steering velocities.

In the *circle with tangent orientation* trajectory, the reference orientation is defined by

$$\theta^{\text{ref}}(s) = \text{atan2}\left(\frac{\partial y^{\text{ref}}(s)}{\partial s}, \frac{\partial x^{\text{ref}}(s)}{\partial s}\right). \tag{8.14}$$

with $\theta_0 = \pi$ [rad], $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = 0.0$ [rad]. Moreover, $R = 0.5$ [m], $\phi = \pi/2$ [rad], and $t_f = 15.7$ [s]. Figure 8.17 shows a sequence of snapshots of the

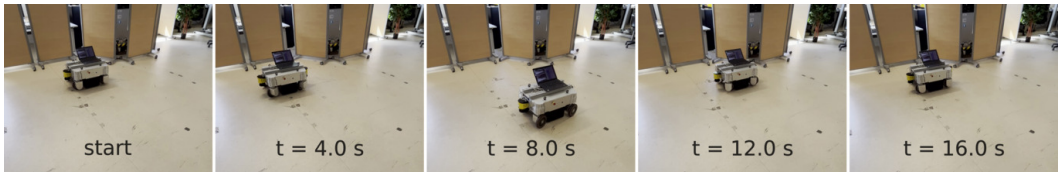

Figure 8.14. Snapshots of the mobile base tracking a *circle with constant orientation*. Notice how the robot keeps the same orientation throughout the motion. This is possible because of the omnidirectional capabilities of the platform.

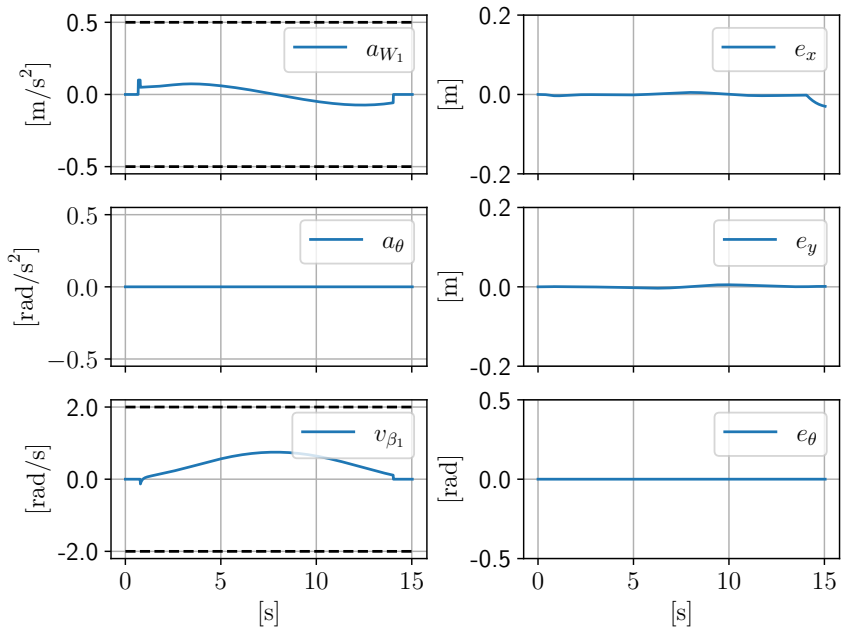

Figure 8.15. Control inputs and trajectory tracking errors for the *circle with constant orientation* trajectory.

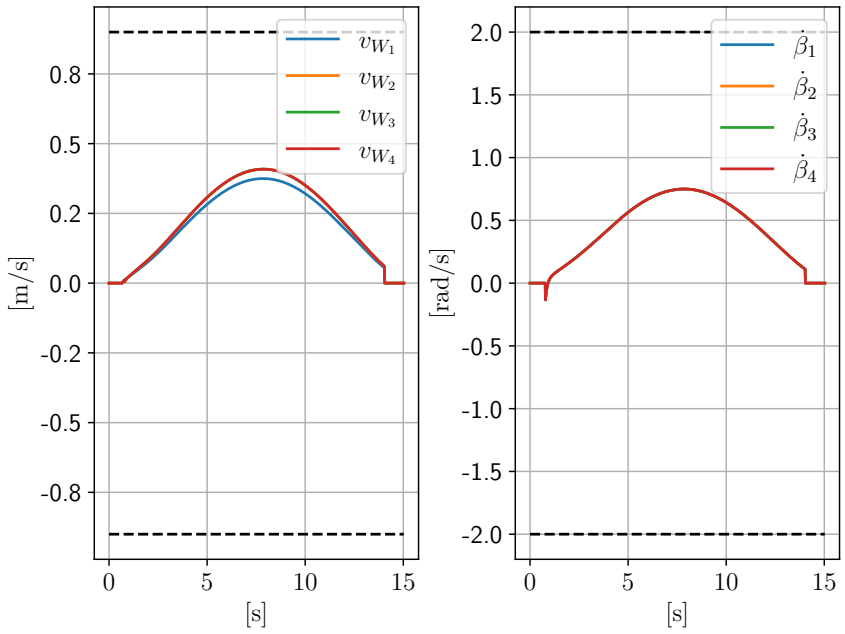

Figure 8.16. Driving and steering velocities of the four wheels for the *circle with constant orientation* trajectory.

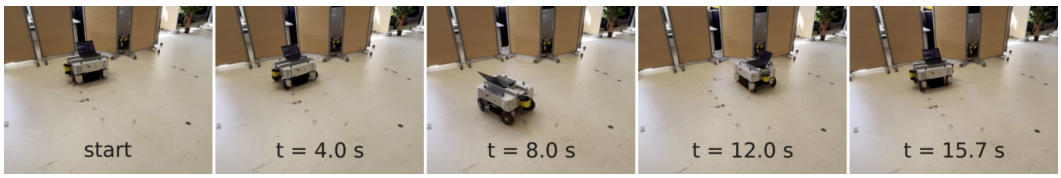

Figure 8.17. Snapshots of the mobile robot tracking a *circle with tangent orientation*.

robot moving while tracking this trajectory. Here, it is possible to notice how the mobile base changes its orientation according to the previously defined circle, ending up in its initial pose at the end of the motion. Figure 8.18 show the control inputs computed by the NMPC, together with the trajectory tracking error. Figure 8.19 shows the corresponding driving and steering velocities. The trajectory tracking error is always close to zero, and the control inputs, together with steering and driving velocities of the wheels, are always within their boundaries.

