SAPIENZA
UNIVERSITÀ DI ROMA

Motion Planning Techniques for Humanoid Robots

Sapienza Università di Roma

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

Candidate

Paolo Ferrari

ID number 1203661

Thesis Advisor

Prof. Giuseppe Oriolo

Thesis defended on July 13, 2021
in front of a Board of Examiners composed by:

Prof. Giuseppe Baselli (chairman)

Prof. Stefano Panzieri

Prof. Fabio Tardella

Motion Planning Techniques for Humanoid Robots
Ph.D. thesis. Sapienza – University of Rome

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

Version: July 8, 2021

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

Abstract

Thanks to their human-like structure, humanoid robots have the potential for accomplishing complex tasks requiring legged locomotion and/or dual-arm manipulation in both structured and unstructured environments. To successfully fulfil such tasks, appropriate humanoid motions must be generated. Planning these motions is particularly challenging for humanoid robots because of their peculiar characteristics. First, their high number of degrees of freedom makes planning computationally expensive. Second, they can displace their base only through stepping or acyclic multi-contact motions. Third, they must maintain balance at all times.

This thesis addresses the motion planning problem for humanoid robots in various contexts. We start by considering the general problem of planning whole-body motions for a humanoid robot that must execute a task implicitly requiring locomotion in an environment populated by static obstacles. For this problem we propose a complete framework that can incorporate tasks of different nature (i.e., navigation, reaching, manipulation and visual tasks) for planning both in case of known and unknown environments.

One of the advantages of humanoids is the possibility of moving through complex environments by stepping over or onto obstacles. To this end, we propose an integrated method for planning and executing humanoid motions on uneven ground. It is composed by two modules: an offline footstep planner and an online gait generator. For the first module we propose two possible randomized strategies that can efficiently compute feasible and optimal footstep plans, respectively.

In many practical applications, it might be allowed to abandon the task in favor of collision avoidance. For cases in which the robot is assigned a soft task of this type, we present an opportunistic strategy for planning motions that, differently from other approaches, allow the robot to perform the assigned task for as long as possible, and deviate from it only when strictly needed to avoid a collision. The method is first discussed with regard to a generic free-flying robot, and later extended to the case of humanoid robots.

More complex tasks that a humanoid robot can potentially fulfil require to sequentially establish with the environment multiple contacts involving not only the feet as in basic biped locomotion. For this problem we propose a multi-contact motion planner that thanks to its randomized nature avoids any kind of precomputation or heuristics design that are usually required with existing search-based techniques.

Finally, we consider the problem of safe coexistence between human and humanoids. In this context, reactive planning capabilities are essential. We describe a complete framework for the safe deployment of humanoid robots in environments containing humans, where several safety behaviors are activated and deactivated through a state machine according to information coming from the robot sensors.

Ringraziamenti

Il percorso che ha portato alla produzione di questa tesi è stato indubbiamente faticoso e stressante ma mi ha concesso il privilegio di poter quotidianamente imparare ed immaginare. Ciò è stato possibile solo grazie a tutti coloro che in qualche modo a questo percorso hanno partecipato e desidero perciò ringraziarli.

Il mio primo ringraziamento va al Prof. Giuseppe Oriolo che mi ha guidato in questi anni dandomi continuamente supporto e fiducia. Tutto ciò che ho potuto imparare dalla sua enorme competenza, inesauribile curiosità e gusto per un'estetica sempre funzionale al risultato sarà il bagaglio che mi porterò sempre dietro.

Ringrazio il Prof. Leonardo Lanari per aver saputo incontrare il mio approccio originariamente più algoritmico e meno controllista. Se i miei orizzonti si sono ampliati è anche grazie a tutto ciò che ho potuto apprendere lavorando con lui.

Grazie a tutti i membri del Laboratorio di Robotica del DIAG, miei colleghi e amici; grazie a loro ho vissuto ogni giorno in un ambiente sereno e stimolante. Ringrazio particolarmente tutti coloro che con me hanno collaborato: Marco Cognetti, Nicola Scianca, Daniele De Simone, Massimo Cefalo e Michele Cipriano.

Ringrazio il gruppo di Humanoids & Human Centered Mechatronics guidato dal Prof. Nikolaos Tsagarakis presso l'Istituto Italiano di Tecnologia con il quale ho avuto il grande piacere di collaborare durante l'ultimo anno. Un ringraziamento speciale va a Enrico Mingo Hoffman, Luca Rossini e Francesco Ruscelli per la costante disponibilità e la capacità di coordinarsi con me a distanza, sempre con passione e determinazione.

Un enorme ringraziamento va a Brunella, ogni sfida sarebbe inaffrontabile senza averla accanto; il suo amore ha costantemente supportato i miei sforzi e la sua incondizionata pazienza mi ha sempre trasmesso sicurezza.

Grazie ai miei genitori, Antonio e Paola, per avermi insegnato che con il duro lavoro si può ottenere tutto. Grazie a Simone per essere sempre l'amabile fratello che condivide con me l'inguaribile trasporto per la scienza.

Grazie infine ai miei amici di sempre, quelli che parlano il dialetto in cui ogni mia nuova idea nasce.

Contents

Chapter 1

Introduction

The long-term challenge for robotics is the development of systems that can assist, or even substitute, humans in repetitive, tiring and dangerous activities.

Early robots have been introduced in industrial settings and consist in fixed-base manipulators (see Fig. 1.1). The simplest mechanical structure of these robots consists of a single kinematic chain, i.e., a sequence of links interconnected by means of joints. Each joint, providing the robot with a degree of freedom, acts as an articulation. For this reason, robots with one or more kinematic chains are also referred to as articulated robots.

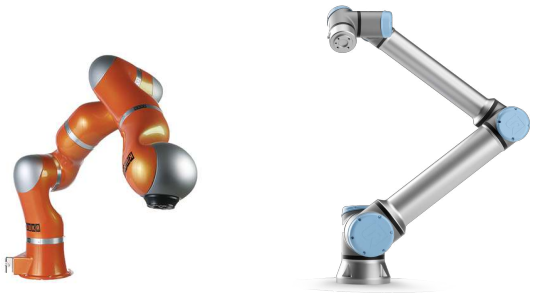

Figure 1.1. Examples of manipulators: LWR by KUKA, and UR10 by Universal Robots. Retrieved from `https://www.coboticsworld.com/`, and `https://cobots.ie/`, respectively.

More complex structures are represented by mobile manipulators. They consist of one or more manipulators mounted on a mobile base. The latter endows the robot with locomotion capabilities, thanks to which its workspace becomes potentially unlimited. When locomotion is constrained to the ground, the mobile base of the robot can be wheeled or legged. Motion with respect to the ground is generated by a set of, respectively, actuated wheels and kinematic chains acting as lower limbs.

As a direct consequence of their structure, wheeled robots (see Fig. 1.2) are particularly tailored to both industrial and service applications in structured environments where the ground is completely flat. On the other hand, legged robots are suitable for applications in both structured and unstructured environments. For example, climbing and descending stairs in human environments (such as industries and houses) or moving across rough terrains (such as outdoor and disaster scenarios) is only possible for legged robots.

Figure 1.2. Examples of mobile manipulators: PR2 by Willow Garage, and TIAGo by PAL Robotics. Retrieved from `https://robots.ieee.org/robots/pr2/`, and `https://pal-robotics.com/robots/tiago/`, respectively.

A special category of legged robots is that of *humanoid robots* (see Fig. 1.3). The structure of a humanoid robot is inspired by the human body and consists of two legs, two arms, a torso and a head. Typically the end-effectors of the kinematic chains composing legs and arms are similar to, respectively, human feet and hands. The head is usually equipped with vision sensors. Thanks to their mechanical structure, humanoids have the potential for effectively perform complex tasks, requiring both legged locomotion and dual-arm manipulation. Moreover, their anthropomorphic structure makes them more easily acceptable to humans, with whom we would like them to coexist and collaborate.

Figure 1.3. Examples of humanoid robots: ASIMO by Honda, REEM-C by PAL Robotics, and HRP-4 by Kawada Robotics. Retrieved from `https://robots.ieee.org/robots/asimo/`, `https://pal-robotics.com/robots/reem-c/`, and `https://robots.ieee.org/robots/hrp4/`, respectively.

Whatever the specific application is, robots are expected to perform tasks in real-world scenarios that generally contain static and dynamic obstacles. Such tasks shall be simply specified by humans via high-level descriptions (e.g., 'go to that location' or 'grasp that object') and then converted into low-level descriptions that instruct the robot on how to move in order to fulfil the assigned task while avoiding collisions with obstacles. The problem of producing such conversion is known as *motion planning* [1]. Equipping a robot with a motion planning module is clearly fundamental in order to endow it with full autonomy. Despite the constantly increasing effort of the robotics community, articulated robots, and in particular humanoids, are still far from being able to autonomously plan their motions for fulfilling complex tasks in cluttered and possibly unknown environments.

This thesis addresses the motion planning problem for humanoid robots in various contexts. The rest of this thesis is organized as follows.

- Chap. 2 provides a review of the literature related to motion planning for humanoid robots.
- Chap. 3 considers the problem of planning whole-body motions for a humanoid robot that must execute a task in an environment cluttered by obstacles. We first describe a unified method for planning offline in the presence of a variety of tasks implicitly requiring locomotion, and then present two extensions for planning online in case of, respectively, time limitations and unknown environments that we introduced in [2, 3].
- Chap. 4 describes an integrated method for planning and executing humanoid motions on uneven ground. The approach we introduced in [4] is presented, along with an extension that allows to generate motions of improved quality.
- Chap. 5 presents an opportunistic strategy for planning in the presence of soft tasks, i.e., tasks that are allowed to be abandoned whenever strictly required for avoiding collisions. We proposed the method in the journal paper [5] for a free-flying robot. This chapter also proposes an extension to the case of humanoid robots.
- Chap. 6 considers the problem of planning multi-contact motions for a humanoid robot that must execute a loco-manipulation task, i.e., a task that requires to establish multiple contacts (involving both feet and hands) with the environment. Preliminary results with regard to the stand up task are shown.
- Chap. 7 describes a complete framework for the safe deployment of humanoid robots in environments containing humans, where reactive planning capabilities are needed. We proposed the described approach in the journal paper [6].
- Chap. 8 offers concluding remarks about the proposed methods and discusses possible future work.

Chapter 2

Literature review

Motion planning is an active field of research since few decades. Contributions continuously come from different disciplines, such as algorithm theory, automatic control and computational geometry, making the related literature very wide and sophisticated.

Extensive work has been done with reference to the simplest form of the motion planning problem, i.e., the *canonical problem* [7], which we describe in the following. Consider a robot that consists of a single rigid body (e.g., a mobile robot), or of a kinematic chain whose base is either fixed (e.g., a manipulator) or mobile (e.g., a mobile manipulator), and moves in a 2D or 3D workspace populated by fixed obstacles whose location and geometry is known. Moreover, assume that the robot is free-flying in its configuration space, i.e., it is not subject to kinematic constraints of any kind. Given a start configuration $\boldsymbol{q}_s$ and a goal configuration $\boldsymbol{q}_g$, the canonical motion planning problem consists in finding a path in the configuration space that drives the robot from $\boldsymbol{q}_s$ to $\boldsymbol{q}_g$ while avoiding collisions with workspace obstacles. In the particular case in which the robot is a single body moving in 2D, the canonical motion planning problem is also known as the *piano movers' problem* [8].

A large number of approaches have been presented for solving the canonical problem, and many of them have been appropriately modified for dealing with extensions of the canonical problem resulting from the elimination of one of the simplifying assumptions for planning, e.g., in the presence of moving obstacles, in unknown environments, for robots subject to nonholonomic constraints, and so on.

Early works dealing with the canonical problem (see [9, 10]) used to represent the connectivity of the free portion of the configuration space by a network of collision-free paths. To this end, they require the preliminary computation of the 'images' of the obstacles in the configuration space, which is in general an expensive procedure. This issue was completely overcame with the introduction of sampling-based motion planners, e.g., PRM (Probabilistic Roadmap) [11] and RRT (Rapidly-exploring Random Tree) [12], that relies on the basic idea of finding a finite set of collision-free configurations that adequately represent the connectivity of the configuration space by iteratively selecting random samples from it. Both PRM and RRT are probabilistically complete and particularly efficient in terms of planning times. Moreover, RRT exhibits the remarkable ability of being directly applicable to systems with differential constraints [13], and has also been extended

to an asymptotically optimal version, called RRT* [14].

All the discussed methods have been introduced in the context of *offline* planning, i.e., a complete solution to the planning problem is computed before starting motion execution, that is tailored for situations in which the geometry of the environment in completely known in advance. When this assumption is removed, *online* planning is clearly required, as the robot must discover the environment while moving. In this context, a large number of methods have been proposed such as online versions of RRT [15, 16, 17] and the artificial potential fields method [18]. The latter is a heuristic approach that builds potential fields in the configuration space in such a way the robot configuration is attracted by the goal and repelled by the obstacles; at each configuration, the anti-gradient of the total potential (sum of attractive and repulsive potential) indicates the best local direction of motion. Similar reactive strategies are represented by the navigation functions [19], which provide artificial potentials that have no local minima, and the dynamic window approach [20], which computes, at each time instant, a finite set of local state-space trajectories respecting the robot kinodynamic limits and selects for execution the one minimizing a certain cost function that accounts for the distance from the goal and from the closest obstacle. More recently, also techniques based on Model Predictive Control have been introduced for generating collision-free motions, both for fixed-base [21] and mobile manipulators [22, 23].

Although many of the techniques described above have been successfully applied to both fixed and mobile manipulators in different contexts, their direct application to humanoid robots is impossible. There are mainly three reasons for which motion planning for humanoids is particularly challenging.

- The high number of degrees of freedom, due to the large set of joints that make up the various kinematic chains, reflects in a high dimensional configuration space. Clearly, the higher the dimension of the configuration space, the harder the computation of a solution to the motion planning problem.

- A humanoid robot is not a free-flying system in its configuration space, but it can displace its base only through stepping or acyclic multi-contact motions. Such motions must be appropriately generated.

- During motion, the robot must maintain equilibrium, either static or dynamic, at all time instants. This typically constrains the trajectory of the Center of Mass (CoM) of the robot, thus reducing the portion of the configuration space in which a solution to the motion planning problem can be found.

In order to make the planning problem tractable, many approaches that rely on simplifying assumptions on either the environment or the robot geometry have been proposed. A common approach consists in assuming, at least in a first planning phase, that collisions may occur only at footstep level [24, 25, 26]. Thus, the problem reduces to finding a footstep plan, i.e., a sequence of isolated contacts with the ground. See [27] for a survey on existing footstep planners. The dual approach consists in using a simplified occupancy volume of the humanoid, e.g., a conservative bounding box, to initially find a collision-free channel between the initial and goal location. Examples of this approach can be found in [28, 29]. After this first

stage, either in case planning involves footsteps or simplified occupancy volumes, locomotion is produced using a gait generation module which typically computes a CoM trajectory of the humanoid that guarantees static or dynamic balance, i.e., the ground projection of the CoM or the Zero Moment Point (ZMP, the point where the horizontal component of the moment of the ground reaction forces becomes zero [30]) remains at all times within the support polygon of the robot, respectively. We omit explicit discussion of existing gait generators, as they are outside the scope of this thesis. However, Appendix A contains a brief overview of the specific gait generator used throughout this thesis.

Clearly, these approaches are only tailored to pure locomotion tasks, usually also referred to as navigation or walk-to tasks, that require the robot to simply move from one place to another of the workspace. In fact, such methods can not plan motions that allow a humanoid to fulfil a more complex task that requires the exploitation of the whole-body structure; for example, extending over a table to pick up an object with one hand is not possible using these methods.

In general, regardless of the specific robotic platform (fixed-base manipulators, mobile manipulators, humanoids, and so on), tasks are expressed in terms of a certain set of coordinates. These may describe quantities related to manipulation (end-effector position and/or orientation), navigation (position of a representative point of the robot, e.g., the center of mass), or perception (placement of sensors in the workspace or directly of features in sensing space, as in visual servoing). The problem of generating collision-free robot motions in the presence of task constraints is known as Task-Constrained Motion Planning (TCMP)[1]. In the literature, there exist two main classes of methods for solving the TCMP problem, i.e., optimization- and sampling-based methods.

Optimization-based methods (see [31] for a general review) cast the TCMP problem in the framework of kinematic control, also called redundancy resolution, with the possible inclusion of specific equality or inequality constraints related to the assigned task [32]. The discrete optimization technique presented in [33] is able to compute very accurate tracking; however, avoidance of workspace obstacles is not considered. In any case, it should be kept in mind that, independently of the specific version, kinematic control is a greedy strategy whose optimization capabilities are inherently local; as a consequence, it can work occasionally but never guarantee completeness (finding a solution whenever one exists).

Sampling-based methods for solving the TCMP problem consists in variants of the basic PRM and RRT methods discussed above. Typically, they use a mechanism for projecting configuration space samples on the submanifold where the task constraint is satisfied; see [34] for a review and [35, 36, 37, 38] for specific techniques aimed at manipulation planning. These methods generally provide probabilistic completeness, but suffer from the limitation that configuration space samples are connected by local paths lying outside the constrained manifold. To improve task tracking accuracy, it is necessary to use a more dense sampling, typically leading to a dramatic increase of the time needed to compute a plan. For complex problems, this approach can turn

[1] In order for a TCMP problem to be well-posed, the robot must be redundant w.r.t. the assigned task (i.e., the number of robot degrees of freedom is larger than that of task components), otherwise at most only one collision-free motion exists that realizes the task; such motion can be found using, e.g., standard pseudoinverse-based inverse kinematics, without the need of a motion planner.

out to be very inefficient and impractical. Such issue was overcame in [39], which introduced a sampling-based approach for solving the TCMP problem that avoids the need for a projection mechanism thanks to a control-based motion generation scheme; as a consequence, it becomes possible to guarantee continuous satisfaction of the task constraint with arbitrary precision.

All the mentioned methods address the TCMP problem for fixed-base or wheeled mobile manipulators. Their direct extension to the case of humanoid robots is impossible, because of the peculiar characteristics of these systems discussed above. Most existing approaches for solving TCMP problems for humanoids are devised for the specific case of manipulation tasks expressed as a desired path or trajectory for one of the hands. In [40], an RRT-connect [41] exploits a precomputed catalogue of statically balanced configurations to grow a tree in the task-constrained configuration space. Similar techniques are presented in [42, 43, 44]. These methods are designed for tasks which do not require stepping, i.e., tasks that can be accomplished by the humanoid while keeping both feet fixed on the ground and the projection of the CoM inside the support polygon. Simultaneous locomotion and manipulation is achieved in [45] using a two-stages approach: first, a collision-free path, that guarantees both static equilibrium and task execution, is computed for a sliding-feet robot model using a RRT-based algorithm; second, such path is converted into a dynamically balanced motion using the preview controller proposed in [46]. A more recent work [47] uses a somewhat dual approach that first finds a dynamically balanced locomotion plan, and then computes a manipulation plan that is admissible w.r.t. both the latter and the assigned task. The limitation of decoupled approaches is that the plan obtained in the first stage might be invalidated in the second, requiring some form of reshaping or even a new query to the first stage.

The most challenging tasks that, unlike wheeled robots, humanoids can potentially fulfil require motions involving multiple contacts between links of the robot (e.g., feet, hands, knees) and the environment. In general, these contacts are not coplanar, i.e., the contact points do not necessarily lie on a common plane. Typical examples of these tasks are climbing a ladder or crawling under a table. Generation of this type of motions is generally referred to as multi-contact motion planning. A recent review on the topic is presented in [48]. Similarly to the case of pure biped locomotion, the multi-contact motion planning problem is usually decoupled in two phases. In the first phase a discrete sequence of contact combinations is computed either via graph [49] or best-first search [50, 51]. In the second phase a continuous whole-body motion that guarantees balance throughout the above sequence is found. Being the contacts non-coplanar, the simple (both static and dynamic) balance condition involving the concept of support polygon for the case of biped locomotion is clearly not applicable; thus, more expressive models, such as the centroidal dynamics model [52], are used in the second phase.

Most of the techniques for planning the motions of a humanoid robot discussed so far are clearly suited for the case of static or moderately changing environments. In highly dynamic (and possibly unknown) environments, offline approaches can not be used and online approaches that are available for fixed-base and wheeled mobile manipulators do not directly apply to humanoids. In the presence of dynamic obstacles, such as moving humans, it is essential that the reaction time, i.e., the time between the detection of a danger and the execution of an appropriate action, is

as small as possible; thus, real-time planning capabilities are required for the robot safe operation is such contexts. Existing methods deal with such motion planning problem by employing vision-based rapid replanning of footstep placement [53, 54], incorporating collision avoidance constraints in MPC schemes [55, 56], or making use of closed-form expressions to quickly generate evasion maneuvers in case of immediate danger [57]. Closely related problems are push recovery and emergency stop (e.g., see [58, 59]) which also involve stepping and balance.

Chapter 3

Whole-body motion planning

Humanoid robots have the potential for fulfilling different kind of tasks in environments cluttered by obstacles, ranging from simple navigation (e.g., go to a desired location) to more complex tasks involving end-effectors or sensors (e.g., open a door or visually track a moving target).

As already discussed in Chap. 2, the problem of generating appropriate motions for achieving these tasks is known as TCMP problem and existing approaches either do not consider tasks requiring locomotion (e.g., [40, 42, 43]) or rely on some sort of separation between locomotion and task execution (e.g., [45, 47]), thus failing to exploit the rich humanoid motion capabilities.

A completely different approach has been introduced in [60]. It consists of a randomized planner that builds a solution by concatenating whole-body motions. Each whole-body motion realizes a CoM movement primitive selected from a precomputed set and simultaneously accomplishes a portion of the task. The CoM primitives are representative of typical humanoid actions such as walking (static and dynamic), and can in principle include more sophisticated movements (e.g., jumping, crouching, etc). With this approach, locomotion and task execution are not separated during the planning stage, and walking emerges naturally from the solution.

All the mentioned approaches are offline techniques. In general, online planning capabilities are required mainly in two situations. The first is when the complexity of the environment is such that the computation of a complete solution requires high planning times, forcing the robot to wait before being allowed to start motion execution. The second is when the environment is not known in advance, making the computation of a complete solution even impossible. To achieve online performance, regardless of the a priori knowledge of the environment, the problem is usually reduced to repeatedly finding partial footstep sequences (e.g., see [25]).

In the specific case of unknown environment, some methods rely on the idea of monitoring the workspace with off-board, fixed sensors to provide a footstep planner with the positions of the humanoid and obstacles (e.g., [53, 61]). Off-board sensing requires an appropriate setting that is difficult, or even impossible, in unstructured environments. Thus, on-board sensing is clearly preferable. This is exploited, for example, for detecting planar surfaces that define safe regions where the robot can step onto (as in [62], where binocular stereo-vision is used), or by representing the environment through a heightmap (using data provided by, e.g., a monocular on-

board camera as in [63], or a pivoting laser scanner mounted on the humanoid torso as in [64, 65]). Planning only at footsteps level implicitly assumes that collisions can occur only at robot soles, which is not the case for most real-world scenarios.

Only few methods have been proposed that consider the 3D structure of the unknown environment, for example, modeling it via a 3D occupancy grid. However, during the planning stage, collisions are often checked using, e.g., bounding boxes [66], ground projections of the 3D map and circular robot models [54], or inverse heightmaps precomputed for each possible robot action [67]. In general, these approaches prevent the robot from passing through a narrow passage or below a low obstacle.

In this chapter we present a complete method for planning whole-body motions for humanoid robots that must execute tasks implicitly requiring stepping in environments cluttered by obstacles. In particular, building on the CoM primitives-based technique introduced in [60], we propose:

- a unified offline planner for the case of known environment in the presence of different kinds of tasks (essentially an extension of the method in [60]), namely navigation, reaching, manipulation and visual tasks;
- an anytime planning framework for the case of time limitations, i.e., when the robot must start moving after a given time budget that, in general, is not sufficient for computing a complete solution;
- a sensor-based planning framework for the case of unknown environment, i.e., when the robot must plan its motions according to incrementally acquired information.

