SAPIENZA
UNIVERSITÀ DI ROMA

PhD Thesis

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

Prof. TBD (chairman)

Prof. TBD

Prof. TBD

PhD Thesis
Sapienza University of Rome

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

Version: January 16, 2024

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

Dedication.

Acknowledgments

Acknowedgments.

Abstract

Abstract.

Contents

Chapter 1

Introduction

Todo.

1.1 Section

Todo.

Chapter 2

Literature review

Todo.

Chapter 3

Humanoid motion generation in a world of stairs

Todo.

3.1 Introduction

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.

In this paper 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 paper, 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.

In the following, we present a review of the state-of-the-art by separating the literature that is more focused on the planning aspect from the literature about gait generation and humanoid robot control. Subsequently, we will describe the contributions of the paper, focusing on the improvements with respect to the state-of-the-art.

3.1.1 Related Work

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., [1, 2, 3]) 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 [4]: 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 [5] 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 [6], 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 [7], 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.

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 [8], A* [9] and MIQP [10]. Other methods maintain a more complete representation of the environment by employing an elevation map. Examples can be found in [11, 12], 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, [13] proposed to use adaptive sets of possible foot displacements in an A*-based planner, which proved to be effective in relatively simple scenarios. Alternatively, [14] 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.

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) [15] 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 [16] 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) [17], 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 [18, 19]. 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.

3.1.2 Contribution and Paper Organization

In this paper, 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.

With respect to the previously reviewed contributions, our work improves the following points:

- the proposed footstep planner finds sequences of footsteps that are optimal with respect to a chosen optimality criterion, whereas most of the literature that makes use of randomized methods does not in any way consider the quality of the produced sequences [7]; we do this by using a planner based on RRT*, which is a probabilistic method with guarantees of asymptotic convergence to an optimal solution;
- in our formulation, optimality does not significantly compromise performance, unlike techniques that make use of A*-based algorithms [9, 11, 13], or others that make use of quadratic optimization [4];
- we propose a scheme capable of working both with off-line planning, in known environments, and in on-line situations;
- to generate a gait that realizes the footstep plan we adopt and improve our Intrinsically Stable MPC (IS-MPC) scheme [20]. IS-MPC incorporates an explicit stability constraint in order to deal with the instability in the humanoid robot dynamics. Along with the stability constraint, we enforce a ZMP position constraint and a ZMP velocity constraint, and show that the combination of all the imposed constraints guarantees satisfaction of the balance condition in 3D.

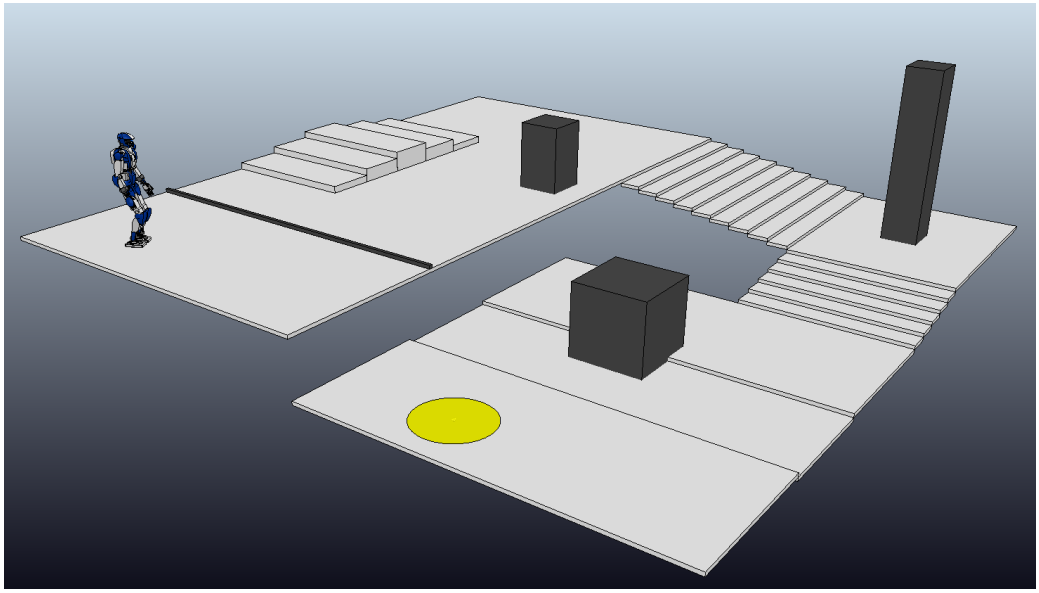

Figure 3.1. An instance of the considered problem. The robot must reach the goal region (in yellow) by traversing a *world of stairs*. Patches that are not visible correspond to infinitely deep holes or trenches.

Compared to our previous work on the subject, [19], the main additions presented in this paper consist of the introduction of optimality criteria in the planner, the extension to on-line situations with the use of on-board sensors, and the inclusion of the ZMP velocity constraint in the IS-MPC. We validate the proposed architectures for the off-line and on-line case via simulations on the HRP-4 humanoid robot in scenarios of different complexity.

The paper is structured as follows. Section 3.2 formulates the problem both in the off-line and in the on-line case. Section 3.3 presents in detail the off-line case by discussing its general architecture in Sect. 3.3.1, describing the footstep planner and the scenarios used for testing in Sect. 3.3.2, the gait generation module together with theoretical developments in Sect. 3.3.3, the localization module in Sect. 3.3.4 and the simulations in Sect. 3.3.5. Section 3.4 extends the work to the on-line case, presenting its architecture in Sect. 3.4.1, the mapping module in Sect. 3.4.2, the sensor-based footstep planner in Sect. 3.4.3, the visual task generation module in Sect. 3.4.4 and simulations in both static and dynamic environments in Sect. 3.4.5. Section 3.5 puts the paper in perspective with respect to the state of the art, and Sect. 3.6 concludes the paper with mentions to possible future work.

3.2 Problem Formulation

In the situation of interest (Fig. 3.1), 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

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

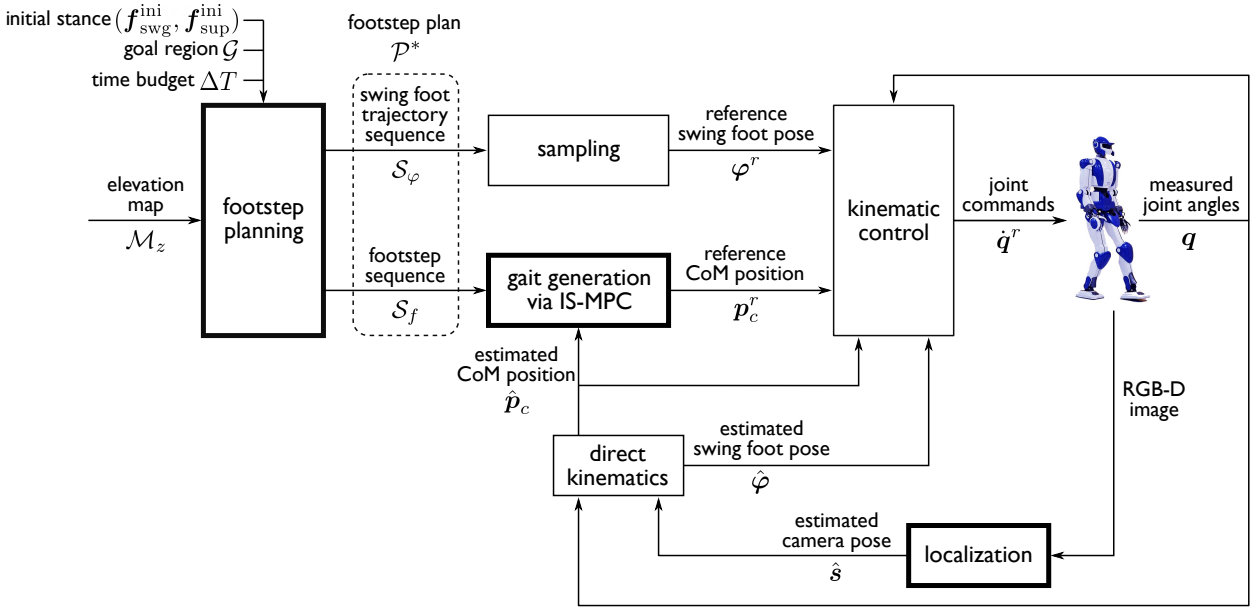

Figure 3.2. Block scheme of the proposed solution approach for the off-line case.

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. 3.1). 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 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) [21]. 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.

3.3 The off-line Case

We start this section by describing the general structure of the proposed method. Then we will describe in detail its main components, i.e., the footstep planner and the gait generation scheme based on IS-MPC. Some simulation results will also be presented.

3.3.1 General Architecture

To solve the described problem in the off-line case, we adopt the architecture shown in Fig. 3.2, 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 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

[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. To support this claim, we explicitly verified that Assumption A4 holds in our simulations, see the related comment toward the end of Sect. 3.3.5. 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.

[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.