In the *circle with inward orientation* trajectory, the robot is required to follow the previously defined circle, while pointing its front towards the center of the circle itself. The reference orientation is defined as

$$\theta^{\text{ref}}(s) = \text{atan2}(y_C - y^{\text{ref}}(s), x_C - x^{\text{ref}}(s)).$$

Here, $\theta_0 = -\pi/2$ [rad], $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = -\pi/2$ [rad]. Moreover, $R = 0.6$ [m], $\phi = \pi/2$ [rad], and $t_f = 18.8$ [s]. Figure 8.20 shows a sequence of snapshots of the mobile base moving while tracking the reference trajectory. In this experiment, it is possible to notice that the position of the mobile base is identical to the one of the previous section. The reference orientation, on the other hand, is such that the front of the robot points towards the center of the circle. Figure 8.21 shows the

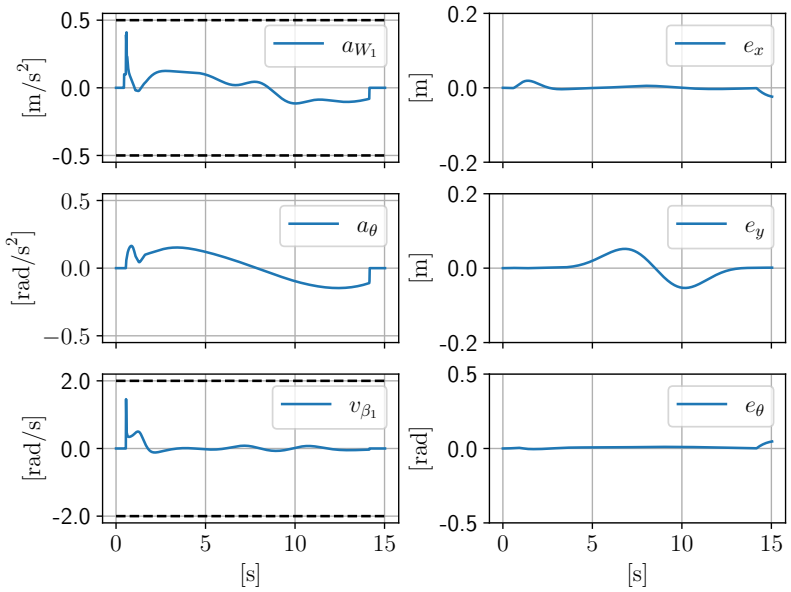

Figure 8.18. Control inputs and trajectory tracking errors for the *circle with tangent orientation* trajectory.

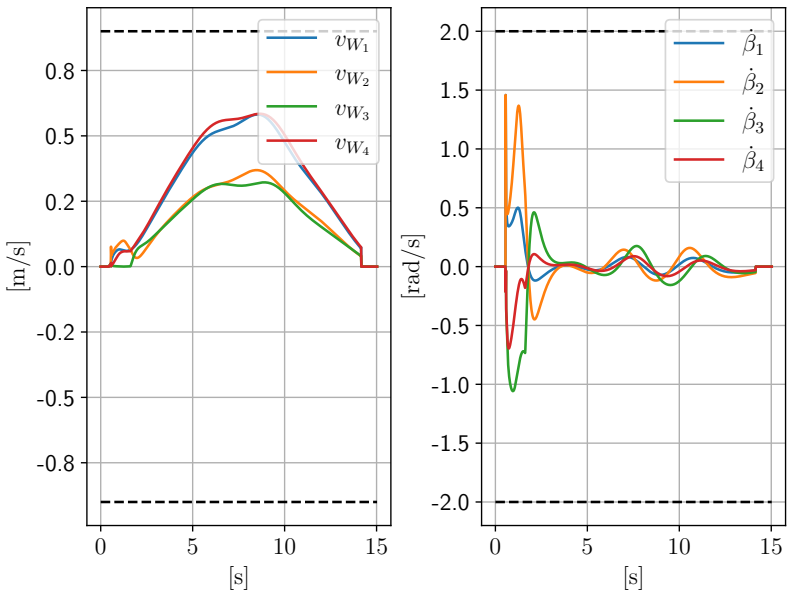

Figure 8.19. Driving and steering velocities of the four wheels for the *circle with tangent orientation* trajectory.

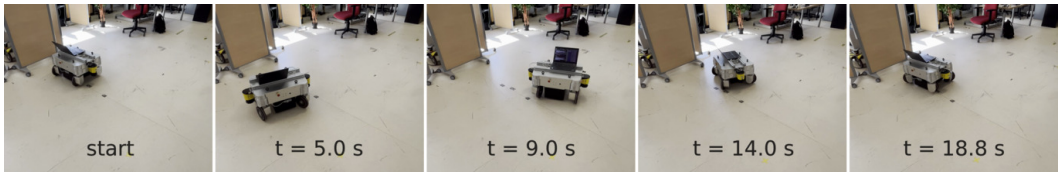

Figure 8.20. Snapshots of the mobile base tracking a *circle with inward orientation*.

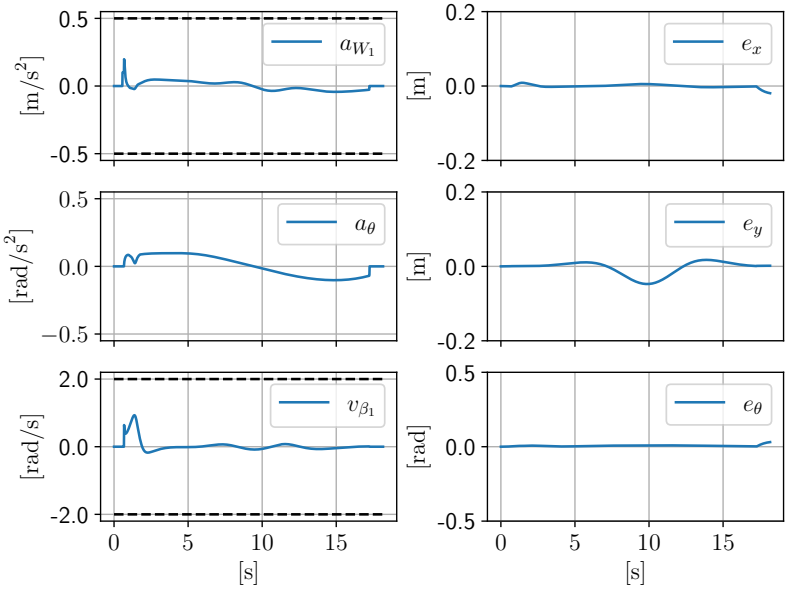

Figure 8.21. Control inputs and trajectory tracking errors for the *circle with inward orientation* trajectory.

control inputs computed by the NMPC and the trajectory tracking error. Figure 8.22 shows the corresponding driving and steering velocities.