The use of CoM movement primitives allows to exploit the whole-body structure of the humanoid. Moreover, both in the case of known and unknown environment, our method takes into account the 3D structure of both the robot and the workspace. As a result, the generated whole-body motions are guaranteed to be collision-free, balanced and task-oriented, without the need of any form of reshaping, differently from other existing approaches.

This chapter is organized as follows. In Sect. 3.1 we formally describe the motion planning problem, the considered tasks and the concept of CoM movement primitives. Sects. 3.2-3.4 describe the proposed offline planner and its extension to the anytime and sensor-based frameworks, together with simulation results obtained with the NAO humanoid. Sect. 3.5 provides some concluding remarks.

3.1 Core problem and approach

In this section, we first introduce a suitable motion model for a humanoid robot (Sect. 3.1.1). Then, in Sect. 3.1.2, we formally describe the general problem of planning whole-body motions for a humanoid robot that must execute a certain task in an environment containing obstacles, and discuss the nature of the considered tasks. The planning strategy at the core of the framework presented in this chapter relies on the concept of *CoM movement primitives* which represent elementary humanoid motions (e.g., stepping, crouching, jumping, and so on). Sect. 3.1.3 provides a detailed presentation of such concept.

3.1.1 Humanoid motion model

The configuration of a humanoid robot is fully described by the n-vector of its joint angles and the pose (position and orientation) of a reference frame attached to one of its links. In view of the use of CoM movement primitives, a convenient choice within our planning framework, as initially introduced in [60], consists in attaching such frame to the CoM and assuming it oriented as the torso[1]. A generic configuration of the humanoid is then defined as

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}_{\mathrm{CoM}} \\ \boldsymbol{q}_{\mathrm{jnt}} \end{pmatrix},$$

where $\boldsymbol{q}_{\mathrm{CoM}} \in SE(3)$ is the pose of the CoM reference frame and $\boldsymbol{q}_{\mathrm{jnt}} \in \mathcal{C}_{\mathrm{jnt}}$ is the n-vector of the joint angles. For illustration, we will express the orientation of a given reference frame as a unit norm quaternion, although other choices are also possible. Then, a generic pose of a certain frame will be specified by a 7-vector, where the upper 3- and lower 4-subvectors describe, respectively, the position and orientation. Note that, 6 of these 7 components are independent. With this choice, the humanoid configuration space $\mathcal{C}$ is $SE(3) \times \mathcal{C}_{\mathrm{jnt}}$ and its dimension is $n + 6$.

As anticipated, within our planning framework, the CoM movements are generated by patching CoM subtrajectories that are extracted from a catalogue of CoM movement primitives, that is supposed to be precomputed and available to the planner. Each primitive has a given duration, specifies a reference trajectory for the CoM, and possibly also for other points of the humanoid (e.g., the swing foot for a stepping motion). Once a primitive has been selected, the robot joint motion will be chosen so as to execute it while satisfying other requirements.

This planning approach is reflected in the following motion model

$$\boldsymbol{q}_{\mathrm{CoM}}(t) = \boldsymbol{q}^k_{\mathrm{CoM}} \oplus \boldsymbol{A}(\boldsymbol{q}^k_{\mathrm{CoM}})\, \boldsymbol{u}^k_{\mathrm{CoM}}(t) \tag{3.1}$$

$$\dot{\boldsymbol{q}}_{\mathrm{jnt}}(t) = \boldsymbol{v}_{\mathrm{jnt}}(t); \tag{3.2}$$

that describes the motion of the robot in the time interval $[t_k, t_k + T_k]$ according to a certain CoM movement primitive of duration T_k. In eqs. (3.1–3.2):

- $\boldsymbol{q}^k_{\mathrm{CoM}} = \boldsymbol{q}_{\mathrm{CoM}}(t_k)$ is the pose of the CoM frame at t_k.
- $\boldsymbol{u}^k_{\mathrm{CoM}}(t)$ is the pose displacement of the CoM frame at t relative to the pose at t_k, as specified by the primitive.
- $\boldsymbol{A}(\boldsymbol{q}^k_{\mathrm{CoM}}) = \begin{pmatrix} \boldsymbol{R}(\boldsymbol{q}^k_{\mathrm{CoM}}) & \boldsymbol{0}_{3\times 4} \\ \boldsymbol{0}_{4\times 3} & \boldsymbol{I}_{4\times 4} \end{pmatrix}$ is the transformation matrix between the CoM frame at t_k and the world frame, with $\boldsymbol{R}(\boldsymbol{q}^k_{\mathrm{CoM}})$ the rotation matrix that rotates the position components of $\boldsymbol{u}^k_{\mathrm{CoM}}(t)$ into the world frame.
- $\oplus$ is an operator that composes two poses. Given two poses $\boldsymbol{q}^{\mathrm{a}}_{\mathrm{CoM}}$ and $\boldsymbol{q}^{\mathrm{b}}_{\mathrm{CoM}}$, it returns a pose $\boldsymbol{q}^{\mathrm{c}}_{\mathrm{CoM}}$ where the upper 3- and the lower 4-subvectors are obtained through, respectively, the sum of the two upper 3-subvectors and the quaternion product of the two lower 4-subvectors of $\boldsymbol{q}^{\mathrm{a}}_{\mathrm{CoM}}$ and $\boldsymbol{q}^{\mathrm{b}}_{\mathrm{CoM}}$.

[1] The use of the torso orientation is due to the fact that the CoM, being a point, does not provide any information regarding the orientation.

- $\boldsymbol{v}_{\text{jnt}}(t)$ is the n-vector of joint velocity commands.

The motion model in (3.1-3.2) is hybrid. In fact, eq. (3.1) is algebraic, as it describes the CoM motion realizing the primitive adopted in the time interval $[t_k, t_k + T_k]$, while eq. (3.2) is differential, as it describes the evolution of the joint variables which change instantaneously to realize the CoM motion and other tasks. Clearly, the CoM motion and the joint velocities are not independent in eqs. (3.1–3.2), because in the whole time interval $[t_k, t_k + T_k]$, $\boldsymbol{v}_{\text{jnt}}(t)$ must be compatible with the chosen primitive.

3.1.2 Task-oriented planning

The motion planning problem considered in this chapter consists to find an appropriate whole-body motion of the humanoid over a time interval $[t_{\text{ini}}, t_{\text{fin}}]$ that realizes an assigned task in an environment populated by static obstacles, while respecting the robot kinematic limits, guaranteeing equilibrium at all time instants, and avoiding collisions. In Sects. 3.2-3.3, we will assume that the geometry of the environment is known in advance, while we will remove this assumption in Sect. 3.4.

Tasks may be of different nature, depending on the specific application, and in general can involve both end-effectors and sensors. We will consider that the robot can be assigned one of the four different tasks described in the following.

T1 *Navigation task.* The robot must bring the midpoint between the feet inside a desired circular region centered at a given point $\boldsymbol{y}_N^*$, and having radius r_N^*. This task simply requires the robot to move from one place of the environment to another.

T2 *Reaching task.* The robot must bring a specified hand to a desired set-point $\boldsymbol{y}_R^*$. For example, such set-point may represent the position of an object that must be grasped.

T3 *Manipulation task.* The robot must execute a desired trajectory $\boldsymbol{y}_M^*(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, with a specified hand. For example, such trajectory may encode a cutting or welding operation in an industrial application, or a simple door opening in a service application.

T4 *Visual task.* The robot must track with a head-mounted camera a 3D point P that moves along a known trajectory $\boldsymbol{y}_P^*(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$. For example, point P may be representative of an object whose motion has to be monitored by the robot.

Note that, each of these tasks directly or indirectly constrain the motion of a frame attached to a specific point of the robot. Collect in a vector $\boldsymbol{y}$ the coordinates of such frame that are of interest for the assigned task. In particular, $\boldsymbol{y}$ will contain the position of the midpoint between the feet (for T1), the position and/or orientation of the chosen hand (for T2-T3), or the pose of the camera (for T4). Coordinates $\boldsymbol{y}$ are related to configuration coordinates $\boldsymbol{q}$ by a forward kinematic map $\boldsymbol{y} = \boldsymbol{k}(\boldsymbol{q})$.

Consider that the robot is assigned one of the tasks in the above list. We emphasize that, in general, all these tasks implicitly require locomotion as they can

not be completed without stepping. A solution to the planning problem consists of a configuration space trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, that satisfies the following four requirements:

R1 Collisions with workspace obstacles and self-collisions are avoided.

R2 Position and velocity limits on the robot joints, respectively in the form $\boldsymbol{q}_{\min} < \boldsymbol{q}_{\text{jnt}}(t) < \boldsymbol{q}_{\max}$ and $\boldsymbol{v}_{\min} < \boldsymbol{v}_{\text{jnt}}(t) < \boldsymbol{v}_{\max}$ are satisfied.

R3 The robot is in equilibrium, either static or dynamic, at all time instants.

R4 The assigned task is realized.

Note that, in case the robot is assigned T3 or T4, the final time instant t_{fin} is automatically determined by the given trajectory $\boldsymbol{y}^*_M(t)$ or $\boldsymbol{y}^*_P(t)$, respectively. Otherwise, t_{fin} will be a byproduct of the planner.

In the following, we will call *feasible* the motions that satisfy requirements R1-R3. For a configuration space trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, fulfilment of requirement R4 must be evaluated with respect to the assigned task. A navigation task (T1) is realized if the desired region is reached at a finite time instant t_{fin}, i.e., $\|\boldsymbol{y}(t_{\text{fin}}) - \boldsymbol{y}^*_N\| < r^*_N$. Similarly, a reaching task (T2) is realized if the desired set-point is reached at a finite time instant t_{fin}, i.e., $\boldsymbol{y}(t_{\text{fin}}) = \boldsymbol{y}^*_R$. A manipulation task (T3) is realized if the desired trajectory is exactly executed at all time instants, i.e., $\boldsymbol{y}(t) = \boldsymbol{y}^*_M(t)$, $\forall t \in [t_{\text{ini}}, t_{\text{fin}}]$. For visual tasks we adopt an Image-Based Visual Servoing (IBVS) formulation, i.e., in order to track the point P, the robot must keep always the point feature associated to point P at the center of the image plane. Such condition implicitly requires that the moving point P is occlusion-free at each time instant, i.e., the line joining the origin of the camera frame and P does not pass through any obstacle. At a generic time instant t, let $\boldsymbol{f}(t) = (f_u, f_v)^T$ be the point feature in the image plane, with $f_u = \lambda \frac{X}{Z}$ and $f_v = \lambda \frac{Y}{Z}$, where (X, Y, Z) are the coordinates of point P at $\boldsymbol{y}^*_P(t)$ in the camera frame, whose pose is $\boldsymbol{y}(t)$, and λ is the focal length of the camera. Then, a visual task (T4) is realized if $\boldsymbol{f}(t) = 0$ and the line joining the origin of the camera frame and P does not intersect any obstacle, $\forall t \in [t_{\text{ini}}, t_{\text{fin}}]$.

For sake of illustration, for manipulation and visual tasks, we assumed that the robot starts from an initial configuration $\boldsymbol{q}_{\text{ini}}$ such that the task is realized at the initial time instant, i.e., $\boldsymbol{y}(t_{\text{ini}}) = \boldsymbol{y}^*_M(t_{\text{ini}})$ and $\boldsymbol{f}(t_{\text{ini}}) = 0$, respectively. In general, R4 can be relaxed so as to require the convergence to the assigned manipulation or visual task, i.e., $\lim_{t\to\infty}(\boldsymbol{y}(t) - \boldsymbol{y}^*_M(t)) = 0$ and $\lim_{t\to\infty} \boldsymbol{f}(t) = 0$, respectively.

3.1.3 CoM movement primitives

CoM movement primitives are representative of elementary humanoid motions, such as static and dynamic steps, crouching, jumping, and so on (see Fig. 3.1). A CoM movement primitive, applied from a certain configuration $\boldsymbol{q}_k = \boldsymbol{q}(t_k)$, describes the history $\boldsymbol{u}^k_{\text{CoM}}(t)$ of the pose displacement of the CoM frame relative to the pose at t_k for each time instant t in the interval $[t_k, t_k + T_k]$ having a fixed duration T_k. For compactness, in the following $\boldsymbol{u}^k_{\text{CoM}}$ will denote such history and indicate the corresponding primitive.

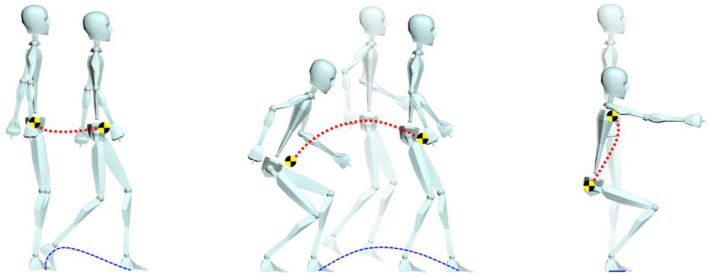

Figure 3.1. Examples of CoM movement primitives: stepping, jumping, and squatting. Courtesy of [60].

With our planning strategy, CoM movement primitives are selected from a precomputed catalogue U that usually includes *stepping* and *non-stepping* primitives. A typical U is composed as follows[2]

$$U = \{U^S_{\text{CoM}} \cup U^D_{\text{CoM}} \cup U^C_{\text{CoM}} \cup \texttt{free\_CoM}\} \,, \tag{3.3}$$

where each subset is defined as follows:

- U^S_{CoM} includes stepping primitives extracted from a *static* walking gait in which the ground projection of the CoM is contained in the humanoid support polygon at all time instants. This subset typically includes a forward step ($\boldsymbol{u}^{\text{SF}}_{\text{CoM}}$), a backward step ($\boldsymbol{u}^{\text{SB}}_{\text{CoM}}$), left ($\boldsymbol{u}^{\text{SL}}_{\text{CoM}}$) and right ($\boldsymbol{u}^{\text{SR}}_{\text{CoM}}$) steps.
- U^D_{CoM} includes stepping primitives extracted from a *dynamic* walking gait in which the ZMP is contained in the humanoid support polygon at all time instants. This subset, which is more complex than U^S_{CoM}, includes starting, cruise, and stopping steps for various directions of motion. For example, for the forward direction, we have the three steps $\boldsymbol{u}^{\text{DF,start}}_{\text{CoM}}$, $\boldsymbol{u}^{\text{DF,cruise}}_{\text{CoM}}$, and $\boldsymbol{u}^{\text{DF,stop}}_{\text{CoM}}$.
- U^C_{CoM} is composed by two different subsets of crouching primitives. The first subset includes two non-stepping primitives for, respectively, standing up ($\boldsymbol{u}^{\text{CU}}_{\text{CoM}}$) and crouching down ($\boldsymbol{u}^{\text{CD}}_{\text{CoM}}$) by vertically moving the CoM while maintaining both feet fixed on the ground. The second subset includes stepping primitives extracted from a *crouched* walking gait in which the ZMP is contained in the humanoid support polygon at all time instants, and the CoM height is lower than that specified by primitives in U^D_{CoM}. Similarly to U^D_{CoM}, such subset includes starting, cruise and stopping steps for various directions of motion. For example, for the forward direction, we have the three steps $\boldsymbol{u}^{\text{CF,start}}_{\text{CoM}}$, $\boldsymbol{u}^{\text{CF,cruise}}_{\text{CoM}}$, and $\boldsymbol{u}^{\text{CF,stop}}_{\text{CoM}}$.
- `free_CoM` is a non-stepping primitive with which the CoM of the humanoid is completely free to move as long as both feet remain fixed on the ground. It is a *stretchable* primitive, in the sense that its duration can be chosen arbitrarily.

[2]The proposed framework can work with any precomputed catalogue of primitives; clearly, the richer the catalogue, the larger the set of tasks for which the planner will be able to compute a whole-body motion.

The three different kinds of stepping primitives (i.e., starting, cruise, and stopping steps) included in the subsets U^D_{CoM} and U^C_{CoM} reflect the three possible phases of a ZMP-based gait. In fact, while during a cruise step the humanoid always maintains dynamic equilibrium, starting and stopping steps permit the humanoid to switch from static to dynamic equilibrium and viceversa, allowing to initiate and terminate a ZMP-based gait, respectively. In addition, starting and stopping steps in U^C_{CoM} permit the humanoid to switch from the upright to the crouched posture (involving an appropriate vertical motion of the CoM) and viceversa, allowing to initiate and terminate a crouched gait, respectively. As for the `free_CoM` primitive, its inclusion in U is particularly important because it allows to build a sequence of movements of arbitrary total duration. This is essential for fulfilling tasks, e.g., a manipulation task specified by a trajectory, whose total duration $t_f - t_i$ is explicitly assigned, using CoM movement primitives whose individual durations are otherwise fixed.

At a given configuration $\boldsymbol{q}_k$, the selection of a specific CoM movement primitive from the catalogue U provides

- a duration T_k for the movement and a resulting time interval $[t_k,\, t_{k+1}]$, with $t_{k+1} = t_k + T_k$;
- a reference trajectory $\boldsymbol{y}^*_{\text{CoM}}$ for the pose of the CoM frame in $[t_k,\, t_{k+1}]$ for all primitives except `free_CoM`;
- a reference trajectory $\boldsymbol{y}^*_{\text{swg}}$ for the pose of the swing foot in $[t_k,\, t_{k+1}]$[3].

We emphasize that a certain CoM primitive does not specify a whole-body motion (i.e., the motion of all the humanoid joints for the entire primitive duration), which can instead be freely chosen by the planner among the infinite that are compatible with the primitive. As a consequence, repetition of the same primitive in different parts of the planning solution will correspond in general to different whole-body motions, depending on the local task history and obstacle placement.

All the stepping primitives described above can be generated, i.e., for each of them the duration and the CoM and swing foot reference trajectories can be precomputed, using suitable (static or dynamic) Walking Pattern Generators. In particular, for the primitives extracted from ZMP-based gaits, e.g., the stepping primitives included in U^D_{CoM} and U^C_{CoM}, we used the MPC-based gait generator introduced in [68] (see also Appendix A). Note that, any step, regardless of the particular type (static, dynamic or crouched), can be performed by swinging either the left or right foot. For this reason, the catalogue U includes both versions for each stepping primitive. In the following discussion, for sake of illustration, we will not explicitly distinguish the two cases and assume that the identity of the swing foot alternates between the two feet.

At a given configuration $\boldsymbol{q}$, the set of selectable CoM primitives consists of a subset of U that depends on $\boldsymbol{q}$ itself, and in particular on which CoM primitive has produced $\boldsymbol{q}$. For example, no static step can be selected after a dynamic cruise step; only another cruise step or a stopping step are admissible. Similarly, the only dynamic step that can follow a static step is a starting step; and so on.

[3]For non-stepping primitives (e.g., `free_CoM`, $\boldsymbol{u}^{\text{CU}}_{\text{CoM}}$, $\boldsymbol{u}^{\text{CD}}_{\text{CoM}}$) such reference trajectory is simply $\boldsymbol{y}^*_{\text{swg}}(t) = \boldsymbol{y}^*_{\text{swg}}(t_k)$ for all $t \in [t_k, t_{k+1}]$, and the identity of the swing foot can be freely chosen.

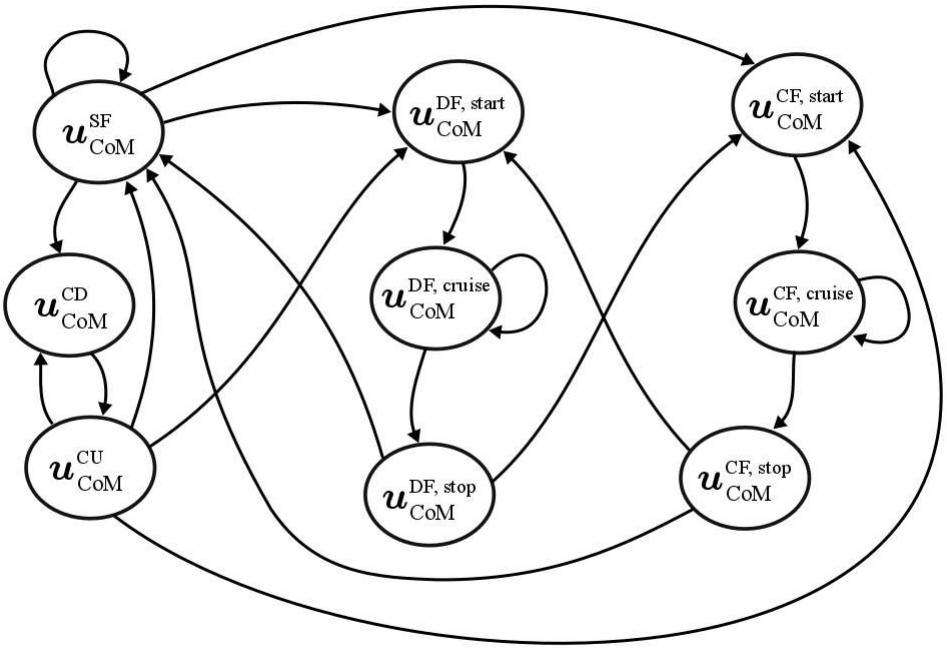

Figure 3.2. The direct graph representing the admissibility of CoM movement primitives. Here, for sake of illustration, only stepping primitives in the forward direction are shown and the `free_CoM` primitive is omitted. The latter is admissible at any statically balanced configuration, i.e., any configuration that has not been produced by a starting or cruise step (either dynamic or crouched).

The admissibility of CoM movement primitives at a given configuration $\boldsymbol{q}$ can be represented as a direct graph, as shown in Fig. 3.2. In this graph each vertex represents a particular CoM primitive included in U, while an edge going from a vertex v to a vertex v' indicates that the primitive represented by v' can be selected from a configuration $\boldsymbol{q}$ that has been produced using the primitive represented by v.

3.2 Basic whole-body planner

In this section, we address the basic instance of the motion planning problem described in Sect. 3.1.2: a humanoid robot is assigned one of the tasks T1-T4 in an environment whose geometry is known in advance. A solution for this problem consists of a configuration space trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, that satisfies the requirements R1-R4.

With our approach, such trajectory is identified by a concatenation of CoM movement primitives that are translated into collision-free whole-body motions which realize such movements while complying with the assigned task.

Sect. 3.2.1 describes the tool used to generate such motions, while Sect. 3.2.2 illustrates the planning strategy. Sect. 3.2.3 presents simulations obtained for a variety of tasks.

3.2.1 Motion generation

The proposed planner works in an iterative fashion by repeated calls to a motion generator. Given a starting configuration $\boldsymbol{q}_k$, with an associated time instant t_k, and a particular CoM movement primitive $\boldsymbol{u}^k_{\text{CoM}}$, with duration T_k, selected from catalogue U, the motion generator computes a feasible whole-body motion $\boldsymbol{q}(t)$, $t \in [t_k, t_{k+1} = t_k + T_k]$, which realizes the reference trajectories $\boldsymbol{y}^*_{\text{CoM}}$ and $\boldsymbol{y}^*_{\text{swg}}$ specified by $\boldsymbol{u}^k_{\text{CoM}}$, as well as the portion of the assigned task contained in the same time interval.