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] ($\boldsymbol{f}_{\text{swg}}^{\text{ini}}, \boldsymbol{f}_{\text{sup}}^{\text{ini}}$) 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 $\boldsymbol{f}^1 = \boldsymbol{f}_{\text{swg}}^{\text{ini}}$ and $\boldsymbol{f}^2 = \boldsymbol{f}_{\text{sup}}^{\text{ini}}$. 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_s^j is split in T_{ss}^j and T_{ds}^j, 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 $t_s^j = t_s^{j-1} + T_s^{j-1}$, with $t_s^1 = 0$.

Once the footstep plan has been generated, the sequence of footsteps $\mathcal{S}_f$ is sent to a gait generator based on Intrinsically Stable MPC (IS-MPC), 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 $\boldsymbol{p}_{\text{c}}^r$ 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, $\boldsymbol{p}_{\text{c}}^r$ 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.

3.3.2 Footstep Planning

The input data for this module are the initial robot stance ($\boldsymbol{f}_{\text{swg}}^{\text{ini}}, \boldsymbol{f}_{\text{sup}}^{\text{ini}}$), 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.

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

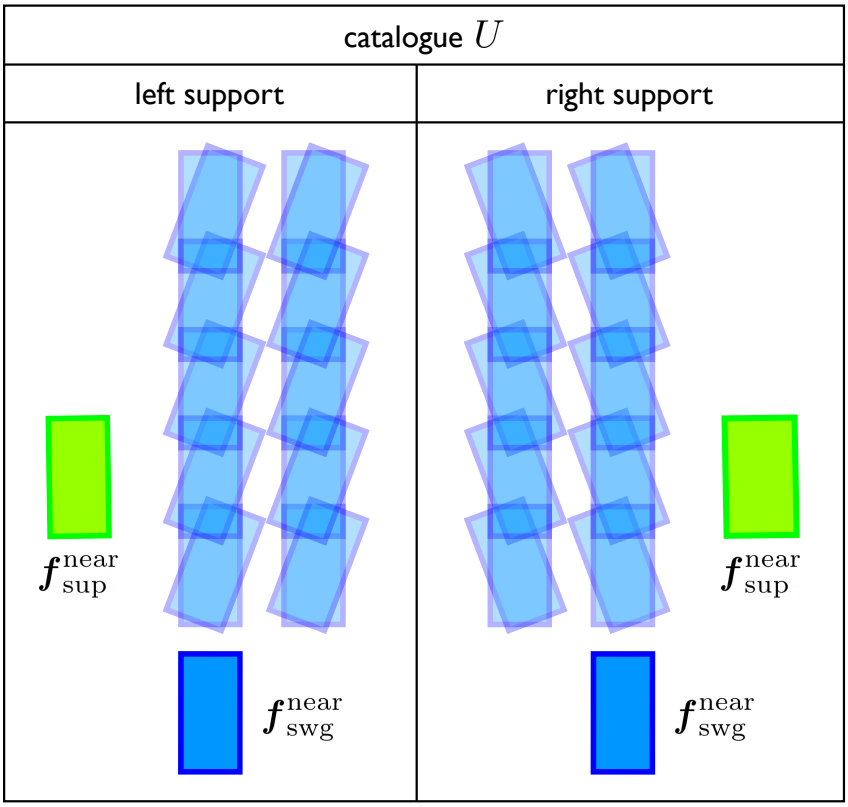

Figure 3.3. An example catalogue U of primitives, containing a finite set of landings for the swing foot with respect to the left or right support foot.

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

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. 3.3.

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

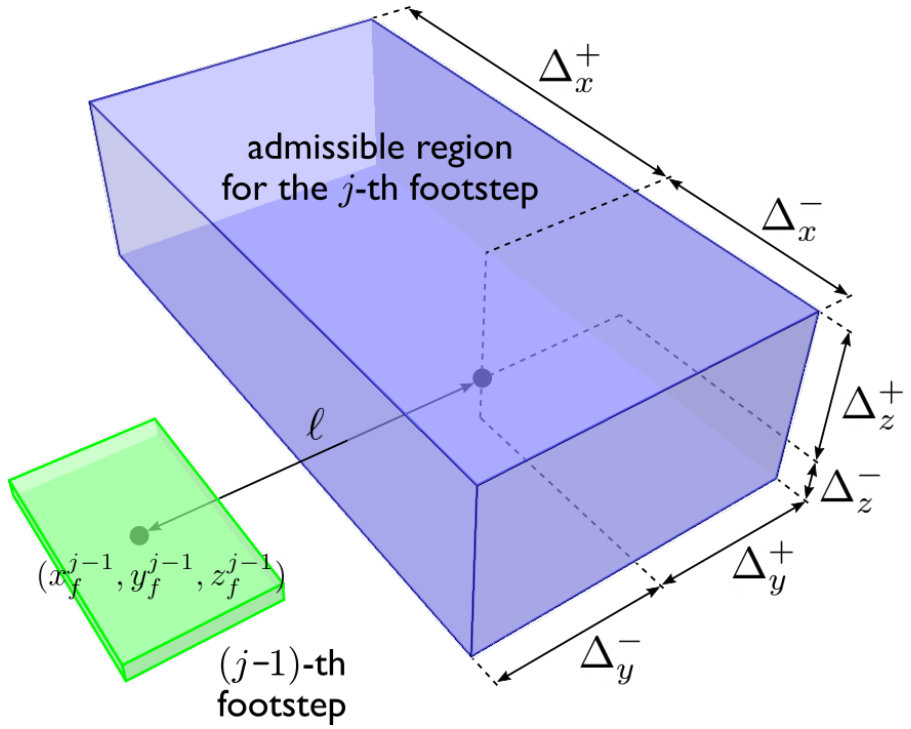

Figure 3.4. The 3D admissible region identified by the first two kinematic constraints of requirement R2, i.e., eqs. (3.1– 3.2). Footstep orientation is not represented.

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*).

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:

$$-\begin{pmatrix}\Delta_x^-\\ \Delta_y^-\end{pmatrix} \le R_{j-1}^T \begin{pmatrix}x_f^j - x_f^{j-1}\\ y_f^j - y_f^{j-1}\end{pmatrix} \pm \begin{pmatrix}0\\ \ell\end{pmatrix} \le \begin{pmatrix}\Delta_x^+\\ \Delta_y^+\end{pmatrix} \tag{3.1}$$

$$-\Delta_z^- \le z_f^j - z_f^{j-1} \le \Delta_z^+ \tag{3.2}$$

$$-\Delta_\theta^- \le \theta_f^j - \theta_f^{j-1} \le \Delta_\theta^+. \tag{3.3}$$

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. 3.4.

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. 3.3.2, 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 hip (Fig. 3.5). 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

$$\gamma(\boldsymbol{f}, \boldsymbol{f}') = \|\boldsymbol{p}_f - \boldsymbol{p}'_f\| + k_\gamma |\theta_f - \theta'_f| \tag{3.4}$$

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

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

$$l_1(v^a, v^b) = 1, \tag{3.5}$$

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).

[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 [22].

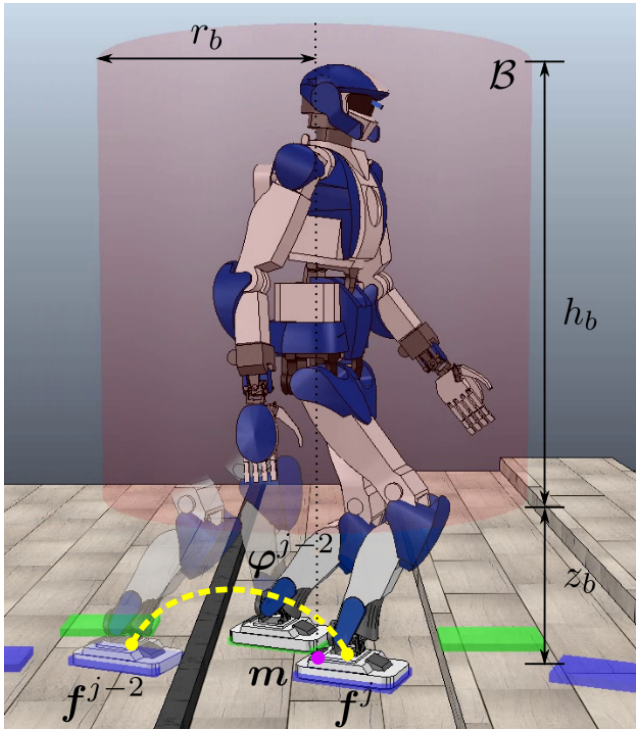

Figure 3.5. Visual representation of the process for checking requirement R3. The red cylinder accounts for the maximum occupancy of the humanoid upper body, while the yellow line represents the swing foot trajectory.