In the *circle with outward orientation* trajectory, the robot is required to follow the previously defined circle, while pointing its back towards the center of the circle itself. The reference orientation is defined as

$$\theta^{\text{ref}}(s) = \text{atan2}(y^{\text{ref}}(s) - y_C, x^{\text{ref}}(s) - x_C).$$

Here, $\theta_0 = \pi/2$ [rad], $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = \pi/2$ [rad]. Moreover, $R = 0.6$ [m], $\phi = \pi/2$ [rad], and $t_f = 18.8$ [s]. Figure 8.23 shows a sequence of snapshots of the mobile base moving while tracking the considered trajectory. Figure 8.24 show the control inputs computed by the NMPC and the trajectory tracking error. Figure 8.25 shows the corresponding driving and steering velocities. In this experiment, the behavior is similar to the *circle with inward orientation*, as the only difference in the reference trajectory is the orientation of the base.

8.3.3 Slalom motions

In this section, we consider tasks in which the robot is required to follow a sinusoidal trajectory. The reference position, which is in common among all trajectories defined

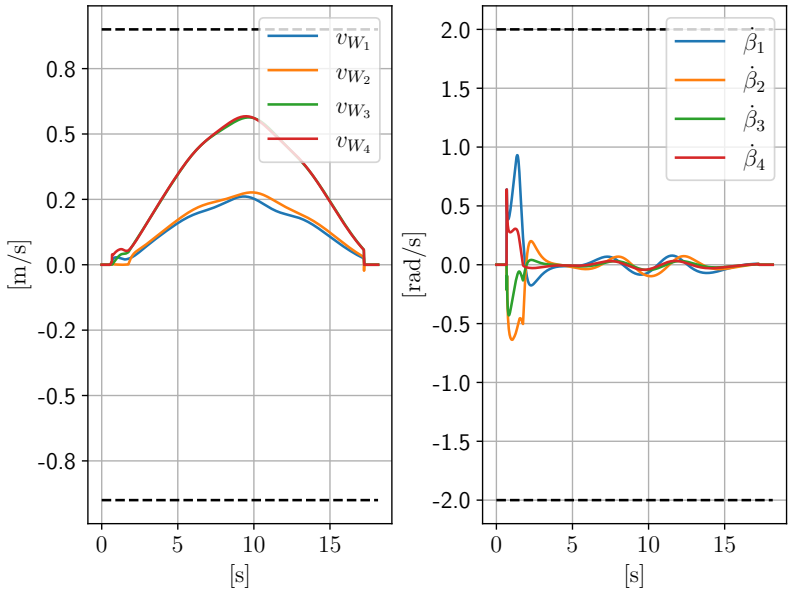

Figure 8.22. Driving and steering velocities of the four wheels for the *circle with inward orientation* trajectory.

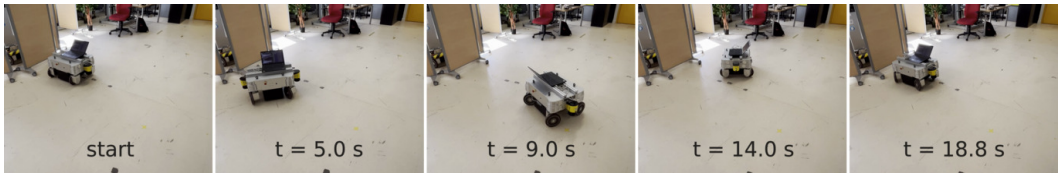

Figure 8.23. Snapshots of the mobile base tracking a *circle with outward orientation*.

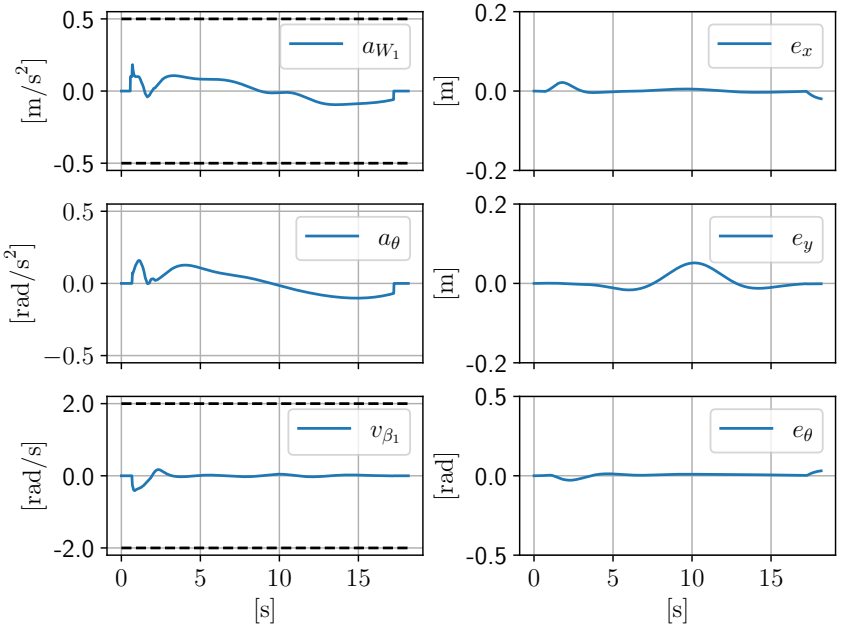

Figure 8.24. Control inputs and trajectory tracking errors for the *circle with outward orientation* trajectory.

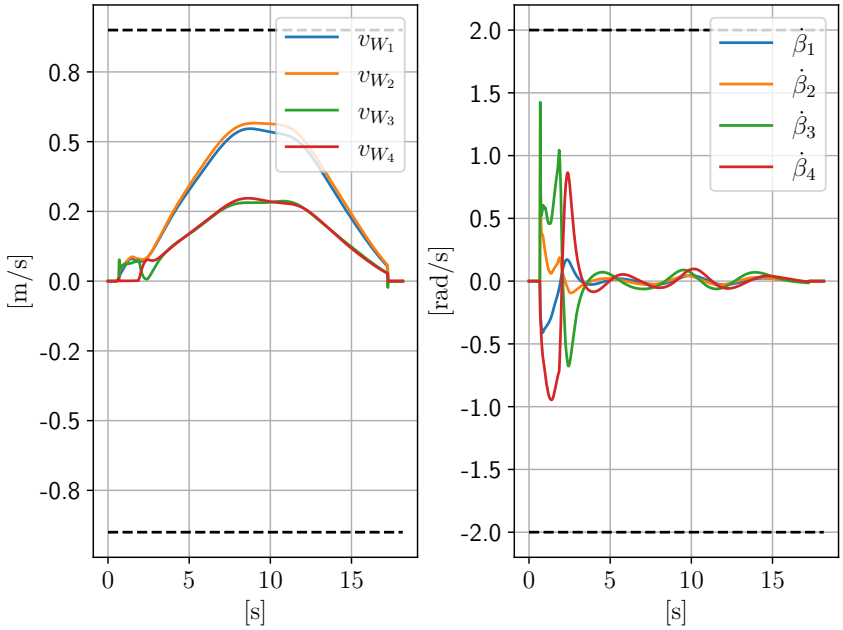

Figure 8.25. Driving and steering velocities of the four wheels for the *circle with outward orientation* trajectory.