To this end, our motion generation scheme computes first the motion of the CoM frame by plugging the current primitive $\boldsymbol{u}^k_{\text{CoM}}$ in (3.1), and then the joint trajectory (3.2) by integrating the joint velocities produced by the following priority-based kinematic control law [31]:

$$\boldsymbol{v}_{\text{jnt}} = \boldsymbol{J}_L^{\dagger}\dot{\boldsymbol{y}}'_L + \boldsymbol{P}_L(\boldsymbol{J}_A\boldsymbol{P}_L)^{\dagger}(\dot{\boldsymbol{y}}'_A - \boldsymbol{J}_A\boldsymbol{J}_L^{\dagger}\dot{\boldsymbol{y}}'_L) + \boldsymbol{P}_{L,A}\,\boldsymbol{v}_0. \tag{3.4}$$

Here, $\boldsymbol{y}_L = (\boldsymbol{y}^T_{\text{CoM}}, \boldsymbol{y}^T_{\text{swg}})^T$ is the primary locomotion task specified by $\boldsymbol{u}^k_{\text{CoM}}$[4], $\boldsymbol{J}_L$ its Jacobian and $\boldsymbol{P}_L = \boldsymbol{I} - \boldsymbol{J}_L^{\dagger}\boldsymbol{J}_L$; $\boldsymbol{y}_A$ is the secondary task arising from the assigned one, $\boldsymbol{J}_A$ its Jacobian and $\boldsymbol{P}_{L,A} = \boldsymbol{P}_L - (\boldsymbol{J}_A\boldsymbol{P}_L)^{\dagger}(\boldsymbol{J}_A\boldsymbol{P}_L)$. In general, $\dot{\boldsymbol{y}}'_L = \dot{\boldsymbol{y}}^*_L + \boldsymbol{K}_L\boldsymbol{e}_L$ and $\dot{\boldsymbol{y}}'_A = \dot{\boldsymbol{y}}^*_A + \boldsymbol{K}_A\boldsymbol{e}_A$, with $\boldsymbol{e}_L = \boldsymbol{y}^*_L - \boldsymbol{y}_L$ and $\boldsymbol{e}_A = \boldsymbol{y}^*_A - \boldsymbol{y}_A$; $\boldsymbol{y}^*_L$ and $\boldsymbol{y}^*_A$ are the reference values of $\boldsymbol{y}_L$ and $\boldsymbol{y}_A$, respectively. $\boldsymbol{K}_L$ and $\boldsymbol{K}_A$ are positive definite gain matrices. Note that, with the adopted priority-based approach, the assigned task will be executed as much as possible without perturbing the execution of the locomotion task. The choice of assigning the highest priority to the locomotion task is due to its fundamental role in guaranteeing the humanoid equilibrium.

Depending on the assigned task, $\boldsymbol{y}_A$ will take values in an appropriate space and (3.4) will assume a specific form. In case of navigation task (T1), the secondary task $\boldsymbol{y}_A$ is null, as only pure locomotion need to be generated, and (3.4) reduces to

$$\boldsymbol{v}_{\text{jnt}} = \boldsymbol{J}_L^{\dagger}\dot{\boldsymbol{y}}'_L + \boldsymbol{P}_L\boldsymbol{v}_0. \tag{3.5}$$

In case of reaching task (T2), the latter is activated only if the robot CoM at $\boldsymbol{q}_k$ is inside a spherical region centered at the desired set-point $\boldsymbol{y}^*_R$, and having a given radius r^*_R[5]. If T2 is inactive, (3.4) reduces again to (3.5). Otherwise, in (3.4), $\boldsymbol{y}_A = \boldsymbol{y}$, $\boldsymbol{y}^*_A = \boldsymbol{y}^*_R$ and $\dot{\boldsymbol{y}}^*_A = 0$. Similarly, in case of manipulation task (T3), in (3.4), $\boldsymbol{y}_A = \boldsymbol{y}$, $\boldsymbol{y}^*_A = \boldsymbol{y}^*_M$, and $\dot{\boldsymbol{y}}^*_A = \dot{\boldsymbol{y}}^*_M$. In case of visual task (T4), using the IBVS approach, $\boldsymbol{y}_A = \boldsymbol{f}$ in (3.4). The rate of variation of $\boldsymbol{f}$ depends on the camera linear and angular velocities $\dot{\boldsymbol{y}} = \boldsymbol{J}\boldsymbol{v}_{\text{jnt}}$, with $\boldsymbol{J}$ the Jacobian of $\boldsymbol{y}$ w.r.t. $\boldsymbol{q}_{\text{jnt}}$, as (see [70] for details)

$$\dot{\boldsymbol{f}} = \boldsymbol{J}_f(\dot{\boldsymbol{y}} - \dot{\boldsymbol{y}}^*_P) \tag{3.6}$$

[4]Note that, the locomotion task reduces to $\boldsymbol{y}_L = \boldsymbol{y}_{\text{swg}}$ if the current primitive is `free_CoM`.

[5]This approach avoids the generation of unnatural motions of the humanoid. In fact, by considering the reaching task always active, the robot will try to extend its arm towards the set-point, especially when the initial error is large. This is also likely to push some joints close to their limit, thus making motion generation more difficult. Generation of smoother reaching motions is out of the scope of this chapter, although this can be easily accounted in the proposed framework by involving the strategy that we proposed in [69].

where $\boldsymbol{J}_f = \boldsymbol{J}_f(\boldsymbol{f}, Z)$ is the interaction matrix [71]. Expressing (3.6) in terms of joint velocities

$$\dot{\boldsymbol{f}} = \boldsymbol{J}_f \boldsymbol{J} \boldsymbol{v}_{\text{jnt}} - \boldsymbol{J}_f \dot{\boldsymbol{y}}_P^*$$

yields that in (3.4), $\dot{\boldsymbol{y}}_A^* = \boldsymbol{J}_f \dot{\boldsymbol{y}}_P^*$ and $\boldsymbol{J}_A = \boldsymbol{J}_f \boldsymbol{J}$. Furthermore, $\boldsymbol{y}_A^* = 0$ to drive the feature to the center of the image plane.

The choice of the null-space vector $\boldsymbol{v}_0$ in (3.4-3.5) is arbitrary. A possible choice consists in setting

$$\boldsymbol{v}_0 = \boldsymbol{v}_{\text{rand}} \tag{3.7}$$

where $\boldsymbol{v}_{\text{rand}}$ is a bounded-norm randomly generated n-vector. This option allows to further explore the space of possible solutions, exploiting the humanoid kinematic redundancy. Another option consists in setting

$$\boldsymbol{v}_0 = -\eta \nabla_{\boldsymbol{q}_{\text{jnt}}} H(\boldsymbol{q}_{\text{jnt}}) \tag{3.8}$$

where η is a positive stepsize and $H(\boldsymbol{q}_{\text{jnt}})$ is a cost function that may be chosen, for example, so as to maximize the available joint range.

Configurations generated via integration of (3.4-3.5) are continuously checked for collisions (requirement R1), and for position and velocity joint limits (requirement R2). In view of the use of CoM movement primitives, equilibrium (requirement R3) is guaranteed by construction except for the `free_CoM` primitive, for which static equilibrium is explicitly checked. In case of visual task (T4), occlusion of the moving point P is checked as well. If a violation occurs, the current execution of the motion generator is interrupted, and a failure is returned to the planner. Otherwise, integration proceeds up to t_{k+1}. In this case, a feasible whole-body motion that complies with the assigned task over $[t_k, t_{k+1}]$ is obtained, and can be returned to the planner.

3.2.2 Planner overview

The proposed planner, whose pseudocode is given in Algorithm 1, uses a RRT-like strategy to build a tree $\mathcal{T}$ in configuration space. In $\mathcal{T}$, a vertex $v = (\boldsymbol{q}, \boldsymbol{u}_{\text{CoM}})$ consists of a configuration $\boldsymbol{q}$ of the humanoid associated to a certain time instant t, and a CoM primitive $\boldsymbol{u}_{\text{CoM}}$ through which it has been generated; an edge represents a feasible (in the sense of requirements R1-R3 of Sect. 3.1.2) whole-body motion joining two adjacent vertexes. Each edge realizes a certain CoM movement primitive of the catalogue U, and (for T3-T4) a portion of the assigned task. At the beginning, $\mathcal{T}$ is rooted at vertex v_{ini} containing the robot initial configuration $\boldsymbol{q}_{\text{ini}}$.

The generic iteration of the planner starts by randomly sampling a point $\boldsymbol{p}_{\text{rand}}$ in the workspace (for T1-T2), on the trajectory $\boldsymbol{y}_M^*$ (for T3), or on the trajectory $\boldsymbol{y}_P^*$ (for T4). Then, the planner assigns to each vertex v in $\mathcal{T}$ a probability that is inversely proportional to an appropriately defined distance between the contained configuration $\boldsymbol{q}$ and $\boldsymbol{p}_{\text{rand}}$. To compute the distance between a configuration $\boldsymbol{q}$ and a point $\boldsymbol{p}$, the planner makes use of a metric $\gamma(\boldsymbol{q}, \boldsymbol{p})$ that is specifically defined according to the assigned task. In particular, $\gamma(\boldsymbol{q}, \boldsymbol{p})$ is defined as the distance between the ground projection of $\boldsymbol{p}$ and that of the robot CoM position at $\boldsymbol{q}$ (for T1-T3), or as the distance between $\boldsymbol{p}$ and the camera principal axis when the robot

Algorithm 1: Basic Whole-Body Planner

$v_{\text{ini}} \leftarrow (\boldsymbol{q}_{\text{ini}}, \emptyset)$;
AddVertex($\mathcal{T}, v_{\text{ini}}$);
$i \leftarrow 0$;
repeat
  $\boldsymbol{p}_{\text{rand}} \leftarrow$RandomSample();
  $v_{\text{near}} \leftarrow$NearestVertex($\mathcal{T}, \boldsymbol{p}_{\text{rand}}$);
  $\boldsymbol{u}_{\text{CoM}} \leftarrow$RandomPrimitive($U, v_{\text{near}}$);
  $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}} \leftarrow$MotionGenerator($\boldsymbol{q}_{\text{near}}, \boldsymbol{u}_{\text{CoM}}$);
  if $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}} \neq \emptyset$ **then**
    $v_{\text{new}} \leftarrow (\boldsymbol{q}_{\text{new}}, \boldsymbol{u}_{\text{CoM}})$;
    Add($\mathcal{T}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}, v_{\text{new}}$);
  $i \leftarrow i + 1$;
until SolutionFound() **or** $i = i_{\max}$;

is in $\boldsymbol{q}$ (for T4). The probability distribution resulting from this procedure is used to randomly choose a vertex v_{near} of $\mathcal{T}$ for a tree expansion attempt.

Once $v_{\text{near}} = (\boldsymbol{q}_{\text{near}}, \boldsymbol{u}_{\text{CoM}}^{\text{near}})$, with associated time instant t_k, has been identified, a CoM movement primitive $\boldsymbol{u}_{\text{CoM}}$, with duration T_k, is randomly selected from the currently admissible subset of U; as explained in Sect. 3.1.3, this subset depends on $\boldsymbol{q}_{\text{near}}$ and, in particular, on the primitive $\boldsymbol{u}_{\text{CoM}}^{\text{near}}$ through which it has been generated.

At this point, the motion generator (see Sect. 3.2.1) is called in order to compute a whole-body motion $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ that realizes the reference CoM and swing foot trajectories specified by $\boldsymbol{u}_{\text{CoM}}$, and complies with the assigned task. If motion generation is successful, the planner constructs a new vertex $v_{\text{new}} = (\boldsymbol{q}_{\text{new}}, \boldsymbol{u}_{\text{CoM}})$, where $\boldsymbol{q}_{\text{new}}$ is the final configuration of $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$, with associated time instant $t_k + T_k$. The new vertex v_{new} is added to the tree, together with the edge $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$.

Planning terminates when $\boldsymbol{q}_{\text{new}}$ completes the assigned task or a maximum number of iterations $i_{\max}$ is reached. In order for $\boldsymbol{q}_{\text{new}}$ to complete the assigned task, it must be such that the midpoint between the feet is inside the desired region, i.e., $\|\boldsymbol{k}(\boldsymbol{q}_{\text{new}}) - \boldsymbol{y}_N^*\| < r_N^*$ (for T1), the chosen hand reached the desired set-point, i.e., $\boldsymbol{k}(\boldsymbol{q}_{\text{new}}) = \boldsymbol{y}_R^*$ (for T2), its associated time instant coincides with t_{fin} (for T3-T4)[6]. In positive case, the sequence of edges joining v_{ini} to v_{new} constitutes the whole-body motion $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, representing the solution to the planning problem.

3.2.3 Planning experiments

We implemented the proposed planner as a C++ plugin for the V-REP simulator, and tested it with the NAO robot, a 58 cm tall humanoid robot with 23 degrees of freedom by SoftBank Robotics. All simulations have been performed on an Intel Core i7 running at 2.7 GHz. We consider four scenarios (see Fig. 3.3) in which the robot is assigned a different task (T1-T4). The planner is provided with a catalogue of primitives U defined as in (3.3), and $i_{\max}$ is set to 1000.

[6]In cases of tasks T3-T4, when the time instant associated to $\boldsymbol{q}_{\text{new}}$ coincides with t_{fin}, by construction, the sequence of edges joining v_{ini} to v_{new} constitutes a configuration space trajectory along which the assigned task is continuously satisfied, from t_{ini} to t_{fin}. Such trajectory will then represent a solution for the planning problem.

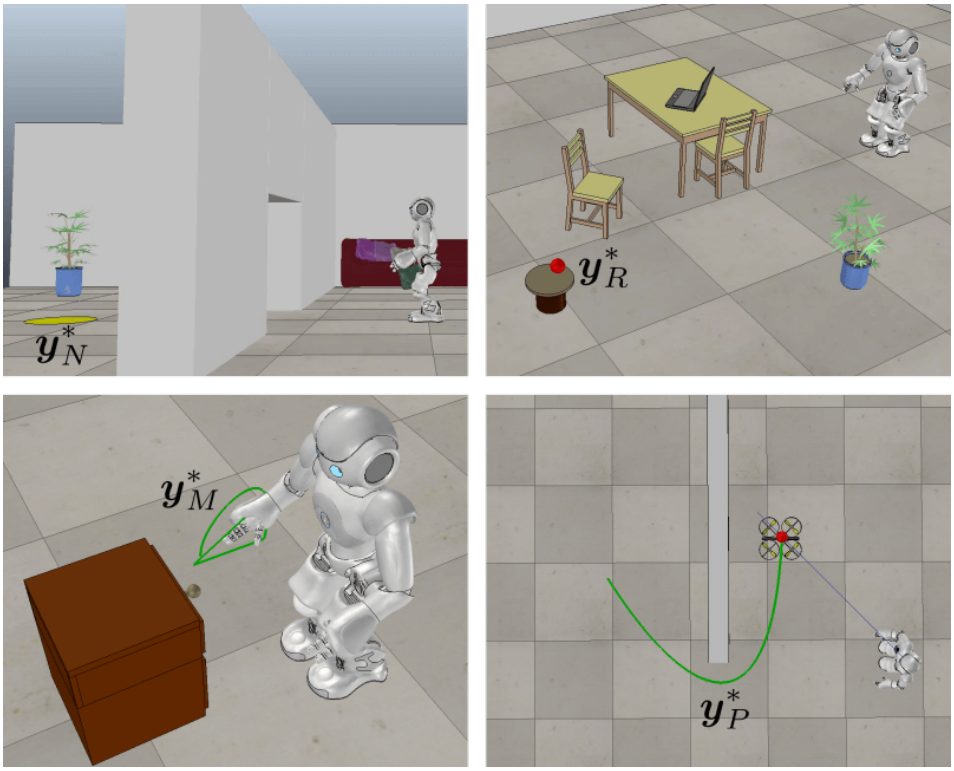

Figure 3.3. The four considered scenarios.

In the first scenario (Fig. 3.3, upper left), the robot is assigned a navigation task (T1): it must bring the midpoint between the feet inside the yellow circular region, whose radius is set to $r_N^* = 0.15$ m. In order to complete the task, the robot must pass through a low passage in the wall separating its initial location and the destination. The solution shown in Fig. 3.4 leads the robot towards the wall by some forward dynamic steps (snapshot 2), and then switches to a slightly crouched gait (snapshot 3). Once the robot clears the low passage, erect posture is recovered (snapshot 4) and the destination is reached using again some forward dynamic steps (snapshots 5 and 6).

In the second scenario (Fig. 3.3, upper right), the robot is assigned a reaching task (T2): it must grasp the red ball located on a low table. With the solution shown in Fig. 3.5, the robot begins by taking some diagonal dynamic steps (snapshots 2 and 3) to avoid collisions with a table and a chair. Then, it moves towards the red ball by forward dynamic steps (snapshot 4). Once the robot is sufficiently close to the ball, it first executes a crouching movement (snapshot 5), and finally reaches the ball using the `free_CoM` primitive (snapshot 6).

In the third scenario (Fig. 3.3, bottom left), the robot is assigned a manipulation task (T3): it must open a drawer and grasp an object (a ball) located inside it. Such particular task is specified through a desired trajectory for the right hand of the robot, having a total duration of 13 s, consisting in three subtrajectories describing the elementary actions of reaching the knob, opening the drawer, and

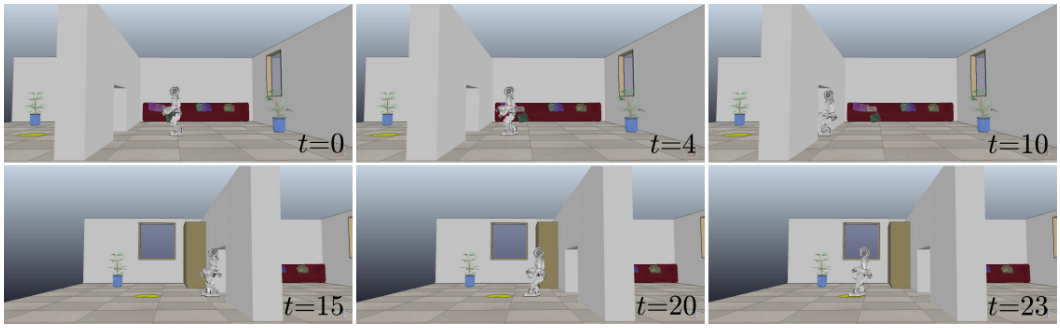

Figure 3.4. Scenario 1: snapshots from a solution.

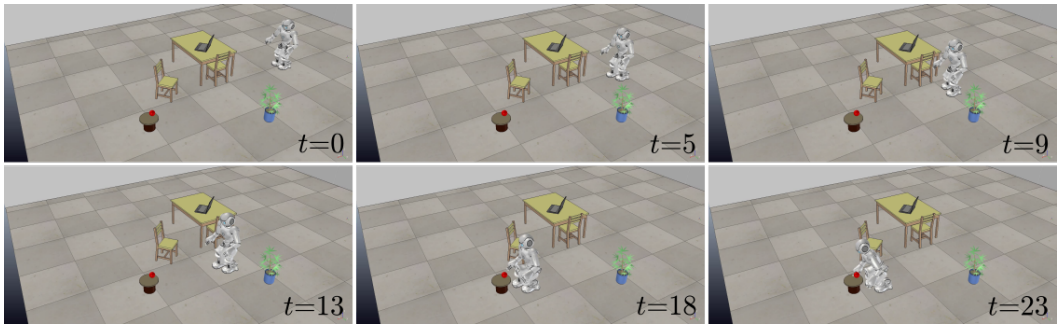

Figure 3.5. Scenario 2: snapshots from a solution.

grasping the object, respectively[7]. A possible solution computed by the proposed planner for this scenario is shown in Fig. 3.6. At the beginning, the robot takes few dynamic steps towards the drawer, and then stops stepping to reach the knob using the `free_CoM` primitive (snapshot 2). Then, the robot performs some backward dynamic steps (snapshot 3), before completing the opening subtrajectory using again the `free_CoM` primitive (snapshot 4). The last portion of the task is completed performing two forward steps and the `free_CoM` primitive (snapshots 5 and 6), while correctly avoiding collisions with the drawer.

In the fourth scenario (Fig. 3.3, bottom right), the robot is assigned a visual task (T4): it must track the red ball rigidly attached on a moving quadrotor. The trajectory of the target is known and has duration 18 s. Such trajectory brings the target behind a wall that occludes the target to the robot at its initial location; then, in order to fulfil the task, the robot must perform an appropriate sequence of steps. With the solution shown in Fig. 3.7, the robot takes some diagonal dynamic steps in the left direction, while keeping the ball at the center of the image plane. Once the robot reaches a location from where the last portion of the target trajectory is no more occluded by the wall, it performs a stopping step (snapshot 5) and tracks the target until the end using the `free_CoM` primitive (snapshot 6).

Since the planner is randomized, to evaluate its performance we have performed a set of 10 simulations on each scenario. Table 3.1 collects some average performance data for each scenario. In particular, the table indicates the number of vertexes in the tree produced by the planner, the time needed to compute a solution, and

[7] In general, such trajectory may be the output of a task planner.

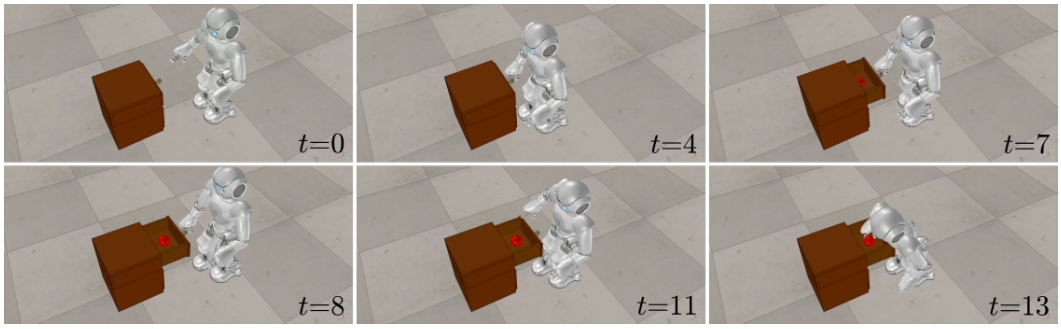

Figure 3.6. Scenario 3: snapshots from a solution.

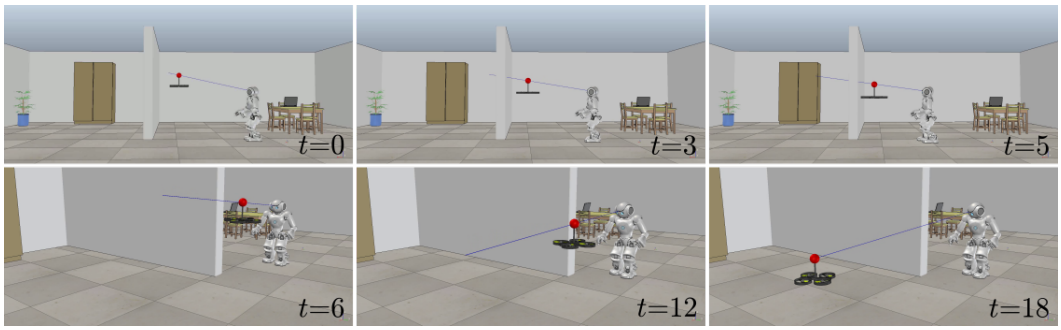

Figure 3.7. Scenario 4: snapshots from a solution.

the duration of the computed whole-body motion. Note that, when considering manipulation or visual tasks, the planner generates smaller trees, i.e., containing fewer vertexes, w.r.t. the cases in which navigation or reaching tasks are considered. This is due to the fact that, for tasks T3-T4, the configurations generated by the motion generator (see Sect. 3.2.1) belong to the task-constrained configuration space, i.e., the submanifold of $\mathcal{C}$ containing all the configurations that satisfy the task at a particular sample. Such additional requirement severely restricts the configuration space region where a solution can be found. A detailed analysis of the effect of including a task-constraint in the planning stage will be given in Sect. 5.1.

Although the presented motion planner proved to be able in computing sensible solutions for a variety of tasks in a reasonable time, we should acknowledge that, especially in larger environments or in the presence of more complex tasks, the computational complexity of the method might increase, ultimately forcing the robot

scenario	tree size (# vertexes)	planning time (s)	motion duration (s)
1	98	34.39	23.18
2	93	24.55	22.75
3	36	25.19	13
4	50	26.64	18

Table 3.1. Planner performance data

to wait for a planning solution before starting the motion. We address this situation in the next section.