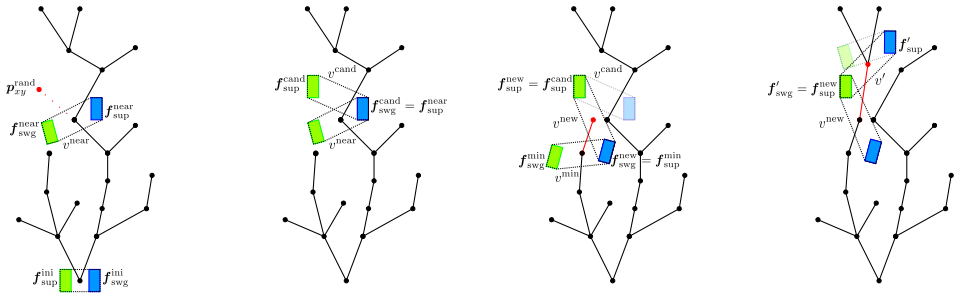

Figure 3.6. The four steps of a generic iteration of the footstep planner. From left to right: selecting a vertex for expansion, generating a candidate vertex, choosing a parent, rewiring.

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

$$l_2(v^a, v^b) = |z^b_{f,\text{sup}} - z^a_{f,\text{swg}}|, \tag{3.6}$$

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

$$l_3(v^a, v^b) = \frac{1}{\sigma(\boldsymbol{f}^b_{\text{sup}})}, \tag{3.7}$$

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

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. 3.6.

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 $\boldsymbol{p}_{xy}$. ◀

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. 3.3). In particular, we set the support foot to $\boldsymbol{f}^{\text{near}}_{\text{sup}}$ and randomly select 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 (3.1–3.3) of requirement R2. Call $\boldsymbol{f}^{\text{cand}}_{\text{sup}}$ the chosen candidate footstep. The z coordinate to be associated to the footstep $\boldsymbol{f}^{\text{cand}}_{\text{sup}}$ is then retrieved from $\mathcal{M}_z$, and both the requirement R1 and the last condition (3.2) of R2 can now be checked.

To test the last requirement R3, we invoke an engine that generates a swing foot trajectory $\boldsymbol{\varphi}^{\text{near}}$ from $\boldsymbol{f}^{\text{near}}_{\text{swg}}$ to $\boldsymbol{f}^{\text{cand}}_{\text{sup}}$. 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 $[h^{\min}, h^{\max}]$ looking for a collision-free trajectory.

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

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

If all requirements have been satisfied, a candidate vertex is generated as $v^{\text{cand}} = (\boldsymbol{f}^{\text{cand}}_{\text{swg}}, \boldsymbol{f}^{\text{cand}}_{\text{sup}})$, with $\boldsymbol{f}^{\text{cand}}_{\text{swg}} = \boldsymbol{f}^{\text{near}}_{\text{sup}}$; 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. ◀

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 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 $v^{\text{min}} = (\boldsymbol{f}^{\text{min}}_{\text{swg}}, \boldsymbol{f}^{\text{min}}_{\text{sup}})$ that allows to reach v^{cand} with minimum cost is chosen as its parent. If $v^{\text{min}} = v^{\text{near}}$, then v^{cand} can be added to the tree together with the edge joining it to v^{near}. However, if a different parent $v^{\text{min}} \neq v^{\text{near}}$ 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 $\boldsymbol{f}^{\text{new}}_{\text{swg}} = \boldsymbol{f}^{\text{min}}_{\text{sup}}$ 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 $\boldsymbol{\varphi}^{\text{min}}$. ◀

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 $\boldsymbol{f}^{\text{new}}_{\text{sup}}$, 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].

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. ◀

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

$$v^* = \underset{v \in \mathcal{V}_{\text{goal}}}{\text{argmin}}\ c(v) \tag{3.8}$$

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

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 [23], although we do not have a formal proof yet.

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

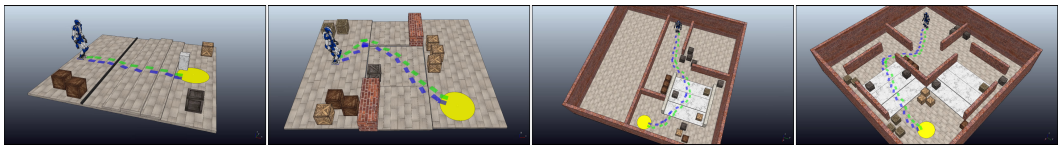

Figure 3.8. Examples of footstep plans found in the scenarios *Rod*, *Ditch*, *Corridor* and *Maze*, respectively, when minimizing the number of steps.

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 [24], 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. 3.7) 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], 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. 3.3). 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,

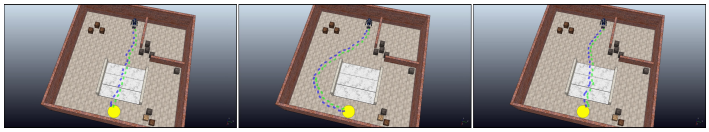

Figure 3.9. Examples of footstep plans found in the scenario *Spacious* when minimizing the number of steps, minimizing the height variation and maximizing the clearance, respectively.

$\Delta_y^- = 0.07$ 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. 3.3.2 are considered in each scenario.

Tables 3.1–3.3 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. 3.3.2, 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 3.8 shows the plans generated by minimizing the number of steps in the scenarios *Rod*, *Ditch*, *Corridor* and *Maze*. In particular, in *Rod* the plan allow 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 3.9 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.

3.3.3 Gait Generation via IS-MPC

The gait generation module receives the planned footstep sequence $\mathcal{S}_f$ as input, and it is in charge of producing a CoM trajectory that the robot can safely track in order to step over said footstep sequence. Along the entire motion, dynamic balance must be maintained.

On flat ground, it is common to ensure dynamic balance as a geometric criterion, by requiring that the ZMP must always be inside the convex hull of contact surfaces, i.e., the *support polygon*. Traditionally, the ZMP is assumed to be located on the ground plane, which is uniquely defined when the environment is flat.

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 3.1. 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 3.2. 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 3.3. Performance of the off-line footstep planner when maximizing the minimum clearance.

In non-flat environments, there is no unique ground surface on which the ZMP can be assumed to be. However, the aforementioned criterion can be extended to these cases by allowing the ZMP to move in 3D [16]. The balance criterion is satisfied as long as the ZMP is inside a three-dimensional *support region* $\mathcal{Z}$ that takes the shape of a pyramid (see Fig. 3.10). The vertex of this pyramid is the robot CoM, and the edges are the lines connecting the CoM to the vertexes of the convex hull of the contact surfaces.

In MPC, dynamic balance is enforced via constraints on the ZMP position. However, the 3D support region $\mathcal{Z}$ cannot be directly employed to enforce a ZMP constraint, because this would be nonlinear, meaning that the resulting optimization problem would not be in a standard linear-quadratic formulation. The cause of this nonlinearity is given by the fact that the vertex of the pyramid $\mathcal{Z}$ is the CoM. Since both the ZMP and the CoM depend on the decision variables of the QP, this would result in product between the decision variables themselves.

In order to avoid this, we will define a smaller region, independent of the CoM position, where the ZMP is allowed to be. Furthermore, we will prove that this region conservatively approximates the actual support region $\mathcal{Z}$, and thus that the balance condition is always satisfied under the imposed constraints.

In this subsection we will describe the MPC gait generation scheme that is used in the proposed formulation. In particular, we will derive the prediction model and define the constraints that ensure stability and dynamic balance. Finally, we will state the QP problem to be solved at each iteration, and give a sketch of the complete algorithm.

Prediction Model

Control of the ZMP is achieved using a dynamic model relating the position of the latter to the position and acceleration of the CoM. This dynamic model can be derived by balancing moments on the humanoid as a whole, and assuming that the rate of change of angular momentum around the CoM can be neglected. With this in mind, denoting the CoM as $\boldsymbol{p}_c = (x_c, y_c, z_c)$ and the ZMP as $\boldsymbol{p}_z = (x_z, y_z, z_z)$, we get

$$\begin{aligned}(z_c - z_z)\ddot{x}_c &= (x_c - x_z)(\ddot{z}_c - g)\\ (z_c - z_z)\ddot{y}_c &= (y_c - y_z)(\ddot{z}_c - g),\end{aligned} \tag{3.9}$$

where g is the gravity acceleration.

This model exhibits a nonlinear coupling between the vertical and the horizontal components of $\boldsymbol{p}_c$ and $\ddot{\boldsymbol{p}}_c$. On flat ground, this nonlinearity is usually handled by assuming a constant CoM height, which leads to the well-known LIP model [15]. In order to allow for vertical movement of the CoM, a different choice is made here, which is to constrain the motion of the CoM to satisfy the relation $(\ddot{z}_c - g)/(z_c - z_z) = \eta^2$, where η is a constant parameter. The resulting dynamic model can be expressed in the form

$$\ddot{\boldsymbol{p}}_c = \eta^2(\boldsymbol{p}_c - \boldsymbol{p}_z) - \boldsymbol{g}, \tag{3.10}$$

where $\boldsymbol{g} = (0 \;\; 0 \;\; g)^T$ is the gravity acceleration vector. This model features a LIP-like dynamic behavior along all three axes. The only difference with respect to a standard

LIP is given by the gravity vector $\boldsymbol{g}$ acting as a constant drift. This causes the system not to be in equilibrium when the CoM and ZMP coincide, but rather when they are displaced by $\boldsymbol{g}/\eta^2$.