below, is given by

$$x^{\text{ref}}(s) = 2\pi s$$
$$y^{\text{ref}}(s) = \alpha \sin(2\pi s).$$

In the *slalom with constant orientation* trajectory, the robot is required to follow the above defined sinusoidal trajectory while keeping its orientation constant. The reference orientation is given by

$$\theta^{\text{ref}}(t) = \theta_0.$$

Here, the initial orientation of the robot is $\theta_0 = 0.0$ [rad]. Moreover, $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = 0.0$ [rad], $\alpha = 0.8$, and $t_f = 31.4$ [s].

Note that, similarly to the *circle with tangent orientation*, the tracking of this kind of trajectory is only possible because of the steerable wheels. Figure 8.26 shows a sequence of snapshots of the mobile base moving while tracking the considered trajectory. Figure 8.27 show the control inputs computed by the NMPC and the trajectory tracking error. Figure 8.28 shows the corresponding driving and steering velocities. Note that the spike in the steering velocities computed by the NMPC is due to the initial configuration of the wheels, which are not aligned with respect to the reference trajectory the robot is required to track. In this case, the auxiliary trajectory generation scheme generates an auxiliary trajectory of the steering velocities which do not satisfy the constraints of the platform. The NMPC is able to take this into account, generating a control action which satisfy all requirements. The resulting behavior is a realignment of the steerable wheels at the beginning of the motion, which makes it possible to successfully complete the assigned task.

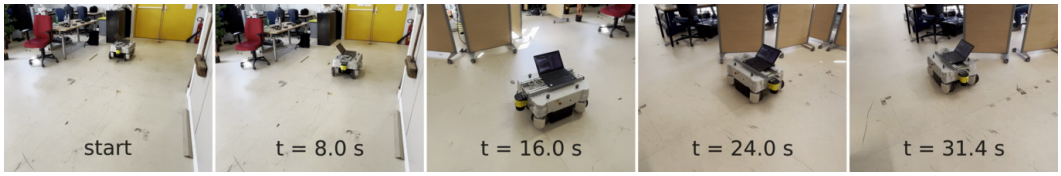

Figure 8.26. Snapshots of the mobile base tracking a *slalom with constant orientation*. Notice how the robot keeps the same orientation throughout the motion. Again, this behavior is possible because of the omnidirectional capabilities of the platform.

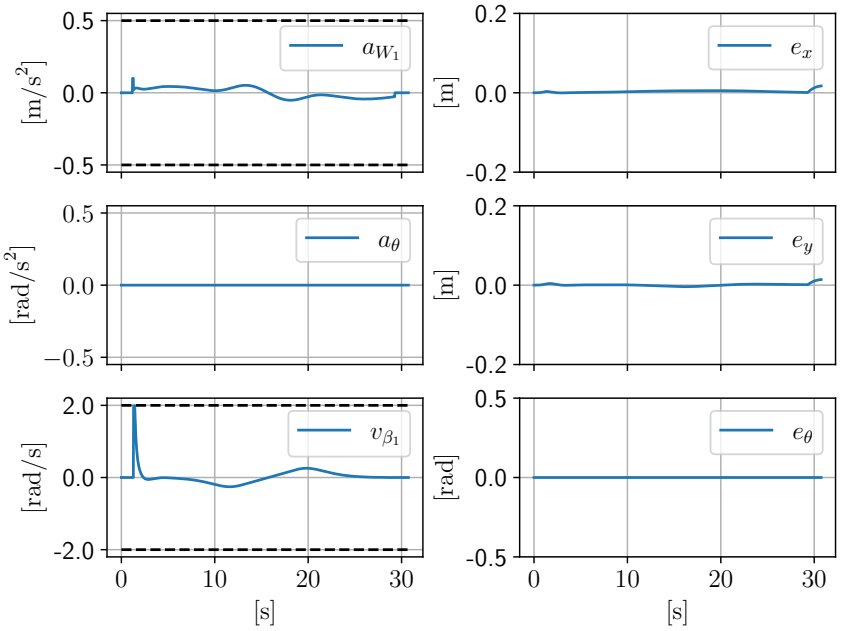

Figure 8.27. Control inputs and trajectory tracking errors for the *slalom with constant orientation* trajectory. Notice how the steering velocity v_{β_1} is at the upper bound when the FSM changes its state from *Starting* to *Moving*. This is because of the auxiliary trajectory generation scheme, which generates too high velocities, because of the steerable wheels which are not aligned with the reference trajectory at the beginning of the motion. The NMPC is capable of taking this into account, generating feasible control inputs.

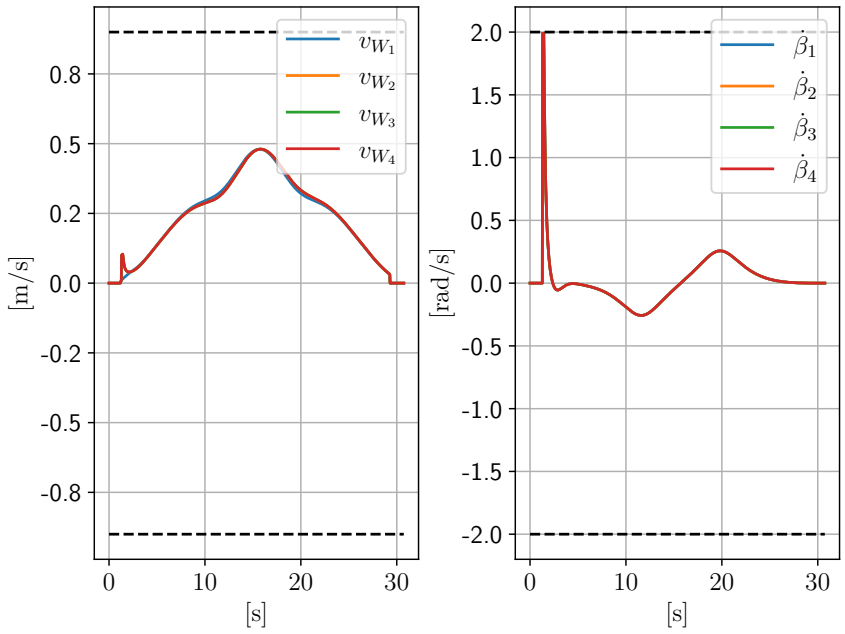

Figure 8.28. Driving and steering velocities of the four wheels for the *slalom with constant orientation* trajectory. The steering velocities at the limit of the bound at the beginning of the motion reflect the behavior of the control inputs computed by the NMPC.