3.3 Anytime whole-body planning framework

In this section, we address the motion planning problem described in Sect. 3.1.2 under the additional constraint of limited time for planning. In particular, a humanoid robot is assigned a reaching task (T2) in a known environment, and must start moving after a given time budget $\overline{\Delta T}_P$, i.e., it is allowed to plan an initial (partial) solution within $\overline{\Delta T}_P$.

A solution for this problem consists of a configuration space trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, that satisfies the requirements R1-R4, constituted by a succession of on-line planned partial solutions.

Sect. 3.3.1 describes the proposed scheme to solve the given problem. At the core of such scheme there is a fast planner, presented in Sect. 3.3.2, consisting in an adaptation of the basic planner described in the previous section. Sect. 3.3.3 discusses how the proposed scheme can easily handle deadlock situations. Simulation results are discussed in Sect. 3.3.4.

3.3.1 Framework overview

To address the described problem, we propose a framework that works in an anytime fashion by performing planning and execution intervals in parallel: a previously planned whole-body motion is executed while simultaneously planning a new one for the subsequent execution interval.

The objective of the generic planning interval is to produce a whole-body motion that starts at the configuration that the robot will reach at the end of the simultaneously running execution interval. Such motion must be planned before the termination of the latter, in such a way to guarantee the continuity of robot motion, avoiding the need to stop for planning future motions. Thus, the duration of the simultaneously executed whole-body motion limits the time budget within which the future motion can be computed, reflecting in a limited workspace region for planning. Henceforth we will refer to such region as *planning zone*.

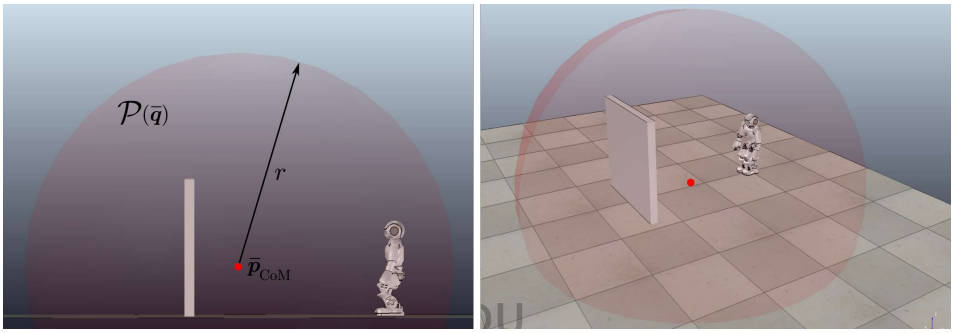

Figure 3.8. An example of planning zone used in our anytime planning framework.

At the generic planning interval, let $\bar{\boldsymbol{q}}$ be the final configuration of the simultaneously executed whole-body motion, and denote by $\mathcal{P}(\bar{\boldsymbol{q}})$ the planning zone, whose location and geometry depend on the configuration $\bar{\boldsymbol{q}}$. To illustrate the proposed method, we will assume that the planning zone consists of a sphere of a given radius r (as depicted in Fig. 3.8) centered at the CoM position $\bar{\boldsymbol{p}}_{\mathrm{CoM}}$ of the humanoid at the configuration $\bar{\boldsymbol{q}}$. In our framework, a planning interval consists in an invocation of a specifically designed Local Motion Planner (LMP), described in the next section, that is in charge of producing, within the corresponding time budget, a whole-body motion that starts at $\bar{\boldsymbol{q}}$, and is feasible within $\mathcal{P}(\bar{\boldsymbol{q}})$. Hereafter, we will refer to the whole-body motion produced during a certain planning interval as *local plan*, as its feasibility (in the sense of requirements R1-R3 described in Sect. 3.1.2) is guaranteed only within a fixed planning zone.

The anytime planning/replanning scheme at the core of our framework is shown in Algorithm 2. It starts by invoking the LMP that is in charge of producing, within a predefined time budget $\overline{\Delta T}_P$, the first local plan $\boldsymbol{q}_0(t)$, $t \in [t_0 = t_{\mathrm{ini}} + \overline{\Delta T}_P, t_1 = t_0 + \Delta T_{E,0}]$, with $\Delta T_{E,0}$ its duration, that starts at the initial robot configuration $\boldsymbol{q}_{\mathrm{ini}}$, and is guaranteed to be feasible in the first planning zone $\mathcal{P}_0 = \mathcal{P}(\boldsymbol{q}_{\mathrm{ini}})$.

Once such plan is computed, an iterative phase is entered. In this phase, each iteration consists of a concurrent run of an execution and planning interval. In particular, at the i-th generic iteration, our anytime framework simultaneously:

- executes the *current* local plan $\boldsymbol{q}_{i-1}(t)$, $t \in [t_{i-1}, t_i = t_{i-1} + \Delta T_{E,i-1}]$, with $\Delta T_{E,i-1}$ its duration, computed at the $(i-1)$-th planning interval;
- plans the *next* local plan $\boldsymbol{q}_i(t)$, $t \in [t_i, t_{i+1} = t_i + \Delta T_{E,i}]$, with $\Delta T_{E,i}$ its duration, to be realized in the $(i+1)$-th execution interval.

In order to plan the next local plan, the LMP is invoked. It is provided with the final configuration $\bar{\boldsymbol{q}}_i = \boldsymbol{q}_{i-1}(t_i)$ of the current local plan, the planning zone $\mathcal{P}_i = \mathcal{P}(\bar{\boldsymbol{q}}_i)$, and the time budget $\Delta T_{P,i} = \Delta T_{E,i-1}$. This invocation of the LMP, that will last at most $\Delta T_{P,i}$, produces a local plan that starts at the final configuration $\bar{\boldsymbol{q}}_i = \boldsymbol{q}(t_i)$ of the current local plan, and is feasible within the planning zone $\mathcal{P}_i$.

Note the following points.

- The duration $\Delta T_{E,i}$ of the local plan computed at the generic i-th iteration is not assigned, as it is autonomously determined by the LMP.
- While the i-th execution and planning intervals will start at the same time instant, their duration may be in general different. In fact, while the duration of the i-th execution interval will be exactly $\Delta T_{E,i-1}$ as determined by the duration of the local plan found at the $(i-1)$-th planning interval, the i-th (simultaneously running) planning interval will terminate as soon as the next local plan is found by the LMP, that may happen before the provided time budget $\Delta T_{P,i}$ runs out.

This iterative procedure terminates when the LMP computes a local plan such that the desired set-point is eventually reached, i.e., $\boldsymbol{k}(\bar{\boldsymbol{q}}_i) = \boldsymbol{y}_R^*$ with $\boldsymbol{k}(\bar{\boldsymbol{q}}_i)$ the end-effector position at the final configuration $\bar{\boldsymbol{q}}_i$ of the next local plan. In this case, the latter represents the last whole-body motion to be executed.

Algorithm 2: Anytime Planning/Replanning Scheme

$\boldsymbol{q}_0(t),\, t \in [t_0, t_1] \leftarrow \text{LMP}(\boldsymbol{q}_{\text{ini}},\, \mathcal{P}_0,\, \overline{\Delta T}_P)$;
$\bar{\boldsymbol{q}}_1 \leftarrow \boldsymbol{q}(t_1)$;
$i \leftarrow 1$;
while $\boldsymbol{k}(\bar{\boldsymbol{q}}_i) \neq \boldsymbol{y}_R^*$ **do**
  $\Delta T_{P,i} \leftarrow \Delta T_{E,i-1}$;
  simultaneously do:
  • Execute($\boldsymbol{q}_{i-1}(t),\, t \in [t_{i-1}, t_i]$);
  • $\boldsymbol{q}_i(t),\, t \in [t_i, t_{i+1}] \leftarrow \text{LMP}(\bar{\boldsymbol{q}}_i,\, \mathcal{P}_i,\, \Delta T_{P,i})$;
  $i \leftarrow i + 1$;
  $\bar{\boldsymbol{q}}_i \leftarrow \boldsymbol{q}(t_i)$;
end
Execute($\boldsymbol{q}_{i-1}(t),\, t \in [t_{i-1}, t_i]$);

3.3.2 Local motion planner

In this section, we present the local motion planner (LMP) that is invoked at each planning interval with the aim of computing a next local plan.

LMP consists in an adaptation of the basic planner presented in Sect. 3.2.2 that is suitable for on-line applications. In order to reduce the computational complexity of the construction of tree $\mathcal{T}$, LMP postpones the joint motion generation, and the related collision checks[8] until they are strictly needed.

The proposed LMP (see Procedure 1) works in two sequential stages. The first stage, called *lazy stage*, quickly creates a new tree $\mathcal{T}$ populated by vertexes containing partial configurations $\boldsymbol{q} = (\boldsymbol{q}_{\text{CoM}}, \emptyset)^T$ where the sub-vector of joint angles $\boldsymbol{q}_{\text{jnt}}$ is unspecified, as well as the whole-body motions joining adjacent vertexes. The sub-vector $\boldsymbol{q}_{\text{CoM}}$ in each vertex is obtained exploiting the concept of CoM movement primitive: the pose displacement of the CoM frame, given its starting pose and a certain primitive, is directly provided by the primitive itself. Consequently, collision checks are not performed along edges, but only at vertex level using a simplified occupancy volume for the robot. The second stage, called *validation stage*, generates feasible whole-body motions, along which collision checks are performed using the actual occupancy volume for the robot, only for selected branches of $\mathcal{T}$ that potentially allow to find a local plan.

The two stages are described in details in the following subsections. The time budget ΔT_P is split into two intervals, ΔT_P^L and ΔT_P^V, dedicated to the lazy and validation stages, respectively. When the time budget ΔT_P^L assigned to the lazy stage runs out, expansion of the tree is stopped. Instead, if the time budget ΔT_P^V assigned to the validation stage runs out before it returns a local plan, a failure is returned[9]. For sake of illustration, in the following, we will omit explicit discussion of the interruption mechanism.

[8] Collision check represents the most expensive operation in sampling-based planning [72, 73]

[9] In case LMP returns a failure, a local plan for the next execution interval does not exist. Although we do not explicitly consider this case in the presented framework, a possible solution consists in allowing the humanoid to perform a safe stopping motion (for example using the technique described in Sect. 7.8.3), and then to invoke again LMP with a sufficiently large time budget.

Procedure 1: LMP($\bar{\boldsymbol{q}}$, $\mathcal{P}$, ΔT_P)

1 $(\Delta T_P^L, \Delta T_P^V) \leftarrow$ SplitTimeBudget(ΔT_P);
2 $\mathcal{T} \leftarrow$ LazyStage($\bar{\boldsymbol{q}}$, $\mathcal{P}$, ΔT_P^L);
3 $\boldsymbol{q}(t), t \in [\bar{t}, \bar{t} + \Delta T_E] \leftarrow$ ValidationStage($\mathcal{T}$, $\mathcal{P}$, ΔT_P^V);
4 **return** $\boldsymbol{q}(t), t \in [\bar{t}, \bar{t} + \Delta T_E]$;

Lazy stage

The lazy stage, whose pseudocode is given in Procedure 2, roots the tree $\mathcal{T}$ at $\bar{v} = (\bar{\boldsymbol{q}}, \bar{\boldsymbol{u}}_{\text{CoM}})$, where $\bar{\boldsymbol{q}}$ is the final configuration on the current plan, and $\bar{\boldsymbol{u}}_{\text{CoM}}$ is the CoM primitive through which it has been generated[10].

At the generic iteration, similarly to the basic planner, a random sample point $\boldsymbol{p}_{\text{rand}}$ is picked in the workspace. Then, its nearest vertex $v_{\text{near}} = ((\boldsymbol{q}_{\text{CoM}}^{\text{near}}, \emptyset)^T, \boldsymbol{u}_{\text{CoM}}^{\text{near}})$ is selected using the distance metric (for T2) described in Sect. 3.2.2 from the allowed set of vertexes $\mathcal{V}_A$ (described in detail below). Once v_{near} is identified, a CoM primitive $\boldsymbol{u}_{\text{CoM}}$ is randomly chosen from the admissible subset of U at v_{near}[11].

Let t_k be the time instant associated to the partial configuration $(\boldsymbol{q}_{\text{CoM}}^{\text{near}}, \emptyset)^T$, and T_k be the duration of the chosen primitive $\boldsymbol{u}_{\text{CoM}}$. The reference pose $\boldsymbol{q}_{\text{CoM}}^{\text{new}}$ of the CoM frame at $t_{k+1} = t_k + T_k$, attained by applying $\boldsymbol{u}_{\text{CoM}}$ from $\boldsymbol{q}_{\text{CoM}}^{\text{near}}$, is computed using (3.1). Then, the generated partial configuration $(\boldsymbol{q}_{\text{CoM}}^{\text{new}}, \emptyset)^T$ is checked for collisions with workspace obstacles (requirement R1) within the planning zone $\mathcal{P}$ using a simplified occupancy volume $\mathcal{S}(\boldsymbol{q}_{\text{CoM}}^{\text{new}})$ for the robot.

In general, different choices for such simplified occupancy volume are possible. The simplest option consists in choosing $\mathcal{S}(\boldsymbol{q}_{\text{CoM}}^{\text{new}})$ as a cylindrical volume having centroid at the position components of $\boldsymbol{q}_{\text{CoM}}^{\text{new}}$. Another option consists in checking collisions only at footstep level by considering the volume occupied by the (alternating) support foot at t_{k+1}. The reference pose $\boldsymbol{q}_{\text{CoM}}^{\text{sup}}$ of the support foot at t_{k+1} can be computed, analogously to (3.1), using its pose at t_k (this information may be stored within vertex v_{near} at the time of its creation) and the reference swing foot trajectory specified by $\boldsymbol{u}_{\text{CoM}}$.

If the volume $\mathcal{S}(\boldsymbol{q}_{\text{CoM}}^{\text{new}})$ is collision-free, a new vertex $v_{\text{new}} = ((\boldsymbol{q}_{\text{CoM}}^{\text{new}}, \emptyset)^T, \boldsymbol{u}_{\text{CoM}})$ is constructed and added to $\mathcal{T}$ as a child of v_{near}. Then, a last operation is performed before starting a new iteration. It consists in updating the subset $\mathcal{V}_A$ of vertexes of $\mathcal{T}$ that are allowed to be selected for expansion attempts. In particular, $\mathcal{V}_A$ contains all the vertexes in $\mathcal{T}$ except those whose associated partial configuration $(\boldsymbol{q}_{\text{CoM}}, \emptyset)^T$ is such that one of the following two conditions is satisfied:

(*i*) the simplified occupancy volume $\mathcal{S}(\boldsymbol{q}_{\text{CoM}})$ is not completely contained in the planning zone $\mathcal{P}$;

(*ii*) the robot CoM is inside a spherical region centered at the desired set-point $\boldsymbol{y}_R^*$, and having a given radius r_R^*, where the reaching task can be activated (see Sect. 3.2.1).

[10] This information can be retrieved from the tree created at the previous invocation of LMP.

[11] In this stage, selection of `free_CoM` is not allowed as it does not provide any reference displacement for the CoM frame.

Procedure 2: LazyStage($\bar{\boldsymbol{q}}$, $\mathcal{P}$, ΔT_P^L)

$\bar{v} \leftarrow (\bar{\boldsymbol{q}}, \bar{\boldsymbol{u}}_{\text{CoM}})$;
AddVertex($\mathcal{T}, \bar{v}$);
$\mathcal{V}_A \leftarrow \{\bar{v}\}$;
while TimeAvailable(ΔT_P^L) **do**
  $\boldsymbol{p}_{\text{rand}} \leftarrow$RandomSample();
  $v_{\text{near}} \leftarrow$NearestVertex($\mathcal{V}_A, \boldsymbol{p}_{\text{rand}}$);
  $\boldsymbol{u}_{\text{CoM}} \leftarrow$RandomPrimitive($U, v_{\text{near}}$);
  $\boldsymbol{q}_{\text{CoM}}^{\text{new}} \leftarrow$ComputeCoMDisplacement($\boldsymbol{q}_{\text{CoM}}^{\text{near}}, \boldsymbol{u}_{\text{CoM}}$);
  if CollisionFree($\mathcal{S}(\boldsymbol{q}_{\text{CoM}}^{\text{new}})$) **then**
    $v_{\text{new}} \leftarrow ((\boldsymbol{q}_{\text{CoM}}^{\text{new}}, \emptyset)^T, \boldsymbol{u}_{\text{CoM}})$;
    AddVertex($\mathcal{T}, v_{\text{near}}, v_{\text{new}}$);
    Update($\mathcal{V}_A, v_{\text{new}}$);
end
return $\mathcal{T}$;

Accordingly, v_{new} is added to $\mathcal{V}_A$ if it does not satisfy neither (i) or (ii). By allowing the selection of vertexes only in $\mathcal{V}_A$, at the end of the lazy stage, each branch of $\mathcal{T}$ will contain, by construction, at most only one vertex satisfying (i) or (ii), and such vertex (if any) will be the last (i.e., the leaf) along the branch. Roughly speaking, a branch whose ending vertex satisfies (i) or (ii) potentially provides a local plan that leads the robot to, respectively, approach the boundary of the current planning zone (so as to continue task-oriented exploration of the workspace), or suffieciently close to the desired set-point (so as to eventually complete the assigned task).

Validation stage

At the end of the lazy stage, a subset of the branches in $\mathcal{T}$ provide a set of *candidate local plans*. In particular, a branch of $\mathcal{T}$ provides a candidate local plan if its ending vertex $\hat{v}$ satisfies one of the conditions (i)-(ii) described above. If $\hat{v}$ satisfies (i), the candidate local plan consists of the portion of the branch leading from the root $\bar{v}$ to the parent of $\hat{v}$. On the other hand, if $\hat{v}$ satisfies (ii), the candidate local plan consists of the entire branch leading from the root $\bar{v}$ to $\hat{v}$.

The first operation of the validation stage, whose pseudocode is given in Procedure 3, consists in selecting the best candidate local plan among those resulting from $\mathcal{T}$. With the aim of further approaching the desired set-point, and possibly completing the task, the best candidate local plan is chosen as the one whose ending vertex contains the closest configuration to the desired set-point $\boldsymbol{y}_R^*$ in terms of the distance metric. Let $\pi^* = \{v_0 = \bar{v}, v_1, \ldots, v_K\}$ be the selected candidate local plan, composed by $K+1$ vertexes. A validation attempt, that consists in generating a feasible whole-body motion between each pair of consecutive vertexes in π^*, starts. At the k-th iteration, the primitive $\boldsymbol{u}_{\text{CoM}}^k$ that produced the partial configuration $(\boldsymbol{q}_{\text{CoM}}^k, \emptyset)^T$ during the lazy stage is extracted from v_k, and the (full) configuration $\boldsymbol{q}_{k-1}$ is extracted from its parent v_{k-1}. Then, the motion generator introduced in Sect. 3.2.1 is invoked for computing a whole-body motion $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}$ that realizes the reference CoM and swing foot trajectories specified by $\boldsymbol{u}_{\text{CoM}}^k$, and satisfies requirements R1-R3. To this purpose, during motion generation, collision checking

Procedure 3: ValidationStage($\mathcal{T}$, $\mathcal{P}$, ΔT_P^V)

```
while TimeAvailable(ΔT_P^V) do
    π* ← BestCandidatePlan(T);
    if π* = ∅ then
        return ∅;
    k ← 0;
    repeat
        k ← k + 1;
        q_{k-1}q_k ←MotionGenerator(q_{k-1}, u^k_CoM);
        if q_{k-1}q_k ≠ ∅ then
            AddEdge(T, q_{k-1}q_k);
            UpdateVertex(T, v_k, q_k);
            if k = K then
                q(t), t ∈ [t̄, t_K] ←RetrieveMotion(v̄, v_K);
                return q(t), t ∈ [t̄, t_K];
    until q_{k-1}q_k = ∅;
    RemoveSubtree(T, v_k);
end
return ∅;
```

is performed on each produced configuration $\boldsymbol{q}$ using the actual occupancy volume $\mathcal{R}(\boldsymbol{q})$ for the robot. If motion generation is successful, the edge $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}$ is added in the tree $\mathcal{T}$ to join vertexes v_{k-1} and v_k, and vertex v_k in $\mathcal{T}$ is updated with the full configuration $\boldsymbol{q}_k = (\boldsymbol{q}^k_{\text{CoM}}, \boldsymbol{q}^k_{\text{jnt}})^T$ resulting at the end of the generated whole-body motion. Otherwise, a feasible whole-body motion joining vertex v_k to its parent does not exist, and the subtree of $\mathcal{T}$ rooted at v_k (that at this point represents a disconnected component) is removed; the best candidate local plan π^* resulting from the updated tree $\mathcal{T}$ is selected, and a new validation attempt starts.

When a feasible whole-body motion between each pair of consecutive vertexes in π^* has been generated, the sequence of edges joining $\bar{v}$ to v_K in $\mathcal{T}$ constitutes the configuration space trajectory $\boldsymbol{q}(t)$, $t \in [\bar{t}, t_K]$, that is returned by the LMP as the produced local plan.

For sake of illustration, in Procedure 3, we omitted to explicitly show two simple operations aimed at, respectively, reusing portions of the tree already generated in previous (failed) validation attempts, and producing a final local plan that completes the assigned task.

The first operation takes place at the beginning of the generic k-th iteration of a validation attempt. It consists in checking if vertex v_k contains a full configuration where also the sub-vector of joint angles is specified. In this case, a feasible whole-body motion between v_k and its parent has already been generated by a previous validation attempt (of a different candidate local plan), and the procedure can directly skip to consider the next vertex v_{k+1} in π^*.

The second operation takes place after the chosen candidate local plan has been entirely validated. In case its ending vertex v_K satisfies condition (ii), an additional invocation of the motion generator is performed with the aim of computing a final whole-body motion $\overline{\boldsymbol{q}_K\boldsymbol{q}_{\text{fin}}}$ that, using the `free_CoM` primitive, allows to reach the desired set-point $\boldsymbol{y}^*_R$. If such motion generation succeeds, the concatenation of the

sequence of edges joining $\bar{v}$ to v_K in $\mathcal{T}$, and $\overline{\boldsymbol{q}_K \boldsymbol{q}_{\text{fin}}}$, constitutes the configuration space trajectory $\boldsymbol{q}(t)$, $t \in [\bar{t}, t_{\text{fin}}]$, that is returned by the LMP as the final local plan. Otherwise, similarly to other cases of motion generation failures, the procedure starts a new validation attempt.

3.3.3 Deadlock management

Until now, we have considered that, at the generic LMP invocation, the planning zone has always a fixed geometry (e.g., a sphere of a given radius), while its location is determined by the last configuration that the humanoid will reach at the end of the simultaneously executed local plan. Such strategy provides LMP only with local information about the environment. In principle, consecutive invocations of LMP might generate movements that enforce the robot to repeatedly navigate among same regions of the workspace, leading to full-fledged deadlock situations. Such problem can be easily eliminated by allowing LMP to keep memory of previously considered planning zones. To this purpose, at the i-th invocation, LMP appropriately instantiates the planning zone $\mathcal{P}_i$ as the union of the current local one $\tilde{\mathcal{P}}_i$ and all the previously considered, i.e.,

$$\mathcal{P}_i = \cup_{j=0}^{i} \tilde{\mathcal{P}}_j. \tag{3.9}$$

An illustration of this strategy is provided in Fig. 3.9.

Depending on the chosen strategy for defining the planning zone, two versions of LMP - namely, *without* and *with memory* - can be obtained. In Sect. 3.3.4, we will show a comparison of these two versions in a deadlock-prone scenario.

3.3.4 Planning experiments

In this section, we present simulations obtained in V-REP with our C++ implementation of the proposed anytime planning framework. The video available at the link `https://youtu.be/ECdDM-us0-k` shows the working principle of the approach and clips of part of the simulations presented in the following.