The choice of restricting the available trajectories to those resulting in a constant η allows to make the prediction model linear. If this restriction is removed, the model is referred to as Variable-Height Inverted Pendulum, which can be treated either as nonlinear or time-varying. This can allow for more general motions to be generated, e.g., running [25], at the cost of a slightly more complex architecture. For the present case, where only walking is considered, the simpler model is preferred.

In order to obtain smoother trajectories, model (3.10) is dynamically extended to have the derivative of the ZMP $\dot{\boldsymbol{p}}_z$ as the input. The 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., $\dot{\boldsymbol{p}}_z(t) = \dot{\boldsymbol{p}}_z^k$ for $t \in [t_k, t_{k+1})$. This 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 steps that are contained, either fully or partially, within this control horizon is denoted as F.

Stability Constraint

Model (3.10) 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, 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 the unstable component of the dynamics in order to guarantee that the CoM trajectory does not diverge with respect to the ZMP.

The unstable component of system (3.10) is highlighted by the coordinate $\boldsymbol{p}_u = \boldsymbol{p}_c + \dot{\boldsymbol{p}}_c/\eta$, also referred to as Divergent Component of Motion (DCM) or Capture Point (CP), that evolves according to the dynamics

$$\dot{\boldsymbol{p}}_u = \eta(\boldsymbol{p}_u - \boldsymbol{p}_z) - \frac{\boldsymbol{g}}{\eta}. \tag{3.11}$$

Despite the instability, the evolution of the system is bounded if the following *stability condition* is satisfied

$$\boldsymbol{p}_u^k = \eta \int_{t_k}^{\infty} e^{-\eta(\tau - t_k)} \boldsymbol{p}_z(\tau) d\tau + \frac{\boldsymbol{g}}{\eta^2}, \tag{3.12}$$

as stated by the following proposition.

Proposition 1 *Consider system (3.10). If the ZMP velocity is bounded,* $|\dot{x}_z| \leq v^{\max}$, $|\dot{y}_z| \leq v^{\max}$, $|\dot{z}_z| \leq v^{\max}$, *for some* $v^{\max} > 0$, *and condition (3.12) is satisfied, then the following bound holds:*

$$\frac{\boldsymbol{g}}{\eta^2} - \frac{v^{\max}}{\eta}\mathbf{1} \leq \boldsymbol{p}_c - \boldsymbol{p}_z \leq \frac{\boldsymbol{g}}{\eta^2} + \frac{v^{\max}}{\eta}\mathbf{1}, \tag{3.13}$$

where $\mathbf{1}$ *is a vector with all components equal to 1. Moreover, (3.13) implies that the system is internally stable, i.e., the CoM is bounded with respect to the ZMP.*

Proof. See Appendix.

Condition (3.12) 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 $\tilde{\boldsymbol{x}}_z$. In [20], the anticipative tail was used to prove recursive feasibility and stability of the MPC scheme.

The stability constraint is then written as

$$\eta \int_{t_k}^{t_{k+C}} e^{-\eta(\tau - t_k)} \boldsymbol{p}_z d\tau = \boldsymbol{p}_u^k - \tilde{\boldsymbol{c}}^k - \frac{\boldsymbol{g}}{\eta^2}. \tag{3.14}$$

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

$$\tilde{\boldsymbol{c}}^k = \eta \int_{t_{k+C}}^{\infty} e^{-\eta(\tau - t_k)} \tilde{\boldsymbol{p}}_z d\tau. \tag{3.15}$$

Note that in [20] we considered the footstep plan to be available over a receding window called the *preview horizon*. Here there is no need to make such an assumption, as the footstep plan is provided in its entirety, and once the goal is reached the robot comes to a complete stop.

Enforcing constraint (3.14) allows, similarly to what stated in Prop. 1, 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. Because of this fact, we will assume in the following that Prop. 1 is valid as stated, even though a small numerical correction should be applied to make up for the difference between the stability condition and the stability constraint.

ZMP Velocity Constraint

This constraint imposes a limit on how fast the ZMP can move, i.e.,

$$|\dot{x}_z| \leq v^{\max}, \quad |\dot{y}_z| \leq v^{\max}, \quad |\dot{z}_z| \leq v^{\max} \tag{3.16}$$

Enforcing such limit allows to indirectly control the maximum CoM/ZMP displacement, by using the result of Prop. 1. This will be useful when defining the ZMP position constraint, as will be made clear in the following paragraphs.

ZMP Position Constraint

As already noted, the humanoid is balanced as long as the ZMP is inside the pyramid $\mathcal{Z}$, which is a nonlinear condition due to the vertex of the pyramid being at the CoM (Fig. 3.10). To preserve linearity we consider a smaller allowed region for the ZMP, consisting in a box of fixed size with changing center and orientation. We refer to this as the *moving box*[9], and we will prove that it conservatively approximates the support region, meaning that it is always contained inside the pyramid $\mathcal{Z}$.

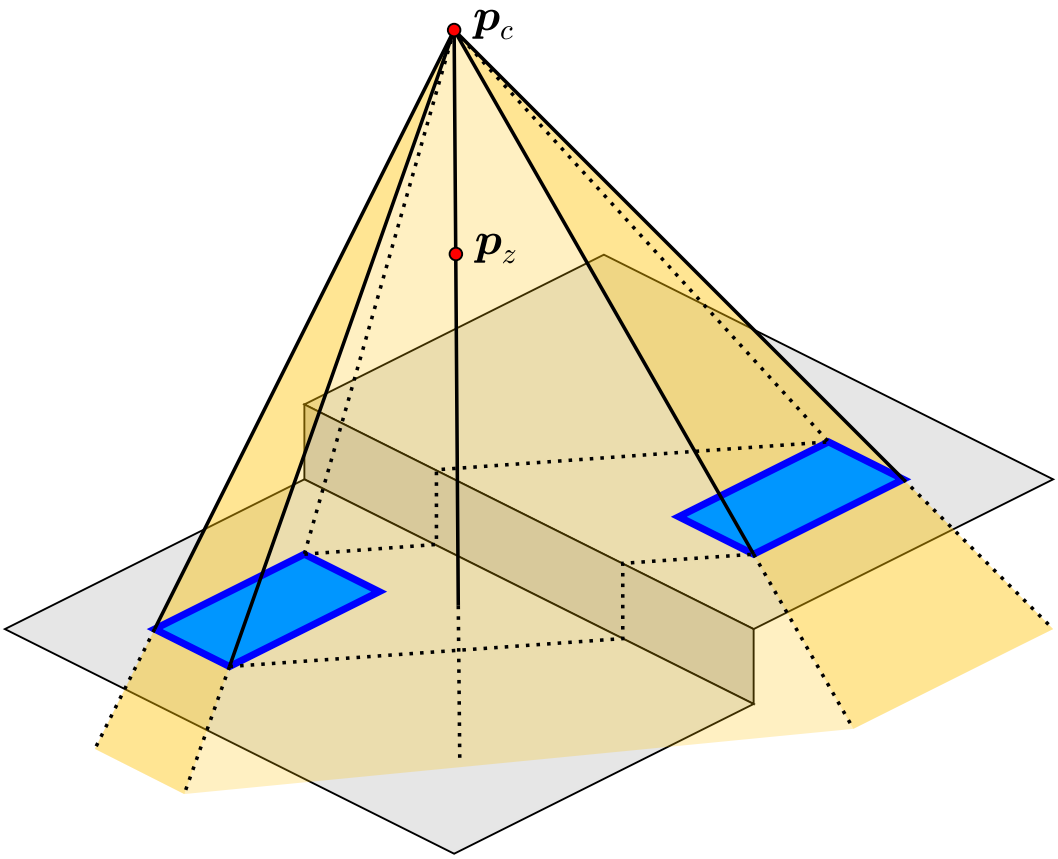

Figure 3.10. Balance condition in 3D: the ZMP must lie inside the yellow pyramid.