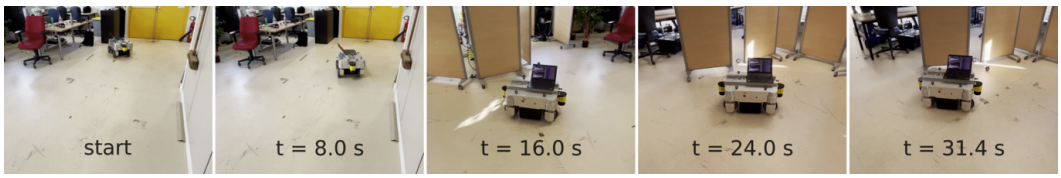

Figure 8.29. Snapshots of the mobile base tracking a *slalom with tangent orientation*.

In the *slalom with tangent orientation* trajectory, the robot is required to follow again the sinusoidal trajectory defined above, but while keeping its orientation tangent to the trajectory itself. The reference orientation is given by eq. (8.14). Here, the initial orientation of the robot is $\theta_0 = \text{atan}(\alpha)$ [rad]. Moreover, $\beta_{1,0} = \beta_{2,0} = \beta_{3,0} = \beta_{4,0} = 0.0$ [rad], with $\alpha = 0.8$, and $t_f = 31.4$ [s].

Figure 8.29 shows a sequence of snapshots of the mobile base moving while tracking the considered trajectory. Here, the reference position is the same as the one defined in the previous section, while the reference orientation in different. In the snapshots, it is possible to notice how the robot tracks the slalom while keeping its orientation tangent to the slalom itself. Figure 8.30 shows the control inputs computed by the NMPC and the trajectory tracking error, which is always close to zero. Figure 8.31 shows the corresponding driving and steering velocities. Once again, the NMPC generates feasible control inputs while satisfying driving and steering velocity constraints on all wheels.

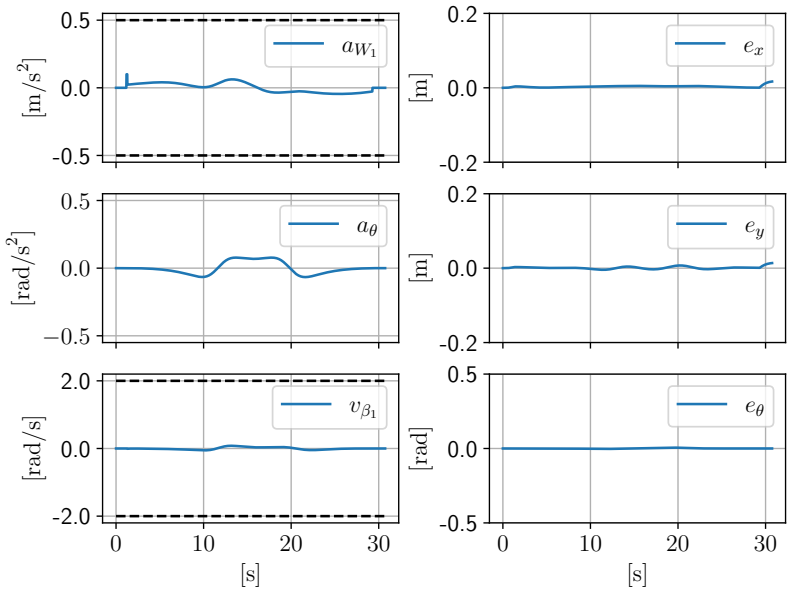

Figure 8.30. Control inputs and trajectory tracking errors for the *slalom with tangent orientation* trajectory.

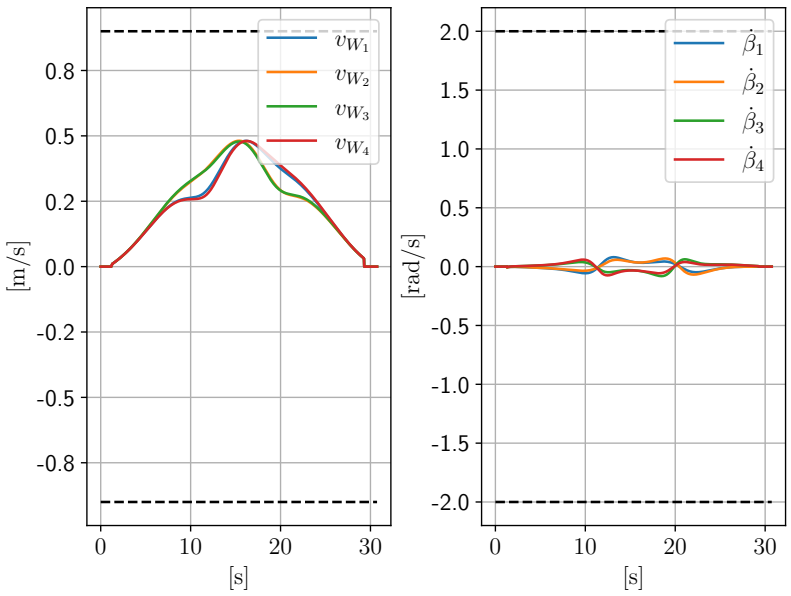

Figure 8.31. Driving and steering velocities of the four wheels for the *slalom with tangent orientation* trajectory.

Chapter 9

Conclusions

This thesis addressed the problem of generation and control of motion for humanoids and steerable wheeled mobile robots. In particular, in the first part of the thesis we have studied the problem of motion generation for humanoids in a *world of stairs*, a particular uneven terrain where the contact surfaces are piecewise-horizontal. After introducing their dynamics in Chapter 3, and Intrinsically-Stable Model Predictive Control (IS-MPC) in Chapter 4, we have presented a sensor-based framework for locomotion which makes use of a footstep planner based on RRT* in Chapter 5, and a feasibility-aware plan adaptation module based on mixed-integer nonlinear optimization in Chapter 6. In the second part of the thesis, after a brief overview of Nonlinear Model Predictive Control (NMPC) based on the real-time iteration (RTI) scheme in Chapter 7, we have introduced a framework based on RTI for the control of the motion of steerable wheeled mobile robots (SWMRs) in Chapter 8. In the following, we summarize the main scientific contributions of this manuscript, and discuss possible future works.