We consider three different scenarios of increasing complexity. In all of them, NAO is assigned a reaching task consisting in grasping with the right hand a ball

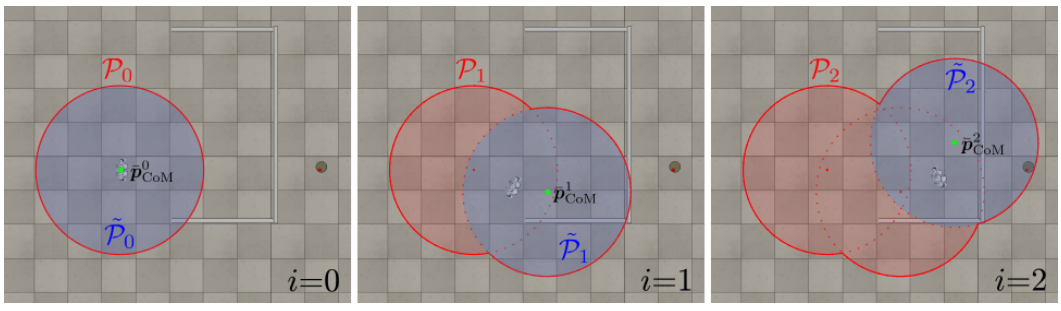

Figure 3.9. LMP with memory: at the i-the invocation, the planning zone $\mathcal{P}_i$ (whose boundary is shown in red) is the union of the current local one $\tilde{\mathcal{P}}_i$ (blue area), centered at the CoM position $\bar{\boldsymbol{p}}^i_{\text{CoM}}$ (green dot) specified by the final configuration $\bar{\boldsymbol{q}}_i$ on the current plan, and all the previously considered.

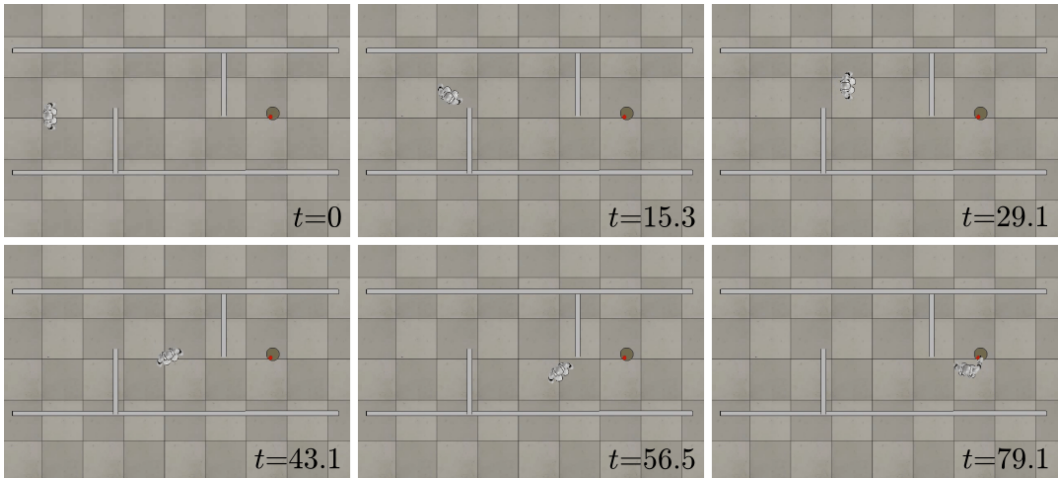

Figure 3.10. Scenario 1: the robot correctly navigate a corridor including two narrow passages using LMP without memory.

placed on a table that is outside its initial workspace. The set of CoM primitives is defined as $U = \{\texttt{free\_CoM} \cup U^D_{\text{CoM}}\}$.

At any LMP invocation, the provided time budget ΔT_P is split between the lazy and validation stages as

$$\begin{aligned} \Delta T^L_P &= \overline{\Delta T}^L_P, \\ \Delta T^V_P &= \Delta T_P - \Delta T^L_P, \end{aligned}$$

where $\overline{\Delta T}^L_P = 3.5$ s. The initial time budget $\overline{\Delta T}_P$ is set to 5 s. Furthermore, the lazy stage checks collisions using a simplified occupancy volume consisting in a cylindrical bounding box having radius and height equal to 0.16 and 0.56, respectively.

In the first scenario (see Fig.3.10), in order to complete the task, the robot must navigate a corridor in which two walls create two consecutive narrow passages. For this simple scenario, we used the version without memory of LMP, i.e., at each invocation the planning zone consists of a sphere having radius 1.25 m centered at the CoM position of the robot at the final configuration of the current local plan. At the beginning, a local plan allowing the robot to manage the first passage is produced. During its execution (snapshots 2 and 3), the next local plan is computed; this allows to correctly manage the second passage as well (snapshots 4 and 5), while simultaneously computing the final local plan through which the robot can complete the task (snapshot 6). For this scenario, the basic framework that we presented in Sect. 3.2, in order to find a solution, requires almost two minutes; during this time the robot must wait, and will be allowed to start moving only once a solution is found. The presented anytime framework eliminates such problem, and the robot starts moving after the initial time budget $\overline{\Delta T}_P$.

The second scenario (see Fig. 3.9) contains three walls that create a dead-end. Such situation allows to exhibit the benefit of keeping memory of previously considered planning zones. To this end, we compare the two versions of LMP (without and with memory). For the version without memory, the planning zone is defined as in the previous scenario, while for the version with memory, at each

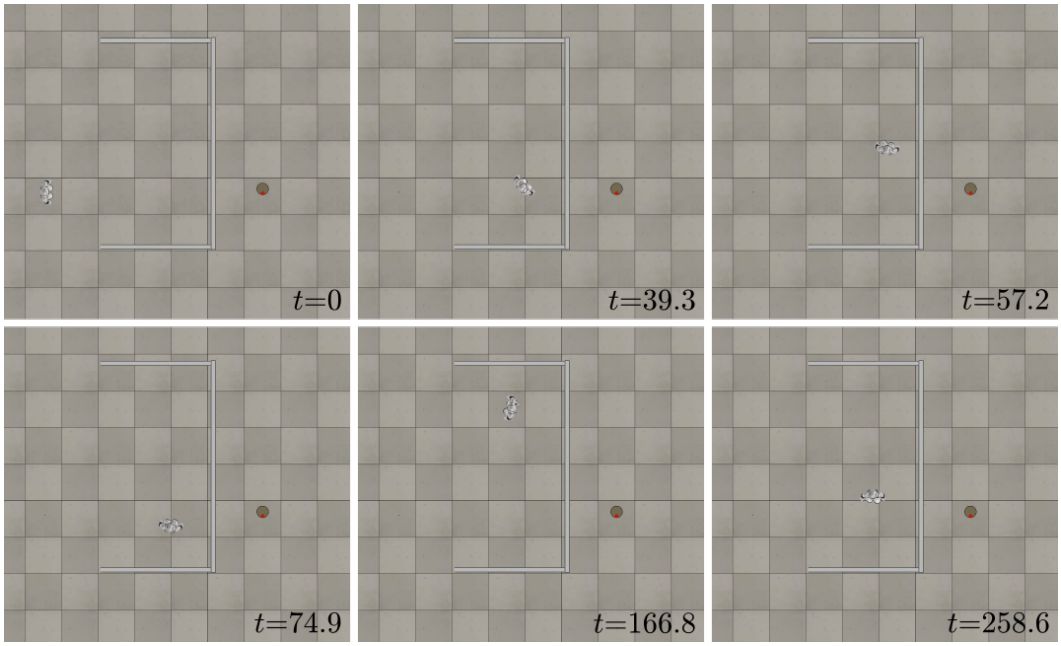

Figure 3.11. Scenario 2: the robot gets trapped in the dead-end using LMP without memory.

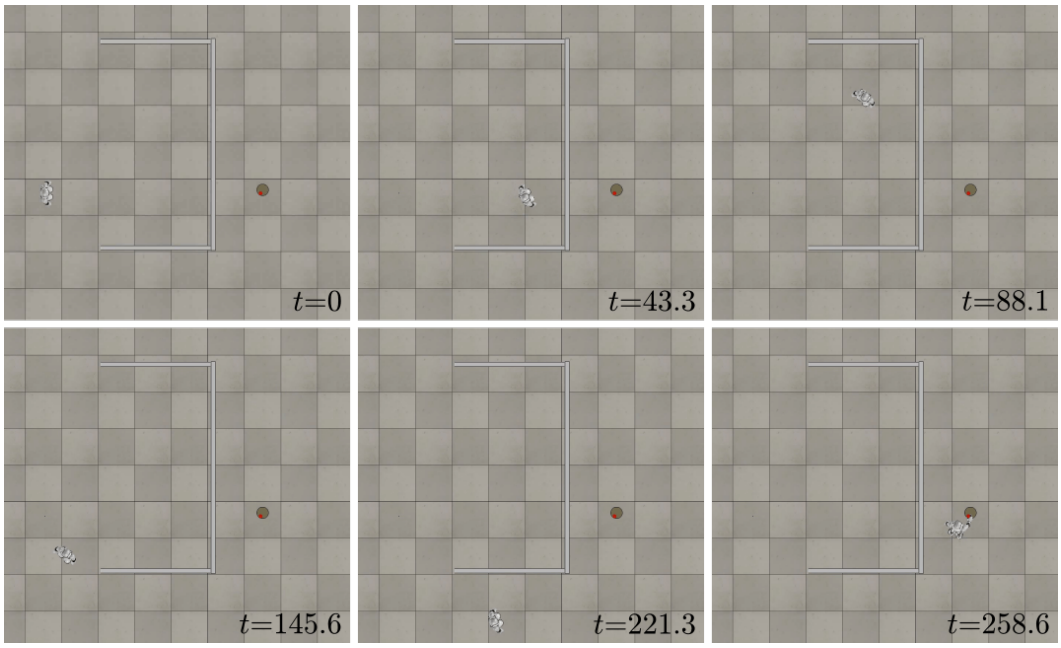

Figure 3.12. Scenario 2: the robot exits the dead-end and correctly completes the task using LMP with memory.

invocation, LMP instantiate the planning zone as in (3.9). Note that, in any case, the dead-end is not entirely contained in the initial planning zone (see Fig. 3.9, left). Using the version without memory (see Fig. 3.11), the robot moves towards the red ball until it reaches the proximity of the corner formed by two walls (snapshot

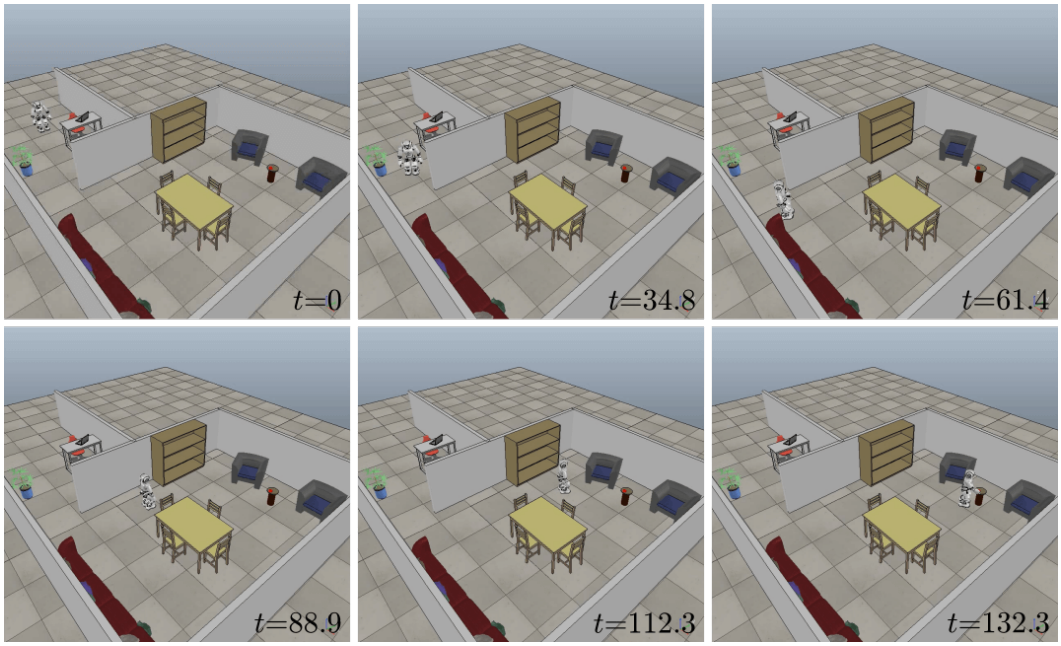

Figure 3.13. Scenario 3: the robot moves from one room to another and completes the task using LMP with memory.

2); at this point LMP, in order to avoid collisions with the walls, generates a local plan leading the robot to turn left (snapshot 3). Then, backward movements are generated (snapshot 4), since they represent the local plan allowing the robot to approach the goal as much as possible. Such movements bring the robot in already visited zones of the workspace, from which LMP generates again forward movements (snapshot 5) to approach the boundary of the current planning zone. This loop continues, trapping the robot within the dead-end, and precluding it to fulfill the task. The version with memory results effective (see Fig. 3.12). As before, the robot starts by moving towards the ball until it reaches the proximity of the wall obstructing the passage (snapshot 2), and then turns left. Once the dead-end is included in the planning zone, LMP produces a local plan that leads the robot outside the closed region (snapshots 3 and 4). It is important to emphasize that this behavior results automatically from the candidate local plan chosen by the validation stage. Sequences of dynamic steps are then generated (snapshot 5), allowing the robot to approach the goal and finally complete the task (snapshot 6).

In the third scenario (see Fig. 3.13), in order to complete the task, the robot must move from one room to another, while avoiding collisions with different obstacles (e.g., walls, chairs, tables). Using the version with memory of LMP, even in this cluttered scenario, the proposed anytime planning framework proved to be effective in generating feasible whole-body motions.

Table 3.2 collects some performance data observed in each scenario[12]. The three planning experiments required three, eight and five planning intervals (i.e., LMP

[12]For the second scenario, we include only data observed using the version with memory of LMP, as the version without memory revealed unsuccessful.

scenario	i	time budget (s)	# candidate local plans	motion duration (s)
1	0	5	3	29.1
	1	29.1	10	27.4
	2	27.4	31	22.6
2	0	5	41	26.9
	1	26.9	3	28.8
	2	28.8	9	20.7
	3	20.7	2	75.0
	4	75.0	17	21.0
	5	21.0	18	31.5
	6	31.5	17	26.4
	7	26.4	12	28.3
3	0	5	7	27.0
	1	27.0	3	26.4
	2	26.4	9	32.7
	3	32.7	14	26.4
	4	26.4	11	19.8

Table 3.2. Planner performance data.

invocations), respectively. For each invocation, the table reports: the assigned time budget, the number of candidate plans resulting at the end of the lazy stage, and the duration of the generated local plan.

3.4 Sensor-based whole-body planning framework

In this section, we address the motion planning problem described in Sect. 3.1.2 removing the assumption of known environment.

In particular, a humanoid robot is assigned a reaching task (T2) in an unknown environment. We assume that the robot is equipped with a head-mounted depth camera through which it can continuously acquire information about its surroundings while moving. Furthermore, we assume that the robot can perfectly localize itself with respect to a fixed inertial frame using an external module[13].

A solution for this problem consists of a configuration space trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, that satisfies the requirements R1-R4, constituted by a succession of partial solutions that are planned on-line based on the available information about the environment.

Sect. 3.4.1 proposes a sensor-based scheme to address the given problem, while Sects. 3.4.2, 3.4.3, 3.4.4 describe in detail the involved modules. Sect. 3.4.5 presents simulations showing the effectiveness of the proposed approach for generating feasible whole-body motions allowing the humanoid to fulfil a reaching task in an unknown environment.

[13]Removing such assumption is part of our future work.

3.4.1 Framework overview

To address the described problem, we propose a framework constituted by three cooperating modules, namely the *mapping*, *planning*, and *execution* modules, that are in charge of, respectively, incrementally build an environment map $\mathcal{M}$, computing feasible whole-body motions $\boldsymbol{q}(t)$, and sending commands to the humanoid actuators.

At the beginning, the mapping module generates a 3D map $\mathcal{M}(t_{\text{ini}})$ according to information that the robot can acquire at its initial configuration $\boldsymbol{q}_{\text{ini}}$. Then, an initial plan, i.e., a whole-body motion $\boldsymbol{q}(t)$ that is feasible within $\mathcal{M}(t_{\text{ini}})$, is computed by the planning module. Once such initial plan is computed, the robot starts performing it, according to the commands sent by the execution module. Hereafter, we refer to the time instant in which the robot starts moving as t_0.

Then, the three modules start to run in parallel:

- The mapping module continuously updates the environment map $\mathcal{M}$ according to the newly acquired information.

- The execution module realizes the planned whole-body motion $\boldsymbol{q}(t)$, with $t \in [t_0, \bar{t}]$ and $\bar{t}$ its final time instant. Henceforth, we will refer to such previously planned whole-body motion as the *current plan*. At the generic time instant t_c, the robot, under the action of the execution module, will be in the configuration $\boldsymbol{q}(t_c)$ specified by the current plan. Consequently, the current plan will be composed by two portions: the already executed part in the time interval $[t_0, t_c]$, and the remaining part that defines the future motion of the robot in the time interval $(t_c, \bar{t}]$.

- The planning module repeatedly extends the current plan. To this end, it works iteratively. At each iteration, the produced extension of the current plan consists in a whole-body motion that starts at its final configuration $\bar{\boldsymbol{q}} = \boldsymbol{q}(\bar{t})$, and is feasible according to the currently available information contained in the map $\mathcal{M}$. To extend the current plan before the robot completes the previously planned motion, each iteration of the planning module must complete within the remaining time on the current plan at most.

In the following we describe separately the three modules.

3.4.2 Mapping module

The mapping module is in charge of continuously integrating information gathered by the depth camera into the environment map $\mathcal{M}$. To take full advantage of the humanoid capabilities in scenarios containing complex shaped obstacles, we maintain the map $\mathcal{M}$ in the form of a 3D occupancy grid, which models both free and occupied space, and, at the same time, implicitly models unknown space, just by missing information.

In our framework, the map $\mathcal{M}$ is kept as a volumetric octree-based map, called *OctoMap*, whose characteristics perfectly match the needs described above. This representation compactly models free and occupied areas by voxels, each one containing

an occupancy probability that can be dynamically updated[14].

At the initial time instant t_{ini}, the map $\mathcal{M}(t_{\text{ini}})$ is

$$\mathcal{M}(t_{\text{ini}}) = \mathcal{R}(\boldsymbol{q}_{\text{ini}}) \cup \mathcal{M}_0(t_{\text{ini}}) \tag{3.10}$$

where $\mathcal{R}(\boldsymbol{q}_{\text{ini}})$ represents the free volume that the robot body occupies (computed on the basis of proprioceptive sensors), and $\mathcal{M}_0(t_{\text{ini}})$ is the initial knowledge at the configuration $\boldsymbol{q}_{\text{ini}}$, that may consist of a limited exogenous knowledge of the starting location, further enlarged by collecting information through, e.g., a pan-tilt motion performed on the spot.

The map is then updated while the robot moves in the unknown environment. At each reading, the camera provides a depth image in which each pixel contains the distance between the 3D point in the Cartesian space to which the pixel refers to, and the camera image plane. Such depth image indicates a beam of rays originating in the camera origin and ending at the observed Cartesian points. Given the coordinate z_f of the camera far clipping plane on the principal axis, if the depth of a certain pixel is less than z_f, the endpoint of the corresponding ray is on the surface of an obstacle. In this case, the voxel corresponding to the endpoint is updated in $\mathcal{M}$ as occupied, and all the other voxels along the ray are updated as free; otherwise, all the voxels along the whole ray are updated as free.

3.4.3 Planning module

The planning module generates the whole-body motions that the robot performs through the execution module. As already mentioned, this module firstly generates an initial plan within the initial environment map, and then enters an iterative phase where extensions of the current plan are computed within updated versions of the map provided by the mapping module described in Sect. 3.4.2.

To this end, the planning module works by repeatedly calling a local motion planner (LMP) that consists in an adaptation, for the case of unknown environment, of that presented in Sect. 3.3.2. The only difference is that, in the case of unknown environment, the LMP is provided with an environment map (in the form discussed in Sect. 3.4.2) that replaces the planning zone used in Sect. 3.3.2. Each invocation of the LMP is in charge of producing a whole-body motion, henceforth referred to as a *local plan*, that starts at the final configuration $\bar{\boldsymbol{q}}$ of the current plan $\boldsymbol{q}(t)$, $t \in [t_0, \bar{t}]$, provides an extension of it for further exploration aimed at completing the task, and is feasible within the currently available map. Such local plan must be computed before the time instant $\bar{t}$, in such a way the current plan can be extended before the robot reaches its final configuration $\bar{\boldsymbol{q}}$. This guarantees that the robot moves continuously, without the need to stop for planning its future motions. Thus, each invocation of the LMP is allowed to run for a certain time budget that is limited by the remaining time on the current plan.

The pseudocode of the planning module is given in Algorithm 3. It starts by invoking LMP that is in charge of producing, within a predefined time budget $\overline{\Delta T}_P$, the first local plan $\boldsymbol{q}(t)$, $t \in [t_0 = t_{\text{ini}} + \overline{\Delta T}_P, t_1 = t_0 + \Delta T_{E,0}]$, with $\Delta T_{E,0}$ its

[14] More details about this powerful mapping framework can be found in [74], while the software is available as an open-source C++ library at `http://octomap.github.io`.

Algorithm 3: Planning Module

$\boldsymbol{q}(t),\, t \in [t_0, t_1] \leftarrow \text{LMP}(\boldsymbol{q}_{\text{ini}},\, \overline{\Delta T}_P,\, \mathcal{M}_{\text{ini}})$;
SleepFor($\alpha_P \Delta T_{E,0}$);
$\bar{\boldsymbol{q}}_1 \leftarrow \boldsymbol{q}(t_1)$;
$i \leftarrow 1$;
while $\boldsymbol{f}(\bar{\boldsymbol{q}}_i) \neq \boldsymbol{y}_M^*$ **do**
  $\Delta T_{P,i} \leftarrow \text{ComputeTimeBudget}(t_c,\, t_i)$ (3.11);
  $\mathcal{M}_{P,i} \leftarrow \mathcal{M}(t_c)$;
  $\boldsymbol{q}_i(t),\, t \in [t_i, t_{i+1}] \leftarrow \text{LMP}(\bar{\boldsymbol{q}}_i,\, \Delta T_{P,i},\, \mathcal{M}_{P,i})$;
  $\boldsymbol{q}(t),\, t \in [t_0, t_{i+1}] \leftarrow \text{Concatenate}(\boldsymbol{q}(t),\, \boldsymbol{q}_i(t))$;
  $i \leftarrow i + 1$;
  $\bar{\boldsymbol{q}}_i \leftarrow \boldsymbol{q}(t_i)$;
end

duration, that starts at the initial robot configuration $\boldsymbol{q}_{\text{ini}}$, and is guaranteed to be feasible in the initial map $\mathcal{M}(t_{\text{ini}})$. Once such plan is computed, the robot starts executing it and the planning module waits for a portion $\alpha_P \Delta T_{E,0}$ of its duration before entering the iterative phase. The rationale beyond this choice is to avoid an immediate replanning on the same map used to compute the initial plan. In fact, since at this point the robot is not moved yet, the mapping module has not gathered new information about the environment, and replanning would not provide any plan extension.