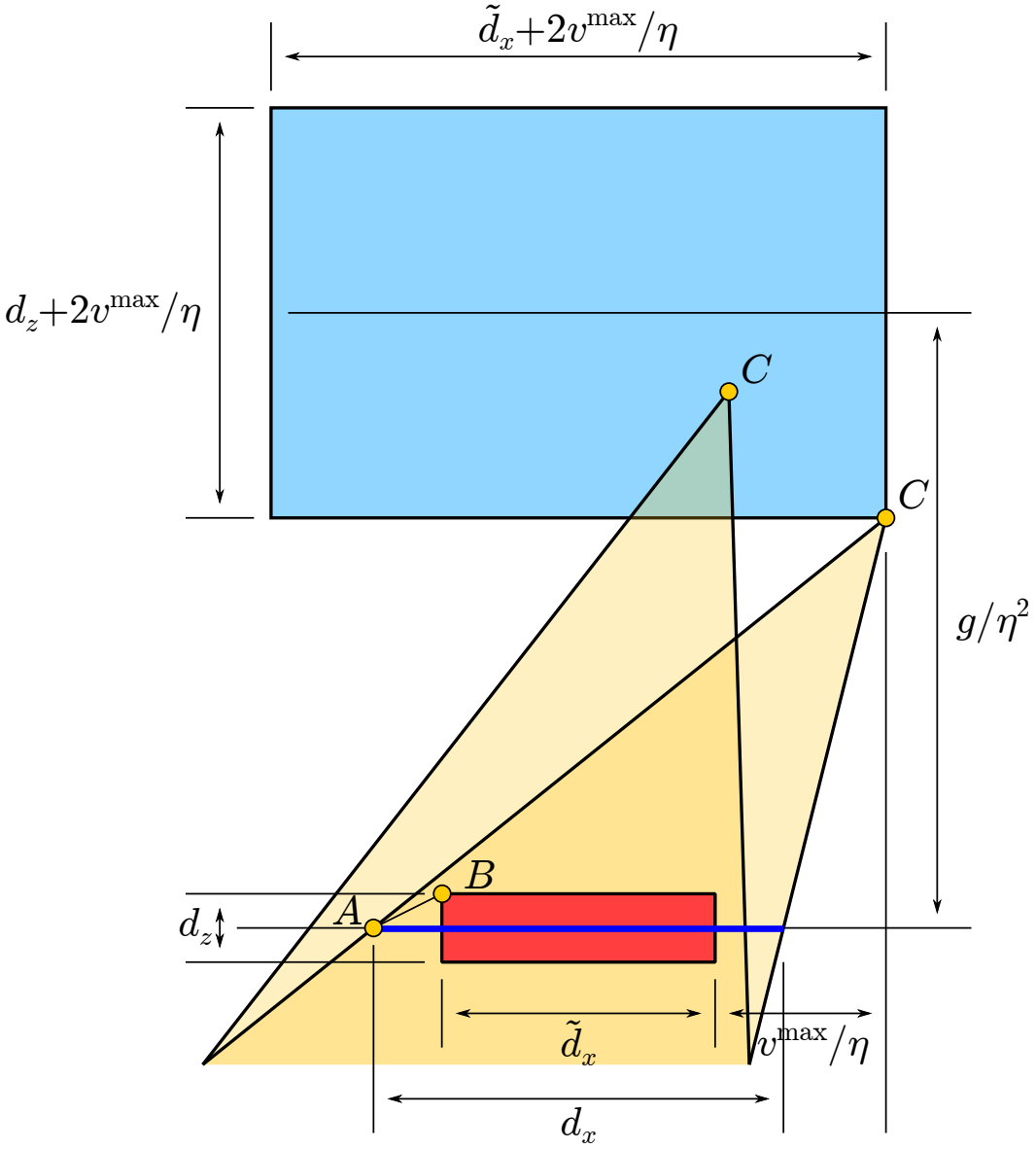

Figure 3.11. The red rectangle represents a section of the moving box in the x-z plane. This region is fully contained inside the yellow pyramid as long as the CoM belongs to the blue region. Here, two scenarios are considered: one in which the CoM is well within the blue region, and a worst-case scenario in which the CoM is in a lower vertex of the blue region.

The center $\boldsymbol{p}_{\text{mc}}$ and orientation θ_{mc} of the moving box are taken to be consistent with the pose of footstep $\boldsymbol{f}^j$ during the j-th single support phase. In the following double support phase they will gradually slide in order to reach the pose of the next footstep $\boldsymbol{f}^{j+1}$, with a linear[10] timing law. At each time t, the center of the moving box $\boldsymbol{p}_{\text{mc}}$ is expressed as

$$\boldsymbol{p}_{\text{mc}}(t) = \begin{cases} \boldsymbol{p}^j & t \in [t_s^j, t_s^j + T_{\text{ss}}^j) \\ (1 - \alpha^j(t))\boldsymbol{p}^j + \alpha^j(t)\boldsymbol{p}^{j+1} & t \in [t_s^j + T_{\text{ss}}^j, t_s^{k+1}) \end{cases} \tag{3.17}$$

where $j = 0, \ldots, F$ is a index over the footsteps within the control horizon, and $\alpha^j(t) = (t - t_s^j - T_{\text{ss}}^j)/T_{\text{ds}}^j$ denotes the time elapsed since the start of the double support phase, expressed as a fraction of the duration of the double support phase itself. The orientation of the moving box θ_{mc} can be similarly expressed as

$$\theta_{\text{mc}}(t) = \begin{cases} \theta^j & t \in [t_s^j, t_s^j + T_{\text{ss}}^j) \\ \text{anglin}(\theta^j, \theta^{j+1}, \alpha^j(t)) & t \in [t_s^j + T_{\text{ss}}^j, t_s^{k+1}) \end{cases} \tag{3.18}$$

where $j = 0, \ldots, F$, and anglin is a function[11] that computes a linear combination of angles in such a way to correctly account for wrapping around $\pm\pi$.

The ZMP position constraint is expressed as

$$-\tilde{\boldsymbol{d}}/2 \leq \boldsymbol{R}_{k+i}^T(\boldsymbol{p}_z^{k+i} - \boldsymbol{p}_{\text{mc}}^{k+i}) \leq \tilde{\boldsymbol{d}}/2 \tag{3.19}$$

where $\boldsymbol{p}_z^{k+i}$ and $\boldsymbol{p}_{\text{mc}}^{k+i}$ respectively denote the ZMP and the center of the moving box sampled at time t_{k+i}, $\tilde{\boldsymbol{d}} = (\tilde{d}_x, \tilde{d}_y, d_z)^T$ is a vector collecting the dimensions of the moving box along all three axes, and $\boldsymbol{R}_{k+i}$ is the rotation matrix associated with θ_{mc}^{k+i}.

The size of the moving box $\tilde{\boldsymbol{d}} = (\tilde{d}_x, \tilde{d}_y, d_z)^T$ is determined in such a way to always be contained inside the pyramid $\mathcal{Z}$, as expressed by the following proposition.

Proposition 2 *Assume the stability constraint (3.14) and the ZMP velocity constraint (3.16) are enforced. If* $v^{\max}$ *is chosen in such a way that*

$$v^{\max} \leq \min\{v_y^{\max}, v_x^{\max}\}, \tag{3.20}$$

[9] Approximating the pyramidal region $\mathcal{Z}$ 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.

[10] The timing law can be arbitrarily chosen as long as it leads the moving box from one footstep to the next within the duration of the double support phase.

[11] For two generic angles θ_a and θ_b, linearly combined with a weight α, the function anglin can be defined as

$$\begin{aligned} \text{anglin}(\theta_a, \theta_b, \alpha) = \text{atan2}(&\sin((1-\alpha)\theta_a) + \sin(\alpha\theta_b), \\ &\cos((1-\alpha)\theta_a) + \cos(\alpha\theta_b)). \end{aligned}$$

Note that this definition is meaningless when the two angles are separated by exactly π, but this can never occur as the angle between consecutive footsteps is limited by requirement R2.

where $v_x^{\max}$ *and* $v_y^{\max}$ *are given by the following compact expression*

$$v_{x,y}^{\max} = \eta \left(g/\eta^2 - d_{x,y} \frac{d_z}{d_{x,y} - \tilde{d}_{x,y}} \right) \left(1 + \frac{d_z}{d_{x,y} - \tilde{d}_{x,y}} \right)^{-1}, \tag{3.21}$$

with d_x *and* d_y *denoting the size of the actual footprint, then the moving box is always contained in the pyramid* $\mathcal{Z}$*.*

Proof: Consider the geometric construction in Fig. 3.11. This construction shows a projection of the pyramid $\mathcal{Z}$ in the x-z plane during a single support phase. The red area represents a cross section of the moving box, with dimensions $\tilde{d}_x$ and d_z. The blue area represents the region where the CoM can be found, given the CoM/ZMP bounds (3.13). In the figure two different possibilities are depicted for the CoM position (identified by the point C). Among them, consider the worst-case scenario, in which the CoM is in the bottom-right corner of the blue region[12]. A condition for which the red region is always fully contained inside $\mathcal{Z}$ can be found by imposing that the slope of the segment AB is less than the slope of the segment AC, i.e.,

$$\frac{d_z/2}{d_x/2 - \tilde{d}_x/2} \leq \frac{g/\eta^2 - d_z/2 - v_x^{\max}/\eta}{d_x/2 + \tilde{d}_x/2 + v_x^{\max}/\eta}, \tag{3.22}$$

which, after simple manipulations, leads to (3.21). During double support phases the construction of Fig. 3.11 is not accurate anymore, as the pyramid $\mathcal{Z}$ is much larger than the one represented. However, it is not necessary to repeat the reasoning, as a smaller pyramid, analogous to that shown in Fig. 3.11, can always be constructed and will always be contained inside $\mathcal{Z}$. The last step is to repeat the construction in the y-z plane in order to find $v_y^{\max}$, which proves the thesis. ∎

QP Problem

At a generic time t_k, IS-MPC solves the following QP problem

$$\begin{cases} \min\limits_{\dot{\boldsymbol{p}}_z^k, \ldots, \dot{\boldsymbol{p}}_z^{k+C-1}} \quad \sum\limits_{i=0}^{C-1} \left(\|\dot{\boldsymbol{p}}_z^{k+i}\|^2 + \beta \|\boldsymbol{p}_z^{k+i} - \boldsymbol{p}_{\text{mc}}^{k+i}\|^2 \right) \\ \text{subject to:} \\ \quad \bullet \text{ stability constraint (3.14)} \\ \quad \bullet \text{ ZMP velocity constraint (3.16)} \\ \quad \bullet \text{ ZMP position constraint (3.19)} \end{cases}$$

The cost function minimizes the decision variables (ZMP derivative) as a regularization term, and attempts to bring the ZMP close to the center of the moving box, which is typically beneficial as it produces a more robust walking pattern. β is a weight on the second term.