In Chapter 5, we addressed the problem of motion generation for a humanoid robot that must reach a certain goal region walking in a *world of stairs*. We considered two versions of such problem: the off-line and on-line case. In the first, the geometry of the environment is completely known in advance, while in the second, it is reconstructed by the robot itself during motion using an on-board sensor. In both cases, available information about the environment is maintained in the form of an elevation map.

For the off-line case, we proposed an architecture working in two main stages: footstep planning and gait generation. First, a feasible footstep plan leading to the goal region is off-line computed using a randomized algorithm that takes into account the plan quality specified by a given optimality criterion. Then, an intrinsically stable MPC-based scheme computes a CoM trajectory that realizes the found footstep sequence, while guaranteeing dynamic balance and boundedness of the CoM with respect to the ZMP at all time instants.

For the on-line case, we proposed an extension of the architecture for the off-line case where footstep plans are computed in parallel to gait generation and map building. To this end, we presented a sensor-based version of the footstep planner that uses the knowledge about the environment incrementally acquired by the robot during motion.

We validated the proposed architectures by providing simulation results obtained in CoppeliaSim on the HRP-4 humanoid robot in scenarios of different complexity.

Our future work will explore several directions, such as

1. providing a formal proof of asymptotic optimality of the proposed footstep planner;
2. developing a more general version of the proposed approach to deal with arbitrary terrains, removing the world of stairs assumption;
3. implementing the presented architectures on a real humanoid robot;
4. extending them to the case of large-scale and multi-floor environments.

In Chapter 6, we presented Feasibility-Aware Plan Adaptation (FAPA), a module for adapting footstep plans (positions, orientations and timings) in such a way to enhance the IS-MPC scheme of Chapter 4. To do so, we exploited the feasibility region of IS-MPC (Sect. 4.6), using a gait feasibility constraint. We considered two versions of the scheme: Fixed patches FAPA (F-FAPA), where the regions assignment for placing the footsteps is fixed, and Variable patches FAPA (V-FAPA), where the regions can be selected automatically. In F-FAPA, the optimization problem is formulated as a Nonlinear Programming Problem (NLP), while in V-FAPA it is formulated as a mixed-integer NLP. We validated FAPA in MATLAB simulations, showing that the plan is adapted in a very flexible way in reaction to strong pushes. In our MATLAB prototype, the performance is fully compatible with real time in the case of F-FAPA, while not yet in the case of V-FAPA.

Future works will be aimed at

1. reimplementing FAPA in C++ in order to meet real-time requirements;
2. introducing the on-line footstep planner of Chapter 5 inside the architecture so that global replanning is possile;
3. explore convex relaxation [93] and use a MIQP solver such as Gurobi [94] to further speed up the computation;
4. deploy FAPA on a real humanoid robot.

In Chapter 8, we presented a framework for trajectory tracking with steerable wheeled mobile robots, which makes use of a Nonlinear MPC based on real-time iteration. Our scheme is capable of tracking trajectories without violating wheels' velocity constraints, while taking into account kinematic model singularities. We have validated our approach on multiple trajectories using the Neobotix MPO-700, showing that our scheme is always able to track them. To the best of our knowledge, this is the first time a NMPC has been implemented on a SWMR.

In our future works, we plan to extend the framework in several ways:

1. extend the NMPC to a dual-arm mobile manipulators such as BAZAR robot [95], making it interact with the environments with the arms while moving;

2. implement a motion planning algorithm such as kinodynamic RRT* [96], making the robot able to navigate autonomously in an environment with obstacles;

3. further improve the performance of the framework by implementing it in C++ (while the scheme runs in real-time thanks to acados which compiles the NMPC, most of the time is taken by the auxiliary trajectory generation module, which completely relies on Python).

Bibliography

[1] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE Int. Conf. on Robotics and Automation*, pages 1620–1626, 2003.

[2] Tomomichi Sugihara, Kenta Imanishi, Takanobu Yamamoto, and Stéphane Caron. 3D biped locomotion control including seamless transition between walking and running via 3D ZMP manipulation. In *2021 IEEE Int. Conf. on Robotics and Automation*, pages 6258–6263, 2021.

[3] Stéphane Caron, Adrien Escande, Leonardo Lanari, and Bastien Mallein. Capturability-based pattern generation for walking with variable height. *IEEE Transactions on Robotics*, 36(2):517–536, 2019.

[4] Alessio Zamparelli, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. Humanoid gait generation on uneven ground using intrinsically stable MPC. *IFAC-PapersOnLine*, 51:393–398, 2018.

[5] Paolo Ferrari, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. An integrated motion planner/controller for humanoid robots on uneven ground. In *18th European Control Conference*, pages 1598–1603, 2019.

[6] A. Ibanez, P. Bidaud, and V. Padois. Emergence of humanoid walking behaviors from Mixed-integer Model Predictive Control. In *2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 4014–4021, 2014.

[7] Young-Dae Hong, Ye-Hoon Kim, Ji-Hyeong Han, Jeong-Ki Yoo, and Jong-Hwan Kim. Evolutionary Multiobjective Footstep Planning for Humanoid Robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(4):520–532, 2011.

[8] Mohammadreza Kasaei, Ali Ahmadi, Nuno Lau, and Artur Pereira. A modular framework to generate robust biped locomotion: from planning to control. *SN Applied Sciences*, 3:1–18, 2021.

[9] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 279–286, 2014.

[10] Daeun Song, Pierre Fernbach, Thomas Flayols, Andrea Del Prete, Nicolas Mansard, Steve Tonneau, and Young J. Kim. Solving Footstep Planning as a Feasibility Problem Using L1-Norm Minimization. *IEEE Robotics and Automation Letters*, 6:5961–5968, 2021.

[11] Robert J. Griffin, Georg Wiedebach, Stephen McCrory, Sylvain Bertrand, Inho Lee, and Jerry Pratt. Footstep Planning for Autonomous Walking Over Rough Terrain. In *2019 IEEE Int. Conf. on Robotics and Automation*, pages 9–16, 2019.

[12] Hong Liu, Qing Sun, and Tianwei Zhang. Hierarchical RRT for Humanoid Robot Footstep Planning with Multiple Constraints in Complex Environments. In *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 3187–3194, 2012.

[13] Kei Okada, Takashi Ogura, Atsushi Haneda, and Masayuki Inaba. Autonomous 3D walking system for a humanoid robot based on visual step recognition and 3D foot step planner. In *2005 IEEE Int. Conf. on Robotics and Automation*, pages 623–628, 2005.