At the i-th iteration, the planning module invokes LMP for computing an extension of the current plan $\boldsymbol{q}(t)$, $t \in [t_0, t_i]$. Let t_c be the time instant at the beginning of the i-th iteration of the planning module; at the corresponding invocation, LMP is provided with:

- The final configuration $\bar{\boldsymbol{q}}_i = \boldsymbol{q}(t_i)$ of the current plan.
- The *planning map* $\mathcal{M}_{P,i}$, that coincides with the currently available map $\mathcal{M}(t_c)$. The planning map $\mathcal{M}_{P,i}$ is then fixed during the i-th replanning, while the map $\mathcal{M}$ is continuously updated by the mapping module.
- The time budget $\Delta T_{P,i}$, that is computed as a portion of the remaining time on the current plan
$$\Delta T_{P,i} = \alpha_P (t_i - t_c) \tag{3.11}$$
where $\alpha_P \in (0, 1)$ is a design parameter. Note that, the larger (smaller) α_P, the less (more) frequent the replanning, the more (less) enlarged the planning map $\mathcal{M}_{P,i}$ w.r.t. the previous one $\mathcal{M}_{P,i-1}$. Choosing a large α_P potentially allows to find longer local plans. On the other hand, a small α_P reflects in a more reactive behavior.

This invocation of LMP, that will last at most $\Delta T_{P,i}$, produces a local plan $\boldsymbol{q}_i(t)$, $t \in [t_i, t_{i+1} = t_i + \Delta T_{E,i}]$, with $\Delta T_{E,i}$ its duration, that starts at the final configuration $\bar{\boldsymbol{q}}_i = \boldsymbol{q}(t_i)$ of the current plan $\boldsymbol{q}(t)$, $t \in [t_0, t_i]$, and is feasible within the map $\mathcal{M}_{P,i}$. Note that the duration $\Delta T_{E,i}$ of the computed local plan is not assigned, as it is autonomously determined by LMP. The current plan is then extended by concatenating the computed local plan to it.

This iterative procedure terminates when LMP computes a local plan such that the desired set-point is eventually reached, i.e., $\boldsymbol{k}(\bar{\boldsymbol{q}}_i) = \boldsymbol{y}_R^*$ with $\boldsymbol{k}(\bar{\boldsymbol{q}}_i)$ the end-effector position at the last configuration $\bar{\boldsymbol{q}}_i$ of the extended plan. In this case, the planning module stops and the execution module continues until the humanoid completes the task.

3.4.4 Execution module

This module is in charge of sending the joint commands to the robot low level controllers. At each time instant, such commands are all taken from the current plan $\boldsymbol{q}(t)$, $t \in [t_0, \bar{t}]$, except for the robot yaw neck joint that is used to directly control the pan angle of the depth camera, rigidly attached to the head. With the aim of favoring the mapping module in enlarging the environment map in the area where the extension of the current plan has to be computed through the planning module, the yaw neck joint velocity $\dot{q}_{\mathrm{y}}$ is computed on-line so as to make the robot looking in the direction of the location that it will reach at the end of the current plan.

At the generic time instant, the robot yaw neck joint velocity $\dot{q}_{\mathrm{y}}$ is computed using a simple proportional control

$$\dot{q}_{\mathrm{y}} = K_{\mathrm{y}}(q_{\mathrm{y}}^d - q_{\mathrm{y}}) \tag{3.12}$$

where q_{y}^d and q_{y} are, respectively, the desired and current yaw joint positions. Given the final configuration $\bar{\boldsymbol{q}} = \boldsymbol{q}(\bar{t})$ on the current plan, and the configuration $\boldsymbol{q}$ specified for the current time instant, q_{y}^d is defined as the angle between the robot sagittal axis at $\boldsymbol{q}$ (that can be readily identified through the subvector $\boldsymbol{q}_{\mathrm{CoM}}$), and the line joining the origins of the CoM frames at $\boldsymbol{q}$ and $\bar{\boldsymbol{q}}$. K_{y} is a positive scalar gain. Note that, when the current plan is extended, its final configuration $\bar{\boldsymbol{q}}$ changes, and consequently also q_{y}^d.

3.4.5 Planning experiments

In this section, we present simulations obtained in V-REP with our C++ implementation of the proposed sensor-based planning framework. We consider three different scenarios of increasing complexity. In all of them, NAO is assigned a reaching task consisting in grasping with the right hand a ball placed on a table that is outside its initial workspace. The robot is equipped with a Kinect camera that provides 320×240 depth images. To maintain the 3D environment map, we use an Octomap with a resolution of 5 cm. The set of CoM primitives is defined as $U = \{\texttt{free\_CoM} \cup U_{\mathrm{CoM}}^D\}$.

At any LMP invocation, the provided time budget ΔT_P is split between the lazy and validation stages as

$$\begin{aligned} \Delta T_P^L &= \alpha_{LMP} \Delta T_P, \\ \Delta T_P^V &= \Delta T_P - \Delta T_P^L, \end{aligned}$$

where α_{LMP} is set to 0.6. The parameter α_P in (3.11) is set to 0.5. Furthermore, The lazy stage checks collisions at footsteps level using the occupancy volume of the robot feet.

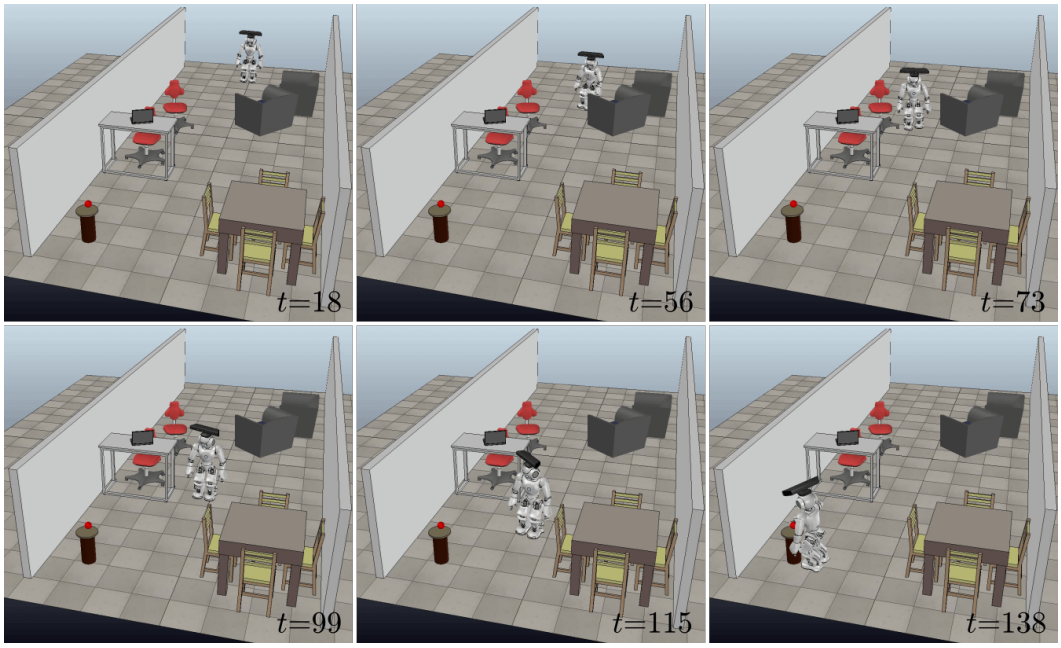

Figure 3.14. Scenario 1: the robot completes the task after navigating a cluttered environment.

At the beginning, the map $\mathcal{M}_0(t_{\text{ini}})$ consists of an initial knowledge provided in advance within a cylindrical area of radius 0.5 m and height 1.2 m centered at $\boldsymbol{p}_{\text{CoM}}^{\text{ini}}$, and further knowledge acquired through an initial pan-tilt motion, whose duration is 18 s. After this phase, the planning module starts to compute an initial local plan within this map through the first invocation of LMP, with a predefined time budget of $\overline{\Delta T}_P = 15$ s.

In the first scenario (see Fig. 3.14), different kinds of obstacles (chairs, tables and sofas) obstruct the path between the robot at its initial configuration, at which only portions of few obstacles can be seen, and the destination. Once the initial local plan is available, the robot starts to perform it through the execution module, while the planning module computes future motions. At each invocation of LMP, previously unknown obstacles are taken into account, and collisions are correctly avoided. The mapping-planning-execution cycle is repeated until the robot grasps the red ball (last snapshot). The overall robot motion (constituted by a sequence of dynamic steps and concluded by the `free_CoM` primitive to finally grasp the ball) results fluid, without the need for the robot to stop in any case. This proves the on-line performances of the proposed framework. Furthermore, we emphasize the capability of our LMP in managing narrow passages. In fact, taking into account the 3D structure of both the environment and the robot, LMP is able to produce motions allowing the robot to correctly navigate the strict free space between the sofa and the chair (third snapshot). Note that, due to the complex shape of the chair, approaches that involve bounding boxes (e.g., [66]) or 2D projections of swept volumes (e.g., [67]) to check collisions would fail to find feasible motions in this case.

In the second scenario (see Fig. 3.15), a wall totally occludes the red ball to the robot at its initial configuration. While executing the first local plan, the robot

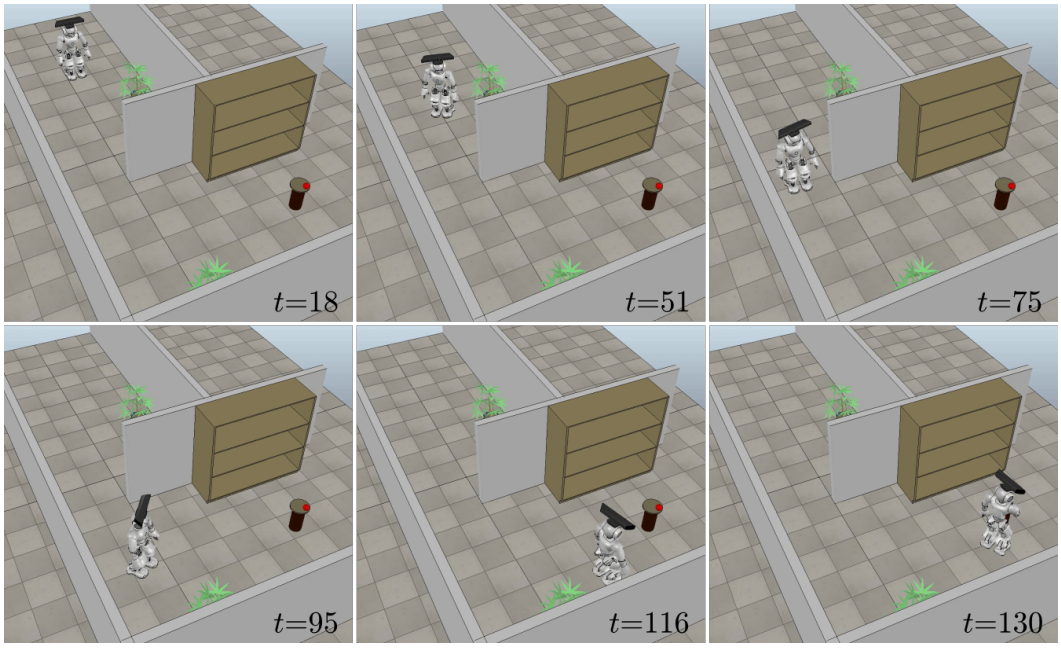

Figure 3.15. Scenario 2: the robot reaches the red ball after turning the wall that initially occluded it.

moves towards the frontier of the initial map. While turning the wall (third and fourth snapshots), the mapping module enlarges the map with newly discovered information, and the planning module accordingly generates task-oriented motions. Note that, in this case the mapping module effectively exploits the ability of the robot to look in the direction of the current plan end, taking full advantage of the strategy described in Sect. 3.4.4. Again, the overall motion of the robot results fluid.

In the third scenario (see Fig. 3.16), we show the capability of the proposed framework of recovering from dead-ends. At its initial configuration (first snapshot), the robot cannot see the wall obstructing the straight path to the destination, as it is outside the initial field of view of the camera. Among all the candidate local plans bringing towards unexplored areas, LMP chooses, and validates, the one that allows the robot to approach as much as possible the destination. This local plan leads the robot to proceed ahead the wall, entering a dead-end (second snapshot), that is incrementally discovered through the mapping module while walking. Once the dead-end is included in the new map, LMP generates an extension of the current plan that allows the robot to exit the closed space and proceed towards the free boundary of the map (third and fourth snapshots). We emphasize that such effective behaviour is the result of the LMP intrinsic bias towards unexplored areas, which is due to the strategy used for selecting the best candidate local plan among those resulting at the end of the lazy stage (see Sect. 3.3.2). Once the robot is outside the dead-end, it proceeds towards the destination, appropriately planning its motions according to continuously acquired information. Also in this case, the task is completed using the `free_CoM` primitive.

Fig. 3.17 shows the final environment maps in the three different scenarios. Table 3.3 collects some performance data of the planning module in each scenario.

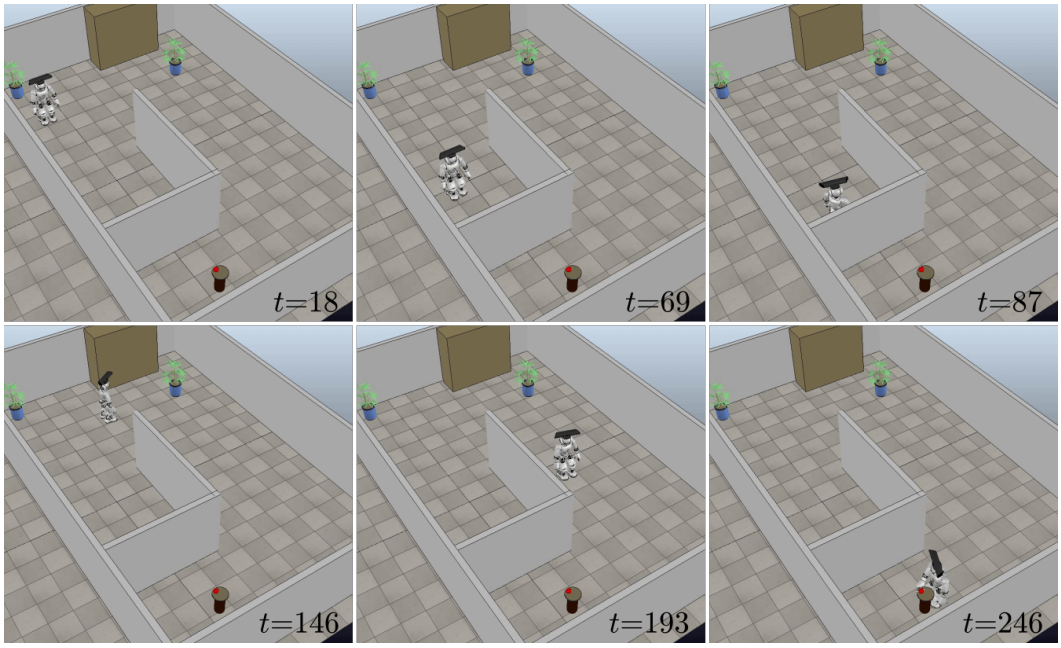

Figure 3.16. Scenario 3: the robot explores (and recovers from) a dead-end before completing the task.

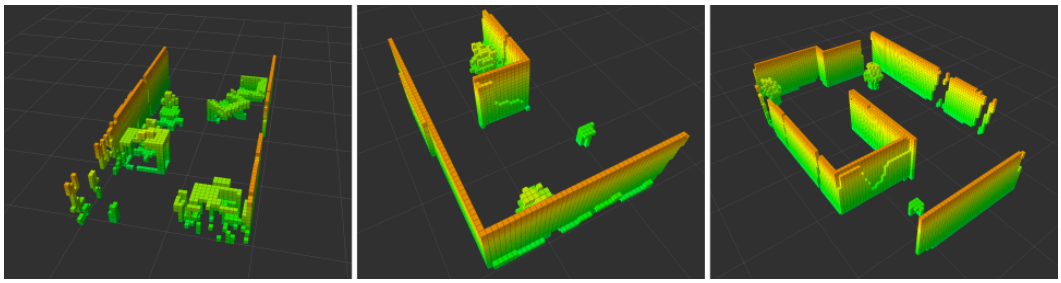

Figure 3.17. The final environment maps in the three scenarios.

The three planning experiments required four, five, and ten invocations of LMP, respectively For each invocation, the table reports: the assigned time budget resulting from (3.11), the number of candidate plans resulting at the end of the lazy stage, and the duration of the generated extension of the current plan. Note that, in the third scenario, in two cases LMP returned no extension of the current plan. This means that, among the candidate local plans produced by the lazy stage, the best one consists in a branch of the tree such that only the configuration in the root vertex is inside the current map. This automatically postpones the extension of the current plan to the subsequent LMP invocation, which will work with a new time budget on a new map that has been updated during the robot motion.

We encourage the reader to watch the video available at the link `https://youtu.be/6oLl2msrnIc` to better appreciate the effectiveness of the generated motions.

scenario	i	time budget (s)	# candidate local plans	motion duration (s)
1	0	15	19	52.45
	1	13.11	29	17.65
	2	15.38	5	11.65
	3	13.52	55	27.5
2	0	15	7	51.4
	1	12.44	30	12.85
	2	12.64	35	13.15
	3	12.90	24	5.35
	4	9.12	29	17.6
3	0	15	4	47.05
	1	11.76	5	49.75
	2	30.76	3	0
	3	13.39	13	34.15
	4	24.76	2	0
	5	12.38	67	26.05
	6	19.22	95	15.25
	7	17.23	77	17.05
	8	17.14	89	13.75
	9	15.45	4	5.6

Table 3.3. Planner performance data.

3.5 Conclusions

In this chapter we proposed a complete method for planning whole-body motions of a humanoid robot that must execute a task that implicitly requires stepping in an environment populated by obstacles. For this problem we presented three planning schemes for, respectively, planning offline a complete solution, planning online in known environment in the presence of time limitations, and planning online in unknown environment exploiting incrementally acquired information. The overall method relies on the concept of CoM movement primitives that are representative of elementary humanoid actions. With this strategy, humanoid whole-body motions are generated directly in the configuration space by concatenating single whole-body motions, each one realizing a certain primitive selected from a precomputed catalogue and, simultaneously, a portion of the assigned task. Thanks to this strategy, the whole-body structure of the humanoid and the 3D nature of the environment (either it is known or unknown) are fully exploited, allowing the generation of solutions that, by construction, guarantees balance and collision avoidance. We have shown via simulations that the offline version is able to generate solutions for a variety of tasks (i.e., navigation, reaching, manipulation and visual tasks), and that the online versions (both anytime and sensor-based) can generate sensible motions in cluttered environments for fulfilling a reaching task. Application of the online versions to the other kinds of tasks is part of our future work.

A limitation of the strategy based on CoM movement primitives at the core of our method is the difficulty in handling scenarios with stairs. In principle, primitives for climbing and descending stairs may be included in the precomputed catalogue; however, such primitives should account for the infinite possible values of the height of the stairs, that in general is arbitrary and possibly not a priori known. Since including primitives for any kind of stair is clearly impracticable, a possible solution consists in using 'deformable' primitives, whose reference CoM and swing foot trajectories can be modified according to the planning needs. In general, in the case of navigation tasks, these difficulties can be overcome by adopting a two-stages approach in which footstep placement and CoM movement are planner in two sequential phases. Chap. 4 will explicitly consider this case.

Another limitation of the proposed method is that it assumes that the environment is static also when motions are planned online, i.e., in the anytime and sensor-based versions. In fact, at each planning stage, the aim is only to extend the current plan, without allowing any modification of the latter. This is clearly an issue in environments containing dynamic obstacles, such as humans, that moves along unpredictable trajectories. Such context will be addressed in Chap. 7.

Future work will address: (*i*) the implementation of the proposed method on the real humanoid; (*ii*) the extension to the case of tasks requiring mobile manipulation of heavy objects; (*iii*) the replacement of the mapping module in the sensor-based framework with a full-fledged SLAM module; (*iv*) the inclusion of a second-order motion generation scheme as in [75] to take into account also torque bounds; (*v*) the reformulation of the strategy for generating the head motion (Sect. 3.4.4) to explicitly maximize the information gain.

Chapter 4

Motion planning on uneven ground

One of the advantages of humanoids is the possibility of moving through complex environments, e.g., by stepping over or onto obstacles. However, while there are a large number of locomotion planning techniques for flat ground, walking on uneven surfaces poses additional challenges which make it largely an open field of research.

To deal with the complexity of such problem, an effective strategy consists in planning in two sequential stages the discrete sequence of isolated contacts with the ground, i.e., the footstep sequence, and the continuous CoM trajectory which is compatible with it and guarantees balance. As discussed in Chap. 2, this strategy prevents the inclusion of tasks in addition to the basic requirements such as avoiding collisions and keeping balance. On the other hand, it allows to effectively address navigation, also called walk-to, tasks in the case of uneven ground, which are instead difficult to tackle using the whole-body planning framework described in the previous chapter (see Sect. 4.5).

One possible approach to plan footsteps (on flat or uneven ground) is based on continuous optimization techniques, which do not restrict steps to a finite set [76]. However, these methods require an expensive pre-computation phase aimed at finding a convex approximation of the free configuration space. An efficient alternative is to consider a finite set of possible foot displacements, so that the solution will consist of a particular sequence of such elements. To search among all possible sequences, both deterministic and randomized approaches have been proposed. Examples of the first kind are A^*-based techniques [25, 77, 78], whose performance strongly depends on the chosen heuristic, which is often difficult to design. Moreover, node expansion in these techniques can be very expensive in the presence of several constraints to be verified (e.g., collision avoidance, kinematic reachability, and so on). Randomized approaches include [79] where, to ensure goal-directedness, all possible node expansions must be evaluated at each iteration, at the expense of efficiency. To overcome such a problem, an approximate swept volume is precomputed in [26] for each possible foot displacement, thus speeding up collision checking for footsteps and swinging trajectories. This approach, however, cannot be extended to the case of variable height steps because of the infinite possibilities for the foot displacement along the vertical direction.

Whatever the footstep planning method is, the humanoid CoM trajectory must be generated in such a way to be compatible with the planned footsteps and to maintain (preferably) dynamic balance. To this purpose, many methods have been proposed in which the complexity of the humanoid dynamics is reduced by using simplified models. On flat ground, many researchers adopt a simplified model known as the Linear Inverted Pendulum (LIP) [80], in which the robot is considered a single point-mass pivoting on a foot and moving at constant height above the ground. The linearity of this model has been exploited to design preview controllers [46] and MPC schemes [81]. When the robot moves on uneven ground one must remove the constant height hypothesis, leading to a nonlinear inverted pendulum model. The nonlinear problem was addressed, for example, in [82, 83]. However, keeping the linearity of the system is still an attractive option in view of the simplicity of the associated controllers and, ultimately, the possibility of an efficient real-time implementation. One way to do this consists in assuming a predefined vertical trajectory of the CoM, so that the robot can be described by a time-varying LIP, as done in [84, 85]. Another, less restrictive possibility [86] is to maintain the time-invariant LIP structure by constraining the CoM vertical motion to satisfy a certain differential equation. Related approaches [87, 88] lead to a 3D model with LIP dynamics on all three axes.

In this chapter we propose an integrated architecture for planning and executing humanoid motions to fulfil a walk-to task on uneven ground. It works in two stages: an off-line footstep planning module computes an appropriate sequence of footsteps, and an on-line gait generation module generates a variable-height CoM trajectory realizing them. We propose two versions of the footstep planner, both based on an efficient randomized strategy which, differently from existing approaches, does not involve any kind of pre-processing of the environment and allows to enforce various feasibility requirements directly in the planning phase. The first version allows to find a footstep sequence that is feasible with respect to the robot kinematic limits, while the second also accounts for the quality of such sequence. We show that the latter can incorporate different performance criteria and asymptotically find optimal solutions. Two humanoid platforms are used to show the effectiveness of the proposed method, i.e., HRP-4 in simulation and NAO in experiments.

This chapter is organized as follows. In Sect. 4.1 we formally define the problem of interest and describe the proposed architecture. Sects. 4.2 and 4.3 present the two versions of the footstep planner, together with the obtained simulation results in different scenarios. Experimental results are presented in Sect. 4.4. Concluding remarks are reported in Sect. 4.5.