In typical MPC fashion, the first sample of the ZMP velocity $\dot{\boldsymbol{p}}_z^k$ is used to integrate the 3D LIP dynamics (3.10). The resulting CoM trajectory $\boldsymbol{p}_c$ is sent,

[12] An equivalent symmetrical scenario would occur if the CoM were in the bottom-left corner.

together with the swing foot trajectory generated by the footstep planner and sampled at time t_k, to the kinematic controller.

3.3.4 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 time the estimate $\hat{\boldsymbol{s}}$ of the pose of the camera frame. To this end, it uses RTAB-Map [26], 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. 3.2).

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.

3.3.5 Simulations

We performed simulations on the HRP-4 humanoid robot in the CoppeliaSim environment. We tested our off-line framework in multiple environments (Fig. 3.7). 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 3.12 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 3.13 shows the robot moving 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).

In order to verify the validity of Assumption A4, we computed the ground reaction forces associated with the generated trajectories, and verified that the no-slipping condition is always satisfied. Indeed, in our simulations the ratio between the horizontal and vertical components of the ground reaction force peaks around 0.1, which is a completely acceptable value under normal circumstances (e.g., steel-concrete friction coefficients are normally not lower than 0.5 [27]).

We invite the reader to watch the accompanying video, which includes clips related to the above simulations as well as additional cases.

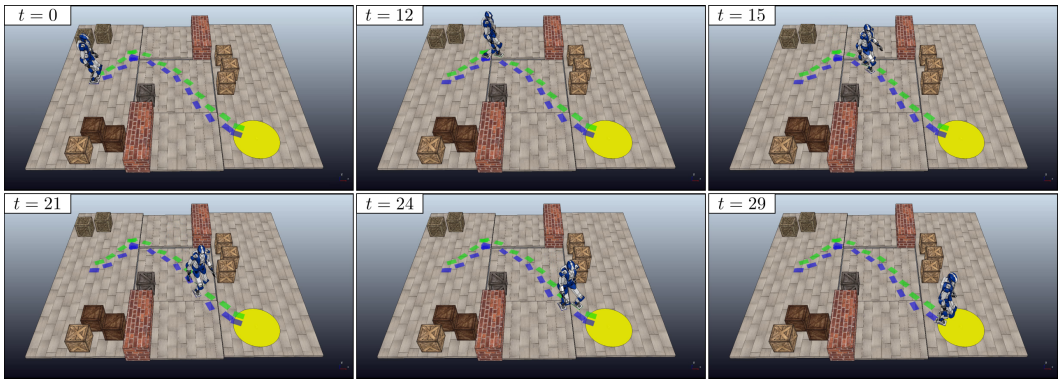

Figure 3.12. The robot reaches the goal going through the ditch, which can only be accessed from the left and exited from the right.

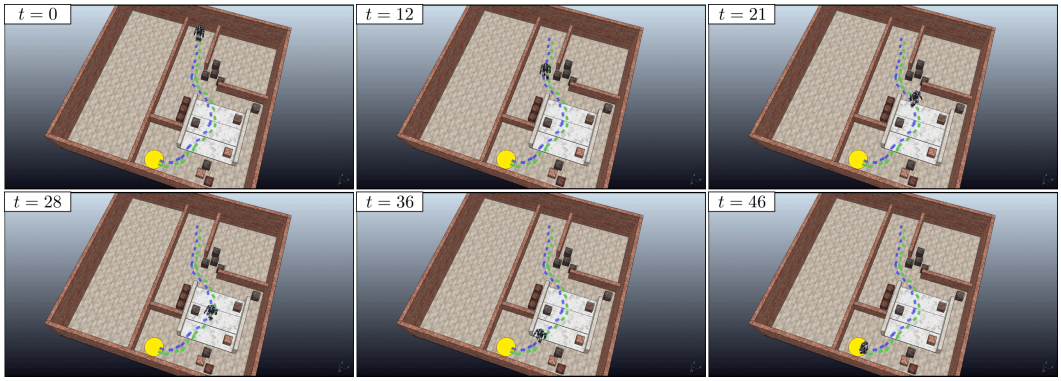

Figure 3.13. The robot reaches the goal avoiding the corridor, climbing and descending the staircase while avoiding the obstacles.

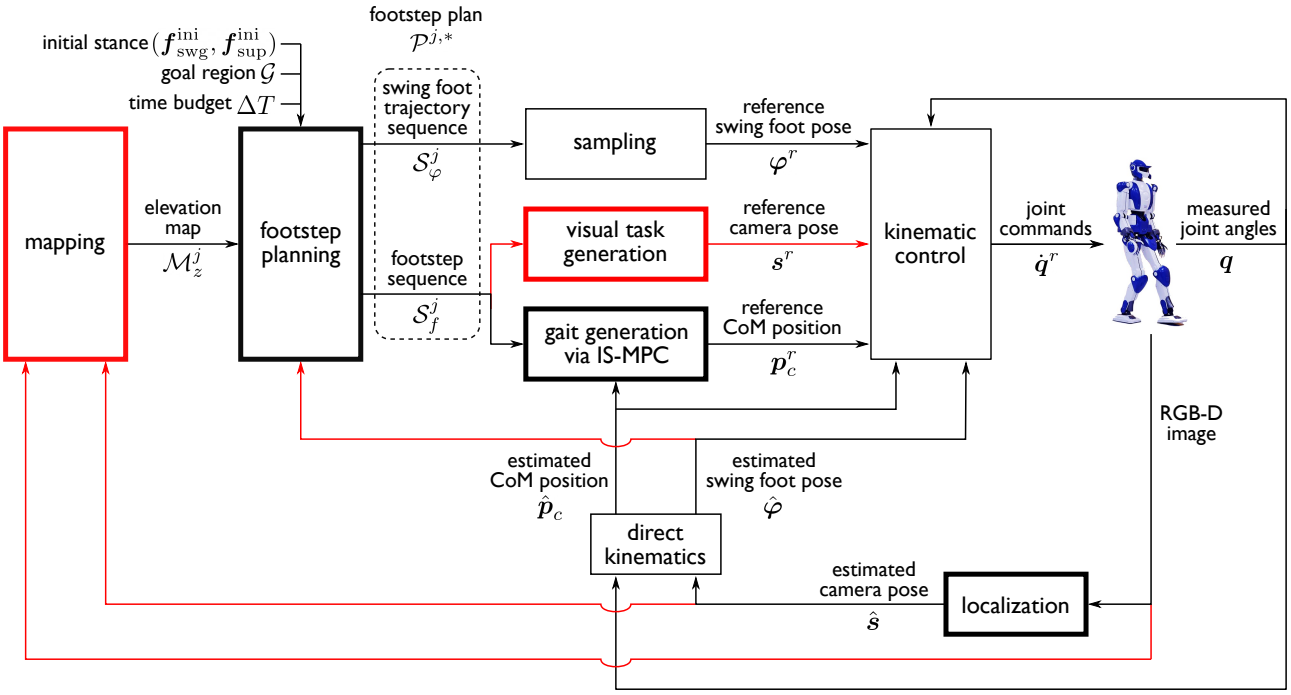

Figure 3.14. Block scheme of the on-line case. The red blocks and arrows highlight the additional modules and signals compared to the off-line case.

3.4 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.

3.4.1 General Architecture

The proposed architecture for the on-line case is given in Fig. 3.14, 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_s^j, i.e., the beginning of the j-th step. Let $(\hat{\boldsymbol{f}}_{\text{swg}}^j, \hat{\boldsymbol{f}}_{\text{sup}}^j)$ be the current stance, with $\hat{\boldsymbol{f}}_{\text{swg}}^j$ and $\hat{\boldsymbol{f}}_{\text{sup}}^j$ the estimates of the swing and support foot poses at t_s^j, and $\mathcal{P}^{j,*} = \{\mathcal{S}_f^j, \mathcal{S}_\varphi^j\}$ 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

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

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

- At any time instant $t \in [t_s^j, t_s^{j+1})$, the current reference position $\boldsymbol{p}_{\text{c}}^r$ of the CoM is produced by the gait generator, based on the sequence $\mathcal{S}_f^j$, 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 $\mathcal{S}_f^j$, that allows to direct the gaze towards the subgoal extracted from $\mathcal{S}_f^j$, and then to enlarge $\mathcal{M}_z$ in the area of the current destination. References $\boldsymbol{p}_{\text{c}}^r$, $\boldsymbol{\varphi}^r$, $\boldsymbol{s}^r$, 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_s^j, the footstep planner is invoked providing in input the stance $(\boldsymbol{f}_{\text{swg}}^{j+1}, \boldsymbol{f}_{\text{sup}}^{j+1})$, the goal region $\mathcal{G}$, a time budget equal to T_s^j, and the elevation map $\mathcal{M}_z^j$ currently available by the mapping module. At t_s^{j+1}, the planner returns a new footstep plan $\mathcal{P}^{j+1,*} = \{\mathcal{S}_f^{j+1}, \mathcal{S}_\varphi^{j+1}\}$, where the sequences $\mathcal{S}_f^{j+1}$ and $\mathcal{S}_\varphi^{j+1}$ are defined similarly to $\mathcal{S}_f^j$ and $\mathcal{S}_\varphi^j$, $\boldsymbol{f}^{j+1|j+1} = \boldsymbol{f}_{\text{swg}}^{j+1}$ and $\boldsymbol{f}^{j+2|j+1} = \boldsymbol{f}_{\text{sup}}^{j+1}$. The first element $\boldsymbol{\varphi}^{j+1|j+1}$ of $\mathcal{S}_\varphi^{j+1}$ will define the next $((j+1)$-th) step of the humanoid.