[14] Joel Chestnutt, Yutaka Takaoka, Keisuke Suga, Koichi Nishiwaki, James Kuffner, and Satoshi Kagami. Biped navigation in rough environments using on-board sensing. In *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 3543–3548, 2009.

[15] Maurice F. Fallon, Pat Marion, Robin Deits, and Thomas Whelan. Continuous humanoid locomotion over uneven terrain using stereo fusion. In *2015 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 881–888, 2015.

[16] Daniel Maier, Christian Lutz, and Maren Bennewitz. Integrated perception, mapping, and footstep planning for humanoid navigation among 3d obstacles. In *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2658–2664, 2013.

[17] Alexander Stumpf, Stefan Kohlbrecher, David C Conner, and Oskar von Stryk. Supervised footstep planning for humanoid robots in rough terrain tasks using a black box walking controller. In *2014 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 287–294, 2014.

[18] Philipp Karkowski, Stefan Oßwald, and Maren Bennewitz. Real-time footstep planning in 3D environments. In *2016 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 69–74, 2016.

[19] Takanobu Yamamoto and Tomomichi Sugihara. Responsive navigation of a biped robot that takes into account terrain, foot-reachability and capturability. *Advanced Robotics*, 35(8):516–530, 2021.

[20] A. Herdt, N. Perrin, and P. B. Wieber. Walking without thinking about it. In *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 190–195, 2010.

[21] Marcos R. O. A. Maximo and Rubens J. M. Afonso. Mixed-integer quadratic programming for automatic walking footstep placement, duration, and rotation. *Optimal Control Applications and Methods*, 41(6):1928–1963, 2020.

[22] Néstor Bohórquez and Pierre-Brice Wieber. Adaptive step duration in biped walking: A robust approach to nonlinear constraints. In *17th IEEE-RAS Int. Conf. on Humanoid Robots*, pages 724–729, 2017.

[23] Stéphane Caron and Quang-Cuong Pham. When to make a step? tackling the timing problem in multi-contact locomotion by topp-mpc. In *17th IEEE-RAS Int. Conf. on Humanoid Robots*, pages 522–528, 2017.

[24] Aurelien Ibanez, Philippe Bidaud, and Vincent Padois. Emergence of humanoid walking behaviors from mixed-integer model predictive control. In *2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 4014–4021, 2014.

[25] Filippo M. Smaldone, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. Feasibility-driven step timing adaptation for robust MPC-based gait generation in humanoids. *IEEE Robotics and Automation Letters*, 6(2):1582–1589, 2021.

[26] Majid Khadiv, Alexander Herzog, S. Ali. A. Moosavian, and Ludovic Righetti. Walking control based on step timing adaptation. *IEEE IEEE Trans. on Robotics*, 36(3):629–643, 2020.

[27] M. Naveau, M. Kudruss, O. Stasse, C. Kirches, K. Mombaur, and P. Souères. A reactive walking pattern generator based on nonlinear model predictive control. *IEEE Robotics and Automation Letters*, 2(1):10–17, 2017.

[28] Néstor Bohórquez and Pierre-Brice Wieber. Adaptive step rotation in biped walking. In *2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 720–725, 2018.

[29] Bernardo Aceituno-Cabezas, Carlos Mastalli, Hongkai Dai, Michele Focchi, Andreea Radulescu, Darwin G. Caldwell, José Cappelletto, Juan C. Grieco, Gerardo Fernández-López, and Claudio Semini. Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538, 2018.

[30] Paolo Robuffo Giordano, Matthias Fuchs, Alin O. Albu-Schäffer, and Gerd Hirzinger. On the Kinematic Modeling and Control of a Mobile Platform Equipped with Steering Wheels and Movable Legs. *2009 IEEE International Conference on Robotics and Automation*, 2009.

[31] Mohamed Sorour, Andrea Cherubini, Philippe Fraisse, and Robin Passama. Motion Discontinuity-Robust Controller for Steerable Mobile Robots. *IEEE Robotics and Automation Letters*, 2(2):452–459, 2017.

[32] G. Campion, G. Bastin, and B. Dandrea-Novel. Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. *IEEE Transactions on Robotics and Automation*, 12(1):47–62, 1996.

[33] Mohamed Sorour, Andrea Cherubini, Robin Passama, and Philippe Fraisse. Kinematic modeling and singularity treatment of steerable wheeled mobile robots with joint acceleration limits. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, page 2110–2115. IEEE Press, 2016.

[34] Christian P. Connette, Andreas Pott, Martin Hagele, and Alexander Verl. Control of an pseudo-omnidirectional, non-holonomic, mobile robot based on an icm representation in spherical coordinates. In *2008 47th IEEE Conference on Decision and Control*, pages 4976–4983, 2008.

[35] Mohamed Sorour, Andrea Cherubini, Abdellah Khelloufi, Robin Passama, and Philippe Fraisse. Complementary-route based ICR control for steerable wheeled mobile robots. *Robotics and Autonomous Systems*, 2019.

[36] François Ferland, Lionel Clavien, Julien Frémy, Dominic Létourneau, François Michaud, and Michel Lauria. Teleoperation of azimut-3, an omnidirectional non-holonomic platform with steerable wheels. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2515–2516, 2010.

[37] Lionel Clavien, Michel Lauria, and François Michaud. Estimation of the instantaneous centre of rotation with nonholonomic omnidirectional mobile robots. *Robotics and Autonomous Systems*, 106:47–57, 2018.

[38] Lionel Clavien, Michel Lauria, and François Michaud. Instantaneous Centre of Rotation Based Motion Control for Omnidirectional Mobile Robots with Sidewards Off-Centred Wheels. *Robot. Auton. Syst.*, 106(C):58–68, aug 2018.

[39] Tomomichi Sugihara and Mitsuharu Morisawa. A survey: dynamics of humanoid robots. *Advanced Robotics*, 34:1338 – 1352, 2020.

[40] D.J. Balkcom, J.C. Trinkle, and E.J. Gottlieb. Computing wrench cones for planar contact tasks. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 869–875 vol.1, 2002.

[41] David E. Orin, Ambarish Goswami, and Sung-Hee Lee. Centroidal dynamics of a humanoid robot. *Auton. Robots*, 35(2–3):161–176, oct 2013.

[42] P. Sardain and G. Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(5):630–637, 2004.

[43] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer-Verlag, 2007.

[44] Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. Zmp support areas for multi-contact mobility under frictional constraints. *IEEE Transactions on Robotics*, 33(1):67–80, February 2017.

[45] Twan Koolen, Michael Posa, and Russ Tedrake. Balance control using center of mass height variation: Limitations imposed by unilateral contact. *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 8–15, 2016.