4.1 Core problem and approach

The situation of interest in this chapter is shown in Fig. 4.1. A humanoid robot is assigned a *walk-to* locomotion task to a desired goal region $\mathcal{G}$ in a *world of stairs*, a specific kind of uneven ground composed by horizontal patches located at different heights. Depending on its elevation with respect to the neighboring areas, a patch may be accessible for the humanoid to climb on from an appropriate direction, or else represent an obstacle to be avoided. Some low-height patches may be shaped

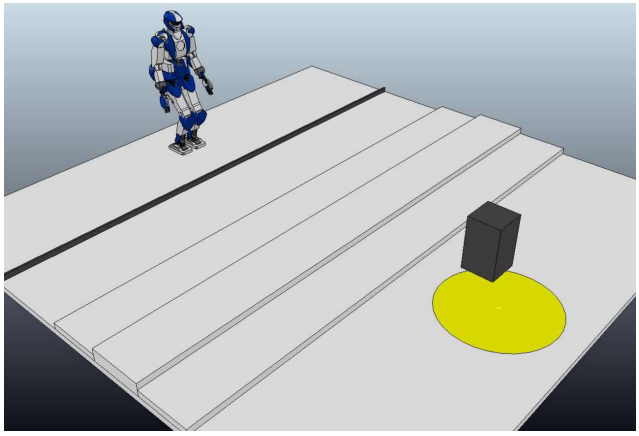

Figure 4.1. An instance of the considered problem. To reach the goal region $\mathcal{G}$ (in yellow), the humanoid must go over the black bar obstacle, climb and descend the staircase, and avoid the black box.

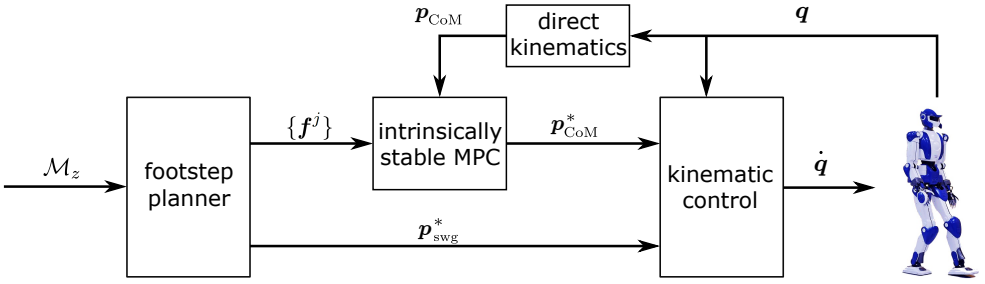

Figure 4.2. Block scheme of the proposed approach.

in such a way that the humanoid may decide to go over (rather then stepping on) them; for example, this is the case of the long bar obstacle in Fig. 4.1.

A natural choice for representing the considered kind of ground is a 2.5D grid map of equally-sized cells, also called *elevation map* [89]. We will denote this map by $\mathcal{M}_z$, and assume that it is known in advance so that whenever needed it can be queried as $z = \mathcal{M}_z(x, y)$, to provide the height of the ground at the cell having coordinates (x, y).

To solve the given problem, we propose an integrated motion planner/controller whose block scheme is shown in Fig. 4.2. Denote by $\boldsymbol{f} = (x_\text{f}, y_\text{f}, z_\text{f}, \theta_\text{f})^T$ the generic pose of a certain foot, with x_f, y_f, and z_f representing its position and θ_f its orientation[1]. Let $\boldsymbol{f}_L^{\text{ini}}$ and $\boldsymbol{f}_R^{\text{ini}}$ be the pose of, respectively, the left and right foot at the initial configuration of the robot. For simplicity, in the following we assume that the robot, during locomotion, performs steps alternating left and right support foot, and that it starts walking by swinging the left foot.

In the proposed scheme, an off-line footstep planner is in charge of finding first a *footstep plan* consisting in an appropriate sequence of footsteps $\mathcal{S}_f = \{\boldsymbol{f}^1 = \boldsymbol{f}_L^{\text{ini}}, \boldsymbol{f}^2 = \boldsymbol{f}_R^{\text{ini}}, \ldots, \boldsymbol{f}^n\}$ leading to the desired location, i.e., $\boldsymbol{f}^n \in \mathcal{G}$, together with a sequence of associated swing foot trajectories $\mathcal{S}_p = \{\boldsymbol{p}^1_{\text{swg}}, \boldsymbol{p}^2_{\text{swg}}, \ldots, \boldsymbol{p}^{n-2}_{\text{swg}}\}$. In the

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

sequence $\mathcal{S}_f$, the generic element $\boldsymbol{f}^j$ is the pose of the j-th footstep, while in the sequence $\mathcal{S}_p$, the generic element $\boldsymbol{p}^j_{\text{swg}}$ is the trajectory leading the foot from $\boldsymbol{f}^j$ to $\boldsymbol{f}^{j+2}$. Note that the number n of footsteps in the sequence $\mathcal{S}_f$ is not pre-assigned, as it will be a byproduct of the planner.

The footstep plan must be feasible (in a sense formally defined in the following) w.r.t. the characteristics of both the environment and the robot. In Sect. 4.2 we present a method for computing feasible footstep plans. In addition to feasibility, the quality of the footstep plan is important as well. To this purpose, in Sect. 4.3, we present an extension of the method that is able to (asymptotically) find optimal footstep plans w.r.t. a desired criterion.

Once the footstep plan has been generated, the footstep sequence $\mathcal{S}_f$ is passed to an on-line gait generation block based on an intrinsically stable MPC, which computes a variable-height CoM trajectory $\boldsymbol{p}^*_{\text{CoM}}$ compatible with the sequence. In particular, this module uses the technique introduced in [90] (see also Appendix A.2). It uses a 3D LIP-like model where the CoM height can vary under the appropriate differential constraint, thus retaining a linear structure for the problem; moreover, it relies on the inclusion of an explicit stability constraint in the MPC formulation to ensure that the resulting variable-height CoM trajectory is bounded w.r.t. the ZMP. The swing foot trajectory $\boldsymbol{p}^*_{\text{swg}}$ at any instant is defined by the appropriate subtrajectory $\boldsymbol{p}^j_{\text{swg}}$ of sequence $\mathcal{S}_p$.

Finally, the reference trajectories $\boldsymbol{p}^*_{\text{CoM}}$ and $\boldsymbol{p}^*_{\text{swg}}$ are sent to a standard pseudoinverse-based kinematic controller to compute joint commands $\dot{\boldsymbol{q}}$ for the robot. Proprioceptive feedback is used for both MPC and kinematic control.

4.2 Basic footstep planner

The basic capability the footstep planner must exhibit is that of generating feasible plans. Sect. 4.2.1 formally defines the requirements that the footstep plan must satisfy in order to be considered feasible, while Sect. 4.2.2 proposes a randomized algorithm for computing a feasible footstep plan. Sect. 4.2.3 presents simulations obtained with the proposed approach.

4.2.1 Problem formulation

As anticipated in Sect. 4.1, the basic footstep planning problem consists in finding a sequence of footsteps $\mathcal{S}_f = \{\boldsymbol{f}^1, \ldots, \boldsymbol{f}^n\}$ leading to the desired location, i.e., $\boldsymbol{f}^n \in \mathcal{G}$, together with a sequence of associated swing foot trajectories $\mathcal{S}_p = \{\boldsymbol{p}^1_{\text{swg}}, \ldots, \boldsymbol{p}^{n-2}_{\text{swg}}\}$, that are *feasible*, i.e., each element $\boldsymbol{f}^j$ and $\boldsymbol{p}^j_{\text{swg}}$ in the sequences $\mathcal{S}_f$ and $\mathcal{S}_p$ satisfy the following requirements:

R1 The height variation w.r.t. the previous footstep is within a maximum range, i.e., $|z^j_{\text{f}} - z^{j-1}_{\text{f}}| \leq \Delta z_{\max}$.

R2 The footstep is fully in contact within a single horizontal patch, i.e., each cell of $\mathcal{M}_z$ belonging to or overlapping with the footprint has the same height z^j_{f}.

R3 Apart from the ground contact at the start and at the end, the swing foot trajectory $\boldsymbol{p}^j_{\text{swg}}$ is collision-free.

4.2.2 Planner overview

To find a feasible (in the sense of the requirements R1-R3) footstep plan which leads the robot to the goal region $\mathcal{G}$, we propose a RRT-based algorithm which does not require any kind of pre-processing of the environment and allows to embed a feasibility check directly in the expansion mechanism.

Our footstep planner, whose pseudocode is given in Algorithm 4, iteratively builds a tree $\mathcal{T}$ in the search space. A vertex $v = (\boldsymbol{f}_{\text{sup}}, \boldsymbol{f}_{\text{swg}})$ specifies the poses $\boldsymbol{f}_{\text{sup}}$ and $\boldsymbol{f}_{\text{swg}}$ of the two feet during a double support phase; in all steps originating from v, the support foot will remain at $\boldsymbol{f}_{\text{sup}}$ while the swinging foot will move from $\boldsymbol{f}_{\text{swg}}$. An edge exists between two vertexes when there exists a collision-free trajectory of the swing foot connecting the two. At the beginning, the algorithm roots $\mathcal{T}$ at $v_{\text{ini}} = (\boldsymbol{f}_R^{\text{ini}}, \boldsymbol{f}_L^{\text{ini}})$, which describes the initial double support configuration.

The generic iteration of the algorithm starts by selecting a sample point $\boldsymbol{p}_{\text{rand}}$ on the ground. We allow the planner to randomly choose between exploration and exploitation, to bias the growth of the tree towards, respectively, unexplored regions and the goal. In the first case, $\boldsymbol{p}_{\text{rand}}$ is generated by randomly choosing its x, y coordinates and retrieving the corresponding z coordinate from $\mathcal{M}_z$. In the second case, $\boldsymbol{p}_{\text{rand}}$ is sampled from the goal region $\mathcal{G}$.

Then, the vertex v_{near} of $\mathcal{T}$ that is closest to $\boldsymbol{p}_{\text{rand}}$ is selected for an expansion attempt. The following metric is used for evaluating the distance between a certain vertex v and a point $\boldsymbol{p} \in I\!R^3$:

$$\gamma(v, \boldsymbol{p}) = d(\boldsymbol{m}, \boldsymbol{p}) + \alpha|\theta_p|.$$

Here, $d(\boldsymbol{m}, \boldsymbol{p})$ is the Euclidean distance of the midpoint $\boldsymbol{m}$ between the feet (at the poses $\boldsymbol{f}_{\text{sup}}$ and $\boldsymbol{f}_{\text{swg}}$ specified in v) from $\boldsymbol{p}$; θ_p is the angle between the robot sagittal axis (defined as the axis passing through $\boldsymbol{m}$ with orientation equal to the average of the orientations of the two footsteps) and the line joining $\boldsymbol{m}$ to $\boldsymbol{p}$; α is a positive scalar.

Algorithm 4: Basic Footstep Planner

```
1  v_ini ← (f_R^ini, f_L^ini);
2  AddVertex(T, ∅, v_ini);
3  i ← 0;
4  repeat
5      i ← i + 1;
6      p_rand ← SamplePoint();
7      v_near ← NearestVertex(T, p_rand);
8      f_cand ← GenerateCandidateFootstep(f_sup^near);
9      if Feasible(f_cand) then
10         p_swg^cand ← GenerateSwingingTrajectory(f_swg^near, f_cand);
11         if p_swg^cand ≠ ∅ then
12             v_new ← (f_cand, f_sup^near);
13             AddVertex(T, v_near, v_new);
14         end
15     end
16 until SolutionFound() or i = i_max;
```

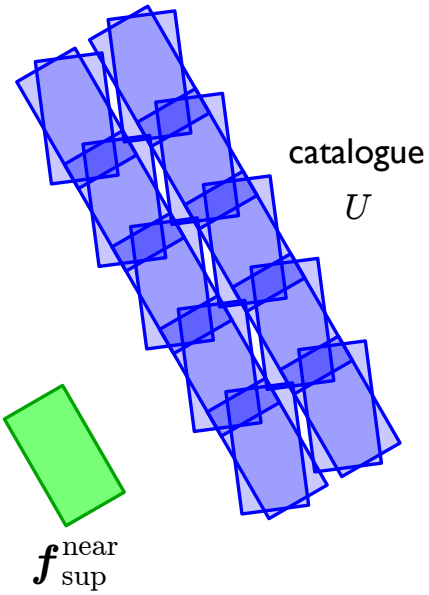

Figure 4.3. The catalogue U of primitives specifies the possible planar poses (in blue) of the candidate footstep $\boldsymbol{f}^{\text{cand}}$ with respect to the pose $\boldsymbol{f}_{\text{sup}}^{\text{near}}$ of the current support foot (in green). Here we have considered the case of a swinging right foot; the catalogue for a swinging left foot is specular. Once a primitive is chosen, the z coordinate of the candidate footstep will be retrieved from the elevation map $\mathcal{M}_z$.

Once v_{near} has been identified, the foot poses $\boldsymbol{f}_{\text{sup}}^{\text{near}}$ and $\boldsymbol{f}_{\text{swg}}^{\text{near}}$ are extracted from it. A candidate footstep $\boldsymbol{f}^{\text{cand}}$ is generated by randomly selecting a final pose of the swinging foot from a given catalogue U of primitives, which is defined with respect to $\boldsymbol{f}_{\text{sup}}^{\text{near}}$ (see Fig. 4.3). The z coordinate of $\boldsymbol{f}^{\text{cand}}$ is retrieved from the elevation map $\mathcal{M}_z$.

At this point, the candidate footstep is checked for feasibility with respect to requirements R1–R2. If the outcome is positive, the algorithm verifies whether a collision-free trajectory $\boldsymbol{p}_{\text{swg}}^{\text{cand}}$ exists that brings the swinging foot from $\boldsymbol{f}_{\text{swg}}^{\text{near}}$ to $\boldsymbol{f}^{\text{cand}}$ (requirement R3). This is done through the iterative procedure given in Procedure 4 that uses a parametrized trajectory which, once the endpoints are selected, can be deformed[2] by changing the maximum height h along the trajectory. Starting from a lower bound $h_{\min}$, the algorithm iteratively checks increasing values for h up to an upper bound $h_{\max}$. If a collision-free trajectory is found, a new vertex $v_{\text{new}} = (\boldsymbol{f}^{\text{cand}}, \boldsymbol{f}_{\text{sup}}^{\text{near}})$ is added to the tree, connected to its parent v_{near} by $\boldsymbol{p}_{\text{swg}}^{\text{cand}}$. Note that in v_{new} the roles of the feet have been swapped: in future steps originating from v_{new}, the support foot will stay at $\boldsymbol{f}^{\text{cand}}$ while the swinging foot will move from $\boldsymbol{f}_{\text{sup}}^{\text{near}}$.

Planning terminates when v_{new} completes the walk-to task (i.e., the corresponding midpoint $\boldsymbol{m}$ belongs to $\mathcal{G}$) or a maximum number of iterations has been reached. In the first case, the path joining the root to v_{new} is extracted from the tree, and the footstep sequence $\mathcal{S}_f$ is reconstructed with the associated sequence of swing foot trajectories $\mathcal{S}_p$.

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

Procedure 4: GenerateSwingingTrajectory($\boldsymbol{f}_j$, $\boldsymbol{f}_k$)

```
h ← h_min;
while h ≤ h_max do
    p_swg^cand ← BuildTrajectory(f_j, f_k, h);
    if CollisionFree(p_swg^cand) then
        return p_swg^cand;
    end
    h ← h + Δh;
end
return ∅;
```

4.2.3 Simulations

In this section, we present simulation results obtained by testing our C++ implementation of the proposed framework in the V-REP environment with the HRP4 robot, a 1.5 m tall humanoid with 34 degrees of freedom by Kawada Robotics [3]. We invite the reader to watch the video available at the link `https://youtu.be/mKVBwlxsXLM`, which shows the working principle of the presented footstep planner along with some simulation results.

Simulations have been performed in three different scenarios. In all of them the robot has to reach a circular goal region. The radius of such region has been set to 0.5 m in the first two scenarios and to 1.0 m in the third one. Since the footstep planner is randomized, to evaluate its performance we have performed a set of 20 simulations on each scenario.

All simulations have been performed on an Intel Core i7 running at 2.7 GHz, using the same parameters settings. In particular, in the footstep planning module we have set $\alpha = 1$, $\Delta z_{\max} = 0.08$ m, $h_{\min} = 0.02$, $h_{\max} = 0.12$ and $\Delta h = 0.02$. According to the HRP4 kinematic characteristics, the catalogue U of primitives has been chosen as $U = \{(x, y, \theta) \in \{-0.1, 0, 0.1, 0.2, 0.3\} \times \{0.2, 0.3\} \times \{0, \frac{\pi}{8}\}\}$ in case of right support (the catalogue for the case of left support is specular). For the gait generation module (see Appendix A.2), we used $\omega = 3.6$ s^{-1}, the step duration is $T_s = 0.8$ s of which 0.5 s of single support and 0.3 s of double support, and the vertical size of the ZMP constraint is set to $d_z^z = 0.03$ m. Finally, the elevation map $\mathcal{M}_z$ has a resolution of 0.02 m.

Fig. 4.4 shows the three scenarios; for each of them, one of the footstep planner

[3] The results presented in this section have been obtained with an optimized version of the C++ implementation through which the results that we originally presented in [4] were obtained.

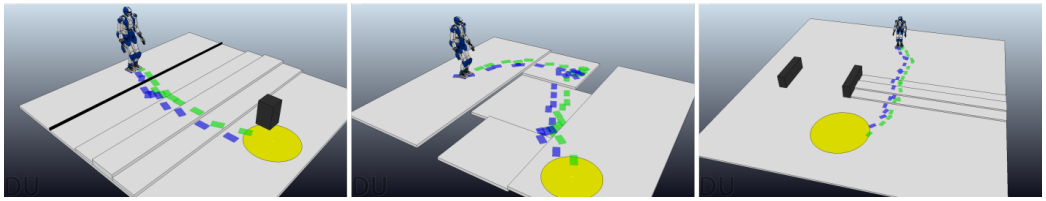

Figure 4.4. The three considered scenarios. For each of them a possible solution of the basic footstep planner is shown.

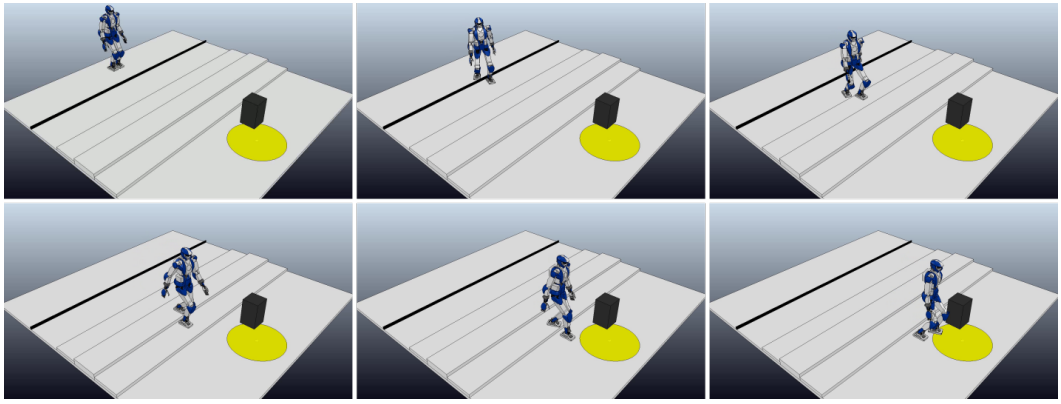

Figure 4.5. Scenario 1: the robot reaches the goal region going over the black linear obstacle, climbing and descending the staircase, and avoiding the black box.

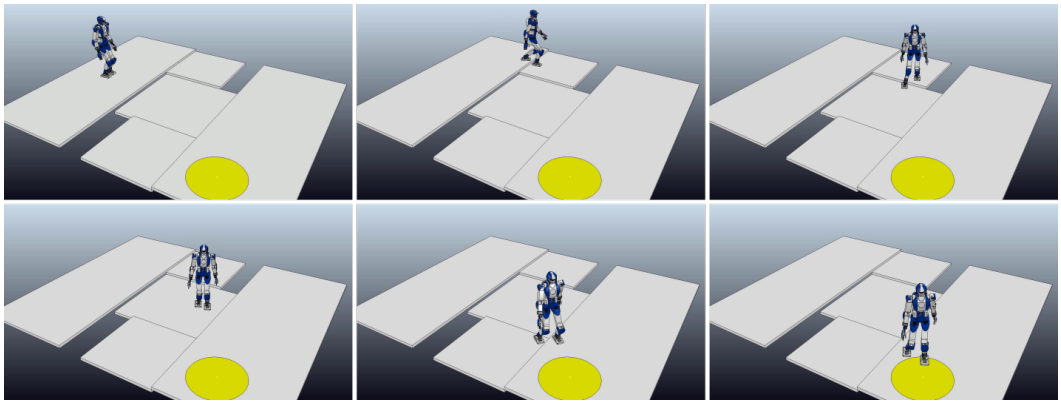

Figure 4.6. Scenario 2: the robot reaches the goal region moving through the ditch, which can only be accessed from the left and exited from the right.

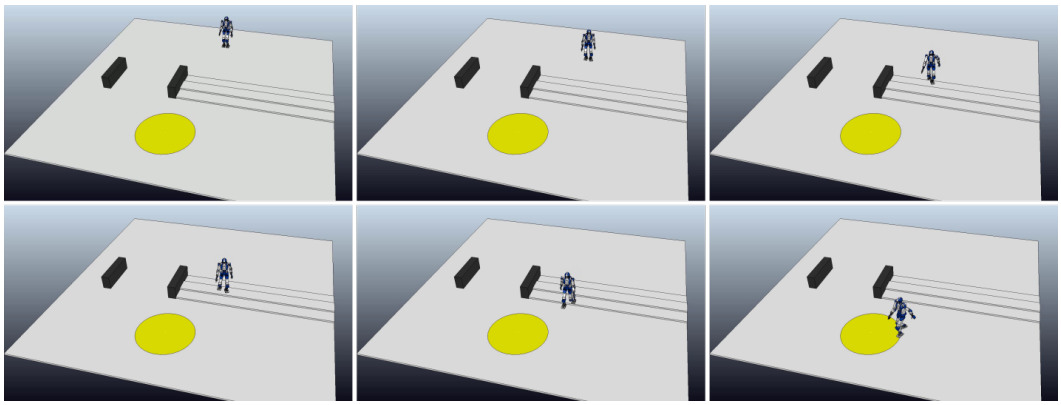

Figure 4.7. Scenario 3: the robot reaches the goal region after climbing and descending the staircase.

solutions is displayed.

In the first scenario, the robot and the goal are placed at opposite sides of the environment. The goal can be reached by walking a mostly straight path, but footsteps need to be placed appropriately to satisfy all the requirements. Fig. 4.5

shows one of the solutions found by our method. The robot starts walking forward and reaches the proximity of the bar obstacle, which does not provide a large enough surface to step on. Then, the robot goes over it by taking a longer step, with a swing foot trajectory sufficiently high to avoid collisions. After that, the staircase is ascended and descended. Finally, the robot avoids the black box and reaches the goal region.

The second scenario is traversed by a ditch which 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. One solution produced by our method is shown in Fig. 4.6. The robot moves appropriately among the different levels, first going down in the ditch and then up towards the goal.

The third scenario is larger than the first two and the robot, in order to reach the goal may either ascend and descend a staircase or walk on a flat ground delimited by two low walls. In view of its probabilistic nature, the proposed footstep planner is free to randomly choose between the two options. A possible solution of our method is shown in Fig. 4.7. Here, the planner arbitrarily chose a footstep plan leading the robot to the goal region by ascending and descending the staircase.