Note that, while the footstep planner will make use of a fixed map $\mathcal{M}_z^j$ during the time interval $[t_s^j, t_s^{j+1})$, 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 $\mathcal{M}_z^{j+1}$ 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.

3.4.2 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 [28], 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_s^j of the generic j-th invocation of the footstep planner, provides it with a copy $\mathcal{M}_z^j$ 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.

3.4.3 Sensor-based Footstep Planning

This module consists in a sensor-based version of the footstep planner proposed in Sect. 3.3.2 which works using the knowledge about the environment incrementally acquired by the robot during motion.

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. 3.3.2, 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

$$\zeta(v, v') = \gamma(\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}'_{\text{swg}}) + \gamma(\boldsymbol{f}_{\text{sup}}, \boldsymbol{f}'_{\text{sup}}) \tag{3.23}$$

where $\gamma(\cdot)$ is the footstep-to-footstep metric defined in (3.4). 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 $\boldsymbol{f}_{\text{sup}}^{j+1}$. ◀

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[13] 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 $\bar{\kappa}$. ◀

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 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. 3.3.2 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.

[13] 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.

- 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 (3.8). 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

$$v^* = \underset{v \in \mathcal{V}_{\text{fron}}}{\text{argmin}}\ c(v) + g(v) \tag{3.24}$$

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. 3.4.5.

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 $\mathcal{G}$.

3.4.4 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 [29]. 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.

3.4.5 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 Sects. 3.3.2 and 3.3.5,

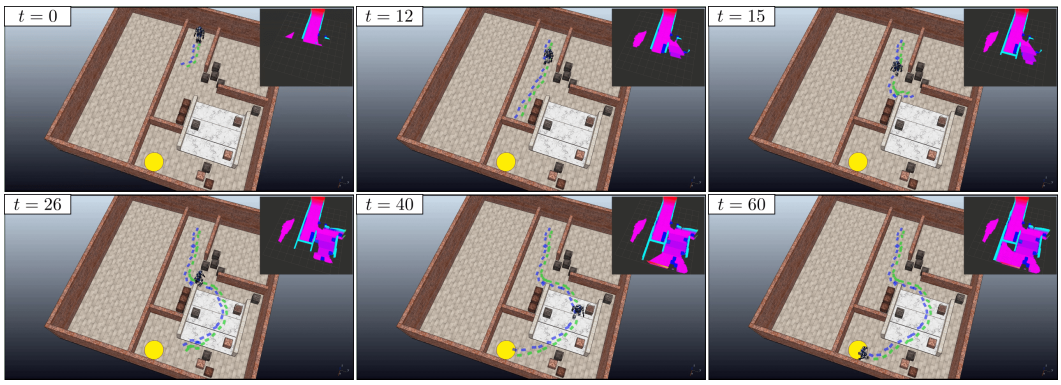

Figure 3.15. The on-line footstep planner in the scenario *Corridor* minimizing the number of steps. Here the planner finds a footstep sequence of 54 steps.

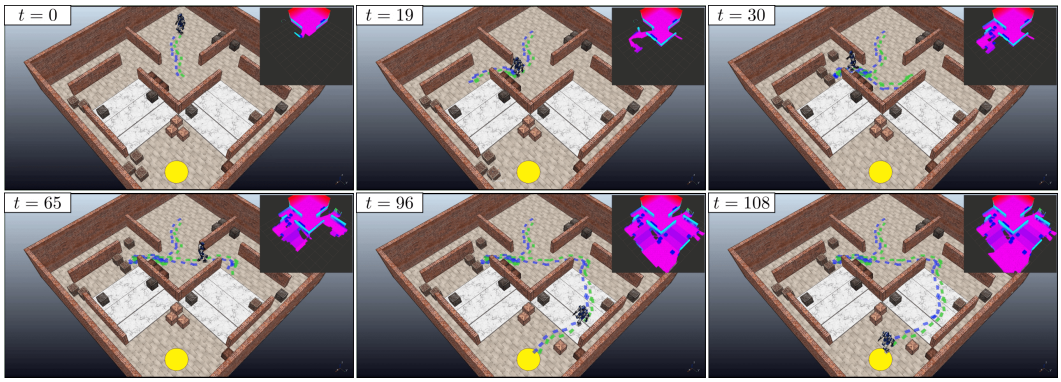

Figure 3.16. The on-line footstep planner in the environment *Maze* minimizing the number of steps. The environment is dynamic, namely the elevation map can be changed by moving obstacles around. Here the planner finds a footstep sequence of 106 steps.

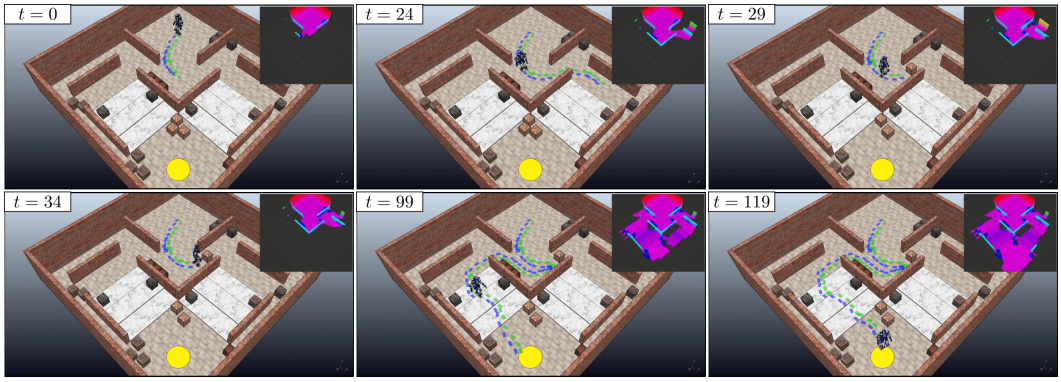

Figure 3.17. The on-line footstep planner in the dynamic environment *Maze* minimizing the number of steps. Here the planner finds a footstep sequence of 103 steps.

with the only exception of setting $k_\gamma = 1$ in (3.4) when used in (3.23), $\bar{n} = 0.9$, $\bar{z} = 0.02$ 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 3.15 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 3.16 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 3.17 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 included in the accompanying video.

3.5 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 [6], which uses a weighted A^* algorithm to search for optimal footstep sequences on uneven ground[14]. 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 [6] 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. 3.3.2. As for the optimality criterion, we aim to minimize the number of footsteps, corresponding to an edge cost given by (3.5). 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 3.4 collects the results obtained for $w = 5$. These results should be compared to those in Table 3.1, 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 [6], its performance may degrade severely in the scenarios that require even mild amounts of backtracking.

As for the gait generation component, there are several alternatives to our method that employ MPC on reduced-order models. The advantage of the proposed approach lies in the introduction of the stability constraint, which allows to prove rigorously recursive feasibility and internal stability of the MPC scheme. The effectiveness of this approach was extensively analyzed in [20], by means of comparisons to state-of-the-art schemes. What we proposed in this paper is an extension accounting for the fact that the CoM height changes. Based on the additional theoretical results which we proved (namely, Props. 1 and 2), we can claim that our 3D gait generation

[14]The algorithm in [6] 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 3.4. Performance of the off-line weighted A^* footstep planner when minimizing the number of steps ($w = 5.0$).

retains the same properties of the original 2D scheme.

3.6 Conclusions

In this paper, we addressed the problem of motion generation for a humanoid robot that must reach a certain goal region walking in an environment consisting of horizontal patches located at different heights, called *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 w.r.t. 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.

3.6.1 Pseudocode

We report here a pseudocode description of the footstep planners presented in Sections 3.3.2 and 3.4.3, along with the relevant procedures.