[46] Stéphane Caron. Biped stabilization by linear feedback of the variable-height inverted pendulum model. In *IEEE International Conference on Robotics and Automation*, May 2020.

[47] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to Humanoid Robotics*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[48] Michele Cipriano, Paolo Ferrari, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. Humanoid motion generation in a world of stairs. *Robotics and Autonomous Systems*, 168:104495, 2023.

[49] Nicola Scianca, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo. MPC for humanoid gait generation: Stability and feasibility. *IEEE Transactions on Robotics*, 36(4):1171–1178, 2020.

[50] J. Englsberger, C. Ott, and A. Albu-Schäffer. Three-dimensional bipedal walking control based on divergent component of motion. *IEEE Transactions on Robotics*, 31(2):355–368, 2015.

[51] J. Pratt, J. Carff, S. Drakunov, and A. Goswami. Capture point: A step toward humanoid push recovery. In *6th IEEE-RAS Int. Conf. on Humanoid Robots*, pages 200–207, 2006.

[52] Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics. In *Robotics: Science and System*, July 2015.

[53] T. Sugihara, Y. Nakamura, and H. Inoue. Real-time humanoid motion generation through ZMP manipulation based on inverted pendulum control. In *2002 IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1404–1409, 2002.

[54] Filippo M Smaldone, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. From walking to running: 3D humanoid gait generation via MPC. *Frontiers in Robotics and AI*, 9, 2022.

[55] S. Caron and A. Kheddar. Dynamic walking over rough terrains by nonlinear predictive control of the floating-base inverted pendulum. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 5017–5024, 2017.

[56] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *Int. J. of Robotics Research*, 33(7):1006–1028, 2014.

[57] Nicola Scianca, Marco Cognetti, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo. Intrinsically stable mpc for humanoid gait generation. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 601–606, 2016.

[58] Leonardo Lanari and Seth Hutchinson. Inversion-based gait generation for humanoid robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1592–1598, 2015.

[59] Ahmed Aboudonia, Nicola Scianca, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo. Humanoid gait generation for walk-to locomotion using single-stage mpc. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 178–183, 2017.

[60] Wolfram Burgard, Martial Hebert, and Maren Bennewitz. World Modeling. In *Springer handbook of robotics*, pages 1135–1152. Springer, 2016.

[61] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. of Robotics Research*, 30(7):846–894, 2011.

[62] Anna Yershova and Steven M LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

[63] Mathieu Labbé and François Michaud. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[64] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018.

[65] François Chaumette, Seth Hutchinson, and Peter Corke. Visual servoing. In *Springer Handbook of Robotics*, pages 841–866. Springer, 2016.

[66] Andrew S Habib, Filippo M Smaldone, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. Handling non-convex constraints in mpc-based humanoid gait generation. In *2022 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 13167–13173, 2022.

[67] Jiatao Ding, Xiaohui Xiao, and Nikos Tsagarakis. Nonlinear optimization of step duration and step location. In *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2259–2265, 2019.

[68] Rubens J.M. Afonso, Marcos R. O. A. Maximo, and Roberto K.H. Galvão. Task allocation and trajectory planning for multiple agents in the presence of obstacle and connectivity constraints with mixed-integer linear programming. *Int. Journal of Robust and Nonlinear Control*, 30(14):5464–5491, 2020.

[69] David Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[70] Sébastien Gros, Mario Zanon, Rien Quirynen, Alberto Bemporad, and Moritz Diehl. From linear to nonlinear MPC: bridging the gap via the real-time iteration. *International Journal of Control*, 2020.

[71] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. *Lecture Notes in Control and Information Sciences*, pages 391–417, 2009.

[72] M. Diehl, H.G. Bock, and J.P. Schloder. Real-time iterations for nonlinear optimal feedback control. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 5871–5876, 2005.

[73] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 2021.

[74] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005.

[75] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012.

[76] C. R. Hargraves and Stephen W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance Control and Dynamics*, 10:338–342, 1986.

[77] Mario Zanon, Andrea Boccia, Vryan Gil S. Palma, Sonja Parenti, and Ilaria Xausa. *Direct Optimal Control and Model Predictive Control*, pages 263–382. Springer International Publishing, Cham, 2017.

[78] H.G. Bock and K.J. Plitt. A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

[79] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, third edition, 2016.

[80] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, 2. ed. edition, 2006.

[81] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[82] Abhishek Goud Pandala, Yanran Ding, and Hae-Won Park. qpswift: A real-time sparse quadratic program solver for robotic applications. *IEEE Robotics and Automation Letters*, 4(4):3355–3362, 2019.

[83] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[84] Gianluca Frison and Moritz Diehl. Hpipm: a high-performance quadratic programming framework for model predictive control. *ArXiv*, abs/2003.02547, 2020.

[85] Moritz Diehl, H.Georg Bock, Johannes P. Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

[86] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[87] Spyridon G. Tarantos and Giuseppe Oriolo. A Dynamics-Aware NMPC Method for Robot Navigation Among Moving Obstacles. In *Intelligent Autonomous Systems 17*, pages 216–230. Springer Nature Switzerland, 2023.

[88] Mario Zanon, Janick V. Frasch, Milan Vukov, Sebastian Sager, and Moritz Diehl. *Model Predictive Control of Autonomous Vehicles*, pages 41–57. Springer International Publishing, 2014.

[89] Manuel Beglini, Tommaso Belvedere, Leonardo Lanari, and Giuseppe Oriolo. An intrinsically stable mpc approach for anti-jackknifing control of tractor-trailer vehicles. *IEEE/ASME Transactions on Mechatronics*, 27(6):4417–4428, 2022.

[90] Marko Bjelonic, Ruben Grandia, Oliver Harley, Cla Galliard, Samuel Zimmermann, and Marco Hutter. Whole-body mpc and online gait sequence generation for wheeled-legged robots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 8388–8395. IEEE Press, 2021.

[91] Giuseppe Oriolo, Alessandro De Luca, and Marilena Vendittelli. WMR Control via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation. *IEEE Transactions on Control Systems Technology*, 2002.

[92] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.

[93] Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[94] Gurobi Optimization, Inc. Gurobi optimizer reference manual.

[95] Andrea Cherubini, Robin Passama, Benjamin Navarro, Mohamed Sorour, Abdellah Khelloufi, Osama Mazhar, Sonny Tarbouriech, Jihong Zhu, Olivier Tempier, André Crosnier, Philippe Fraisse, and Sofiane Ramdani. A collaborative robot for the factory of the future: BAZAR. *The International Journal of Advanced Manufacturing Technology*, 2019.

[96] Dustin J. Webb and Jur van den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013.