Algorithm 1: FootstepPlanner

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

$v^{\text{ini}} \leftarrow (\boldsymbol{f}_{\text{swg}}^{\text{ini}}, \boldsymbol{f}_{\text{sup}}^{\text{ini}})$;
AddVertex($\mathcal{T}$, $\emptyset$, v^{ini}, $\emptyset$);
ExpandTree($\mathcal{T}$, ΔT, $\mathcal{M}_z$);
$\mathcal{P}^* \leftarrow$ RetrieveBestPlan($\mathcal{T}, \mathcal{G}$);
return $\mathcal{P}^*$;

Procedure 1: ExpandTree

Input: $\mathcal{T}, \Delta T, \mathcal{M}_z$
Output: none

while not TimeExpired(ΔT) **do**
  $\boldsymbol{p}_{xy}^{\text{rand}} \leftarrow$ SamplePoint();
  $v^{\text{near}} \leftarrow$ NearestVertex($\mathcal{T}, \boldsymbol{p}_{xy}^{\text{rand}}$);
  $\boldsymbol{f}_{\text{sup}}^{\text{cand}} \leftarrow$ GenerateCandidateFootstep($\boldsymbol{f}_{\text{sup}}^{\text{near}}, U, \mathcal{M}_z$);
  if R1($\boldsymbol{f}_{\text{sup}}^{\text{cand}}$) **and** R2($\boldsymbol{f}_{\text{sup}}^{\text{cand}}, \boldsymbol{f}_{\text{sup}}^{\text{near}}$) **then**
    $\boldsymbol{\varphi}^{\text{near}} \leftarrow$ SwingTrajectoryEngine($\boldsymbol{f}_{\text{swg}}^{\text{near}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}}$);
    if R3($\boldsymbol{\varphi}^{\text{near}}, (\boldsymbol{f}_{\text{sup}}^{\text{near}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$) **then**
      $v^{\text{cand}} \leftarrow (\boldsymbol{f}_{\text{sup}}^{\text{near}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$;
      $\mathcal{N} \leftarrow$ Neighbors($\mathcal{T}, v^{\text{cand}}$);
      $(v^{\text{min}}, \boldsymbol{\varphi}^{\text{min}}) \leftarrow$ ChooseParent($\mathcal{T}, \mathcal{N}, v^{\text{near}}, v^{\text{cand}}, \boldsymbol{\varphi}^{\text{near}}$);
      $v^{\text{new}} \leftarrow (\boldsymbol{f}_{\text{sup}}^{\text{min}}, \boldsymbol{f}_{\text{sup}}^{\text{cand}})$;
      AddVertex($\mathcal{T}, v^{\text{min}}, v^{\text{new}}, \boldsymbol{\varphi}^{\text{min}}$);
      ReWire($\mathcal{T}, \mathcal{N}, v^{\text{new}}$);
    end
  end
end
return;

3.6.2 Proof of Prop. 1

Starting from the initial condition $\boldsymbol{p}_u(t_k)$ described by (3.12), the unstable coordinate $\boldsymbol{p}_u$ evolves as

$$\boldsymbol{p}_u(t) = \eta \int_t^{\infty} e^{-\eta(\tau - t)} \boldsymbol{p}_z(\tau) d\tau + \frac{\boldsymbol{g}}{\eta^2}. \tag{3.25}$$

Integrating 3.25 by parts gives

$$\boldsymbol{p}_u(t) - \boldsymbol{p}_z(t) - \frac{\boldsymbol{g}}{\eta^2} = \int_t^{\infty} e^{-\eta(\tau - t)} \dot{\boldsymbol{p}}_z d\tau. \tag{3.26}$$

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 CollisionFree($\boldsymbol{\varphi}^a$) **then**
    return $\boldsymbol{\varphi}^a$;
  end
  $h \leftarrow h + \Delta h$;
end
return $\emptyset$;

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 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 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 ($v^{\min}$, $\boldsymbol{\varphi}^{\min}$);

Procedure 4: ReWire

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

Output: none

foreach $v' \in \mathcal{N}$ **do**
  if R2($\boldsymbol{f}'_{\text{sup}}, \boldsymbol{f}^{\text{new}}_{\text{sup}}$) **then**
    $\boldsymbol{\varphi}^{\text{new}} \leftarrow$ SwingTrajectoryEngine($\boldsymbol{f}^{\text{new}}_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}$);
    if $R3(\boldsymbol{\varphi}^{\text{new}}, (\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}))$ **and** $c(v^{\text{new}}) + l(v^{\text{new}}, v') < c(v')$ **then**
      UpdateVertex($\mathcal{T}, v', (\boldsymbol{f}^{\text{new}}_{\text{sup}}, \boldsymbol{f}'_{\text{sup}})$);
      UpdateEdge($\mathcal{T}, v^{\text{new}}, v', \boldsymbol{\varphi}^{\text{new}}$);
      $\mathcal{V}_{\text{child}} \leftarrow$ ChildVertexes($\mathcal{T}, v'$);
      foreach $v'' \in \mathcal{V}_{\text{child}}$ **do**
        $\boldsymbol{\varphi}' \leftarrow$ SwingTrajectoryEngine($\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}''_{\text{sup}}$);
        if $\boldsymbol{\varphi}' \neq \emptyset$ **then**
          UpdateEdge($\mathcal{T}, v', v'', \boldsymbol{\varphi}'$);
        else
          RemoveSubtree($\mathcal{T}, v''$);
        end
      end
    end
  end
end
return;

Algorithm 2: SensorBasedFootstepPlanner

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

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

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

Procedure 5: InitializeTree

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

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

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

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;

Since by hypothesis the ZMP velocity $\dot{\boldsymbol{p}}_z$ is component-wise bounded, i.e., $|\dot{x}_z| \leq v^{\max}$, $|\dot{y}_z| \leq v^{\max}$, $|\dot{z}_z| \leq v^{\max}$, then the following upper bound is obtained

$$\boldsymbol{p}_u(t) - \boldsymbol{p}_z(t) - \frac{\boldsymbol{g}}{\eta^2} \leq v^{\max}\mathbf{1}\int_t^\infty e^{-\eta(\tau-t)}d\tau,$$

along with a similar lower bound. The two bounds can be combined in the following expression

$$-\frac{v^{\max}}{\eta}\mathbf{1} \leq \boldsymbol{p}_u(t) - \boldsymbol{p}_z(t) - \frac{\boldsymbol{g}}{\eta^2} \leq \frac{v^{\max}}{\eta}\mathbf{1}. \tag{3.27}$$

Define now the coordinate $\boldsymbol{p}_s = \boldsymbol{p}_c - \dot{\boldsymbol{p}}_c/\eta$. This coordinate pertains to the stable eigenvalue of system (3.10), and its dynamics can be expressed as

$$\dot{\boldsymbol{p}}_s = \eta(-\boldsymbol{p}_s + \boldsymbol{p}_z) + \frac{\boldsymbol{g}}{\eta}.$$

Starting from an initial time t_0, the coordinate $\boldsymbol{p}_s$ evolves as

$$\boldsymbol{p}_s(t) = \boldsymbol{p}_s(t_0)e^{-\eta(t-t_0)} + \eta\int_{t_0}^t e^{-\eta(t-\tau)}\left(\boldsymbol{p}_z(\tau) + \frac{\boldsymbol{g}}{\eta^2}\right)d\tau.$$

Similarly to $\boldsymbol{p}_u$, the above expression can manipulated using integration by parts, and the resulting integral can be bounded using the hypothesis on the ZMP velocity. The resulting expression is

$$-\frac{v^{\max}}{\eta}\mathbf{1} + \boldsymbol{s}_0 \leq \boldsymbol{p}_s(t) - \boldsymbol{p}_z(t) - \frac{\boldsymbol{g}}{\eta^2} \leq \frac{v^{\max}}{\eta}\mathbf{1} + \boldsymbol{s}_0,$$

where $\boldsymbol{s}_0 = \boldsymbol{p}_s(t_0) - \boldsymbol{p}_z(t_0) - \boldsymbol{g}/\eta^2$. Since at t_0 the robot is typically in static equilibrium, corresponding to $\boldsymbol{p}_c(t_0) = \boldsymbol{p}_z(t_0) + \boldsymbol{g}/\eta^2$ and $\dot{\boldsymbol{p}}_c(t_0) = 0$, then $\boldsymbol{s}_0 = 0$. This leads to

$$-\frac{v^{\max}}{\eta}\mathbf{1} \leq \boldsymbol{p}_s(t) - \boldsymbol{p}_z(t) - \frac{\boldsymbol{g}}{\eta^2} \leq \frac{v^{\max}}{\eta}\mathbf{1}. \tag{3.28}$$

By noting that $2\boldsymbol{p}_c = \boldsymbol{p}_u + \boldsymbol{p}_s$, combining eqs. (3.27) and (3.28) proves the thesis.

Chapter 4

Feasibility aware plan adaptation in humanoid gait generation

Todo.

Chapter 5

Nonlinear model predictive control for steerable wheeled mobile robots

Todo.

Chapter 6

Conclusions

Todo.

Appendix A

Feasibility aware plan adaptation in humanoid gait generation

Todo.

Appendix B

Real-time nonlinear model predictive control

Todo.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

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

[25] 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.

[26] 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.

[27] Basile G. Rabbat and Henry G. Russell. Friction coefficient of steel on concrete or grout. *Journal of Structural Engineering*, 111:505–515, 1985.

[28] 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.

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