We emphasize that the resulting motions are produced thanks to the effective combination of the footstep planning module with the MPC-based gait generation module. Table 4.1 reports some data obtained by averaging the planning results of the 20 simulations on each scenario. The table indicates the number of vertexes in the tree produced by the planner, the time needed to compute a footstep plan, the length of the plan in terms of number of footsteps, the overall height variation of the swing foot along the plan, and the minimum clearance. The last three parameters will be discussed in detail in Sect. 4.3.1, and considered directly in the planning stage in Sect. 4.3.2.

It is clear that the first and the third scenarios are significantly easier than the second to solve. The reason is that in both the first and third scenarios there exists a straight path to the goal, while in the second the robot must take an S-shaped path in order to guarantee satisfaction of requirement R1. The descending steps in the second scenario also require the robot to perform the difficult maneuver of rotating in place in a small space. Although, the solution is found in a reasonably short time even in the second scenario.

An analysis of the presented solutions, together with the data provided in Table 4.1, reveals that the generated motions are suboptimal in terms of efficiency, particularly in the second simulation, where the presence of unnecessary steps is evident. Such lack of quality of the generated plans is due to the randomized nature

scenario	tree size (# vertexes)	planning time (s)	plan len. (# steps)	height var. (m)	minimum clearance (m)
1	763.40	1.564	24.45	0.480	0.388
2	1641.40	3.692	44.60	0.480	-
3	555.95	0.712	32.80	0.264	0.493

Table 4.1. Basic footstep planner performance data.

of the proposed footstep planner. In fact, it can arbitrarily produce footstep plans with the constraint of respecting requirements R1-R3, without explicitly accounting for their optimality.

4.3 Optimal footstep planner

The footstep planner proposed in Sect. 4.2 proved capable to compute feasible plans. However, as discussed in Sect. 4.2.3, the quality of the produced plans can not be guaranteed, as it is not explicitly accounted in the planning strategy. Clearly, to evaluate the quality of a plan, a desired criterion needs to be be specified. Sect. 4.3.1 formally defines the optimality of a plan w.r.t. a desired criterion, and describes some possible criteria of interest. Sect. 4.3.2 proposes an extension of the method presented in Sect. 4.2.2 for asymptotically computing optimal footstep plans. Sect. 4.2.3 presents simulations obtained with the proposed approach, and a comparison of the results with those obtained with the basic planner.

4.3.1 Problem formulation

Consider a generic sequence of footsteps $\mathcal{S}_f = \{\boldsymbol{f}^1, \dots, \boldsymbol{f}^n\}$ and a sequence of associated swing foot trajectories $\mathcal{S}_p = \{\boldsymbol{p}^1_{\text{swg}}, \dots, \boldsymbol{p}^{n-2}_{\text{swg}}\}$. Denote by $c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2})$ the cost associated to the step leading the foot from $\boldsymbol{f}^j$ to $\boldsymbol{f}^{j+2}$ through the swing foot trajectory $\boldsymbol{p}^j_{\text{swg}}$. Such step cost is assigned according to the chosen criterion for evaluating the quality of a plan and, in general, is function of $\boldsymbol{f}^j$, $\boldsymbol{f}^{j+2}$ and/or $\boldsymbol{p}^j_{\text{swg}}$. Let $c_{\mathcal{S}}(\boldsymbol{f}^n)$ be the cost function evaluating the overall cost for reaching the final footstep $\boldsymbol{f}^n$ through the sequences $\mathcal{S}_f$ and $\mathcal{S}_p$.

The optimal footstep planning problem consists in finding a sequence of footsteps $\mathcal{S}_f = \{\boldsymbol{f}^1, \dots, \boldsymbol{f}^n\}$ leading to the desired location, i.e., $\boldsymbol{f}^n \in \mathcal{G}$, together with a sequence of associated swing foot trajectories $\mathcal{S}_p = \{\boldsymbol{p}^1_{\text{swg}}, \dots, \boldsymbol{p}^{n-2}_{\text{swg}}\}$, that, in addition to satisfy the feasibility requirements R1-R3 described in Sect. 4.2.1, are *optimal* w.r.t. a desired criterion, i.e., the sequences $\mathcal{S}_f$ and $\mathcal{S}_p$ minimize the cost function $c_{\mathcal{S}}(\boldsymbol{f}^n)$.

Given the sequences $\mathcal{S}_f$ and $\mathcal{S}_p$, depending on the nature of the chosen criterion for evaluating the plan quality, the step costs $c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2})$ can be cumulative or not along $\mathcal{S}_f$ and $\mathcal{S}_p$. Consider a specific footstep $\boldsymbol{f}^k$, with $k \leq n$, in $\mathcal{S}_f$. In the case of cumulative costs, the cost $c_{\mathcal{S}}(\boldsymbol{f}^k)$ for reaching $\boldsymbol{f}^k$ coincides with the sum of the step costs along the subsequences of $\mathcal{S}_f$ and $\mathcal{S}_p$ ending at $\boldsymbol{f}^k$

$$c_{\mathcal{S}}(\boldsymbol{f}^k) = \sum_{j=1}^{k-2} c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2}). \tag{4.1}$$

In the case of non-cumulative costs, the cost $c_{\mathcal{S}}(\boldsymbol{f}^k)$ for reaching $\boldsymbol{f}^k$ is determined by the highest (worst) step cost along the subsequences of $\mathcal{S}_f$ and $\mathcal{S}_p$ ending at $\boldsymbol{f}^k$

$$c_{\mathcal{S}}(\boldsymbol{f}^k) = \max_{j \in \{1,\dots,k-2\}} c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2}). \tag{4.2}$$

The cost $c_{\mathcal{S}}(\boldsymbol{f}^n)$ of a footstep plan ending at $\boldsymbol{f}^n \in \mathcal{G}$ can be computed applying (4.1) or (4.2) depending on the chosen criterion.

Given a generic footstep $\boldsymbol{f}$, and sequences $\mathcal{S}_f$ and $\mathcal{S}_p$, the cost of reaching $\boldsymbol{f}$ via a specific footstep $\boldsymbol{f}^k \in \mathcal{S}_f$ is

$$c_{\mathcal{S}}(\boldsymbol{f}^k, \boldsymbol{f}) = c_{\mathcal{S}}(\boldsymbol{f}^k) + c(\boldsymbol{f}^k, \boldsymbol{f}) \tag{4.3}$$

if the step costs are cumulative,

$$c_{\mathcal{S}}(\boldsymbol{f}^k, \boldsymbol{f}) = \max\{c_{\mathcal{S}}(\boldsymbol{f}^k), c(\boldsymbol{f}^k, \boldsymbol{f})\} \tag{4.4}$$

otherwise. Here, $c_{\mathcal{S}}(\boldsymbol{f}^k)$ is computed using (4.1) and (4.2) in (4.3) and (4.4), respectively.

In the following, we describe three possible criteria for evaluating the plan quality that can be involved in our formulation, highlighting the corresponding step costs and the resulting cost function.

C1 *Minimization of the number of steps*: every step has identical cost, independently from its characteristics (e.g., length or height)

$$c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2}) = 1. \tag{4.5}$$

Step costs in (4.5) are clearly cumulative, hence cost function $c_{\mathcal{S}}(\boldsymbol{f}^n)$ takes the form in (4.1).

C2 *Minimization of the height variation*: the cost of a step is equal to the vertical displacement between the foot position at the begin and at the end of the step

$$c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2}) = |z_{\mathrm{f}}^j - z_{\mathrm{f}}^{j+2}|. \tag{4.6}$$

As in the previous case, step costs in (4.6) are cumulative and cost function $c_{\mathcal{S}}(\boldsymbol{f}^n)$ takes the form in (4.1).

C3 *Maximization of the minimum clearance*: the cost of a step is equal to the inverse of the clearance at the end of the step

$$c(\boldsymbol{f}^j, \boldsymbol{f}^{j+2}) = \frac{1}{\alpha(\boldsymbol{f}^{j+2})} \tag{4.7}$$

with $\alpha(\boldsymbol{f}^{j+2})$ the clearance at footstep $\boldsymbol{f}^{j+2}$.

Let $\mathcal{O}$ be the set of points located on patches with which any kind of contact is forbidden. Such patches act as full-fledged obstacles that must be avoided. We assume that the set $\mathcal{O}$ is provided. Then, the clearance at footstep $\boldsymbol{f}$ is defined as

$$\alpha(\boldsymbol{f}) = \min_{\boldsymbol{p}_o \in \mathcal{O}} \left\| \bar{\boldsymbol{p}}_f - \bar{\boldsymbol{p}}_o \right\|$$

where $\bar{\boldsymbol{p}}_f$ and $\bar{\boldsymbol{p}}_o$ are the planar positions (x- and y- coordinates) of, respectively, the footstep $\boldsymbol{f}$ and its closest point $\boldsymbol{p}_o$ in $\mathcal{O}$.

Differently from criteria C1-C2, the maximization of the minimum clearance does not involve cumulative costs. In fact, step costs in (4.7) can be used in (4.2) to obtain the corresponding cost function $c_{\mathcal{S}}(\boldsymbol{f}^n)$. Note that, under the above formulation, the problem of finding a plan that maximizes the minimum clearance is treated as finding a plan that minimizes (analogously to cases C1-C2) the maximum (from (4.2)) inverse clearance (from (4.7)).

4.3.2 Planner overview

To address the problem described in the previous section, we propose an extension of the method presented in Sect. 4.2.2 that is able to asymptotically find footstep plans that are optimal w.r.t. a desired criterion, i.e., the algorithm asymptotically converges to the optimal solution as the number of iterations increases.

The proposed planner, whose pseudocode is given in Algorithm 5, relies on a RRT*-like strategy. It works similarly to the basic planner, with the difference that it involves two additional operations in order to minimize the cost of the footstep plan. The first is performed before adding the new vertex v_{new} to the tree, and is in charge of selecting its best parent vertex among the potential ones. The second is performed after the insertion of v_{new} in the tree, and is in charge of modifying the connections between vertexes in $\mathcal{T}$ in order to reduce the associated costs.

Once a candidate footstep $\boldsymbol{f}_{\text{cand}}$ and a swing foot trajectory $\boldsymbol{p}_{\text{swg}}^{\text{cand}}$ satisfying requirements R1-R3 are generated, the algorithm retrieves from tree $\mathcal{T}$ the set $\mathcal{V}_{\text{near}}^{\text{in}}$ of vertexes from which footstep $\boldsymbol{f}_{\text{cand}}$ can potentially be reached. In particular, each vertex $v_h = (\boldsymbol{f}_{\text{sup}}^h, \boldsymbol{f}_{\text{swg}}^h) \in \mathcal{V}_{\text{near}}^{\text{in}}$ is such that the identity of the support foot is equal to that at v_{near} (the vertex from which the expansion attempt originated), and $\boldsymbol{f}_{\text{cand}}$ is kinematically reachable from v_h, i.e., $\Delta\boldsymbol{f}_{\min} \leq {}^h\boldsymbol{f}_{\text{cand}} \leq \Delta\boldsymbol{f}_{\max}$, where ${}^h\boldsymbol{f}_{\text{cand}}$ expresses $\boldsymbol{f}_{\text{cand}}$ w.r.t. $\boldsymbol{f}_{\text{sup}}^h$ and bounds $\Delta\boldsymbol{f}_{\min} = (\Delta x_{\min}, \Delta y_{\min}, \Delta z_{\min}, \Delta\theta_{\min})^T$, $\Delta\boldsymbol{f}_{\max} = (\Delta x_{\max}, \Delta y_{\max}, \Delta z_{\max}, \Delta\theta_{\max})^T$ are chosen according to the robot kinematic limits. Here, $\Delta z_{\min} = -\Delta z_{\max}$ to comply with requirement R1. Note that the primitives in the catalogue U are chosen within such bounds as well.

With the aim of selecting the vertex in $\mathcal{V}_{\text{near}}^{\text{in}}$ that permit to reach $\boldsymbol{f}^{\text{cand}}$ with the lowest cost, the iterative procedure shown in Procedure 5 is invoked. At each

Algorithm 5: Optimal Footstep Planner

```
1  v_ini ← (f_R^ini, f_L^ini);
2  AddVertex(T, ∅, v_ini);
3  i ← 0;
4  repeat
5      i ← i + 1;
6      p_rand ← SamplePoint();
7      v_near ← NearestVertex(T, p_rand);
8      f_cand ← GenerateCandidateFootstep(f_sup^near);
9      if Feasible(f_cand) then
10         p_swg^cand ← GenerateSwingingTrajectory(f_swg^near, f_cand);
11         if p_swg^cand ≠ ∅ then
12             V_near^in ← NearestVertexesIn(T, f^cand);
13             v_min ← ChooseParent(V_near^in, v_near, f_cand);
14             v_new ← (f_cand, f_sup^min);
15             AddVertex(T, v_min, v_new);
16             V_near^out ← NearestVertexesOut(T, f_sup^new);
17             ReWire(V_near^out, v_new);
18         end
19     end
20 until i = i_max;
```

Procedure 5: ChooseParent($\mathcal{V}^{\text{in}}_{\text{near}}$, v_{near}, $\boldsymbol{f}_{\text{cand}}$)

```
v_min ← v_near;
c_min ← CostVia(v_near, f_cand);
foreach v_h ∈ V^in_near do
    p^h_swg ← GenerateSwingingTrajectory(f^h_swg, f_cand);
    if p^h_swg ≠ ∅ then
        c_h ← CostVia(v_h, f_cand);
        if c_h < c_min then
            v_min ← v_h;
            c_min ← c_h;
        end
    end
end
return v_min;
```

Procedure 6: ReWire($\mathcal{V}^{\text{out}}_{\text{near}}$, v_{new})

```
foreach v_h ∈ V^out_near do
    p^h_swg ← GenerateSwingingTrajectory(f^new_swg, f^h_sup);
    if p^h_swg ≠ ∅ and CostVia(v_new, f^h_sup) < Cost(f^h_sup) then
        ReConnect(v_new, v_h);
    end
end
return;
```

iteration, the procedure extracts a different vertex $v_h = (\boldsymbol{f}^h_{\text{sup}}, \boldsymbol{f}^h_{\text{swg}})$ from $\mathcal{V}^{\text{in}}_{\text{near}}$, and verifies whether a collision-free trajectory $\boldsymbol{p}^h_{\text{swg}}$ exists that brings the swinging foot from $\boldsymbol{f}^h_{\text{swg}}$ to $\boldsymbol{f}^{\text{cand}}$. If a collision-free trajectory is found, requirements R1-R3 are satisfied[4], and the cost for reaching $\boldsymbol{f}_{\text{cand}}$ via $\boldsymbol{f}^h_{\text{swg}}$ is computed using (4.3) or (4.4) depending on whether the step costs are cumulative or not. The vertex v_{min} that yields the minimum cost for reaching $\boldsymbol{f}_{\text{cand}}$ is returned to the planner as the parent of the new vertex, together with the associated swing foot trajectory $\boldsymbol{p}^{\text{min}}_{\text{swg}}$ (not explicitly displayed in Procedure 5). Then, a new vertex $v_{\text{new}} = (\boldsymbol{f}^{\text{cand}}, \boldsymbol{f}^{\text{min}}_{\text{sup}})$ is added to the tree, connected to its parent v_{min} by $\boldsymbol{p}^{\text{min}}_{\text{swg}}$.

Then, the algorithm retrieves from tree $\mathcal{T}$ the set $\mathcal{V}^{\text{out}}_{\text{near}}$ of vertexes containing footsteps which can potentially be reached from $\boldsymbol{f}^{\text{new}}_{\text{sup}}$. In particular, each vertex $v_h = (\boldsymbol{f}^h_{\text{sup}}, \boldsymbol{f}^h_{\text{swg}}) \in \mathcal{V}^{\text{out}}_{\text{near}}$ is such that the identity of the support foot is opposite to that at v_{new}, and $\boldsymbol{f}^h_{\text{sup}}$ is kinematically reachable from v_{new}, i.e., $\Delta\boldsymbol{f}_{\text{min}} \leq {}^{\text{new}}\boldsymbol{f}^h_{\text{sup}} \leq \Delta\boldsymbol{f}_{\text{max}}$, where ${}^{\text{new}}\boldsymbol{f}^h_{\text{sup}}$ expresses $\boldsymbol{f}^h_{\text{sup}}$ w.r.t. $\boldsymbol{f}^{\text{new}}_{\text{sup}}$.

At this point, the rewiring procedure given in Procedure 6 is invoked. At each iteration, the procedure extracts a different vertex $v_h = (\boldsymbol{f}^h_{\text{sup}}, \boldsymbol{f}^h_{\text{swg}})$ from $\mathcal{V}^{\text{out}}_{\text{near}}$, and verifies whether a collision-free trajectory $\boldsymbol{p}^h_{\text{swg}}$ exists that brings the swinging foot from $\boldsymbol{f}^{\text{new}}_{\text{swg}}$ to $\boldsymbol{f}^h_{\text{sup}}$. If a collision-free trajectory is found, requirements R1-R3 are

[4]In fact, R1 is automatically satisfied as $\Delta z_{\text{min}} = -\Delta z_{\text{max}}$ in the bounds of the admissible region used for retrieving $\mathcal{V}^{\text{in}}_{\text{near}}$, R2 is checked at the time of $\boldsymbol{f}_{\text{cand}}$ creation, and R3 is explicitly checked during the swing foot trajectory generation.

satisfied. Then, the cost for reaching $\boldsymbol{f}^h_{\text{sup}}$ via $\boldsymbol{f}^{\text{new}}_{\text{swg}}$ is computed using (4.3) or (4.4). In case such cost is less than the current cost of reaching $\boldsymbol{f}^h_{\text{sup}}$ (via the current parent of v_h), v_h is set as a child of v_{new}, and the pose of the swing foot $\boldsymbol{f}^h_{\text{swg}}$ is accordingly updated to $\boldsymbol{f}^{\text{new}}_{\text{sup}}$.

Planning terminates when a maximum number of iterations has been reached. The branch of $\mathcal{T}$ leading to $\mathcal{G}$ (if any) with minimum cost is then extracted from the tree, and the footstep sequence $\mathcal{S}_f$ is reconstructed with the associated sequence of swing foot trajectories $\mathcal{S}_p$. Clearly, the larger the maximum number of iterations, the better the obtained footstep plan.

4.3.3 Simulations

In this section, we present simulation results obtained in the V-REP environment with the HRP4 humanoid robot, considering the same scenarios and parameters setting of Sect. 4.2.3. In addition, the bounds $\Delta\boldsymbol{f}_{\text{min}}$ and $\Delta\boldsymbol{f}_{\text{max}}$ are set to $(-0.1, 0.2, -0.08, 0)^T$ and $(0.3, 0.3, 0.08, \frac{\pi}{8})^T$ in case of right support, and to $(-0.1, -0.3, -0.08, -\frac{\pi}{8})^T$ and $(0.3, -0.2, 0.08, 0)^T$ in the case of left support. The set $\mathcal{O}$ contains all points (x, y, z) such that (x, y) indicate a particular cell of $\mathcal{M}_z$, and $z = \mathcal{M}_z(x, y)$ exceeds a threshold of 0.4 m. Thus, in the first scenario, $\mathcal{O}$ contains all the points located on the black box overlapping with the goal region. In the second scenario, $\mathcal{O}$ is instead empty. Finally, in the third scenario, $\mathcal{O}$ contains all the points on the two black walls.

To evaluate the performance of the optimal footstep planner, we tested it in each scenario for the three different cost criteria (C1-C3) described in Sect. 4.3.1. For each combination of scenario and cost criterion, we performed 20 simulations setting $i_{\text{max}} = 5000$. Table 4.2 reports data obtained by averaging the planning results, analogously to Table 4.1. A comparison of the two tables clearly shows that the optimal planner outperforms the basic planner in terms of the plan quality according to the chosen cost criterion.

In the first scenario (see Fig. 4.8 for examples of the planner solutions depending

scenario	cost	tree size (# vertexes)	planning time (s)	plan len. (# steps)	height var. (m)	minimum clearance (m)
1	C1	1986.00	9.743	21.15	0.480	0.438
	C2	1958.25	9.498	26.25	0.480	0.385
	C3	1978.90	9.572	25.25	0.480	0.505
2	C1	2075.00	11.709	37.60	0.480	-
	C2	2125.00	11.333	45.80	0.480	-
	C3	-	-	-	-	-
3	C1	4102.40	14.464	29.60	0.312	0.750
	C2	4098.30	14.493	44.90	0.096	0.396
	C3	4131.35	14.072	41.10	0.336	1.068

Table 4.2. Optimal footstep planner performance data in the three scenarios, varying the cost criterion.

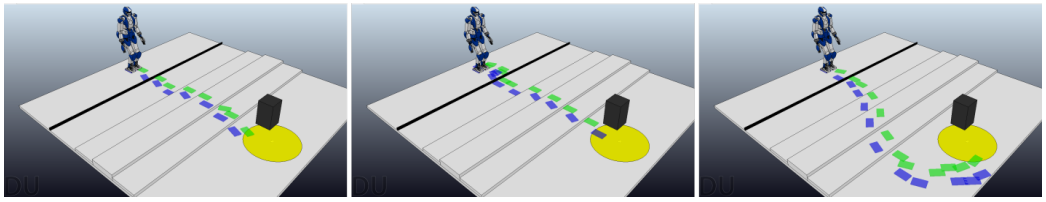

Figure 4.8. Scenario 1: possible solutions of the optimal footstep planner varying the cost criterion, i.e., minimizing the number of steps (left), minimizing the height variation (center), and maximizing the minimum clearance (right).

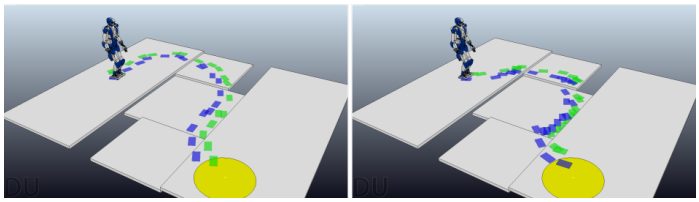

Figure 4.9. Scenario 2: possible solutions of the optimal footstep planner varying the cost criterion, i.e., minimizing the number of steps (left), and minimizing the height variation (right).

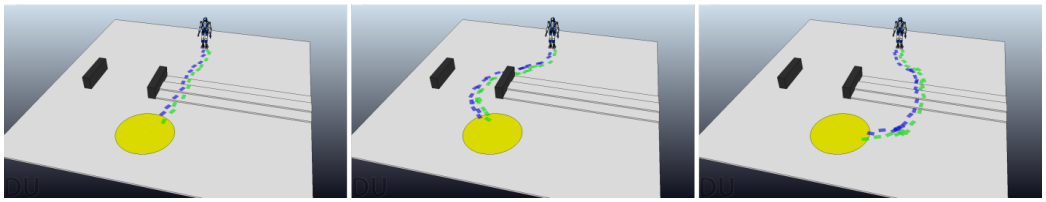

Figure 4.10. Scenario 3: possible solutions of the optimal footstep planner varying the cost criterion, i.e., minimizing the number of steps (left), minimizing the height variation (center), and maximizing the minimum clearance (right).

on the chosen cost criterion), when using C1 as the cost criterion, the number of footsteps along the plan decreases, with a consequent elimination of unnecessary steps. No improvement on the height variation is instead observed when using C2. This is due to the fact that the various patches constituting the environment must all be navigated in order for the robot to reach the goal region. The minimum clearance is slightly increased using C3. For example, in the solution shown in Fig. 4.8, the robot enters the goal region from the opposite side to that where the black box is located, thus maximizing the distance from the latter.

Similar observations apply for the second scenario (see Fig. 4.9) when using C1 and C2. The minimum clearance (C3) is not considered in this scenario as all the patches composing the environment are accessible, as none of them act as obstacle.

In the third scenario, the effect of the different choice of the cost criterion in the optimal footstep planner is more evident (see Fig. 4.10). Using C1, the computed plans are short and lead the robot to walk straight from its initial location to the goal region, ascending and descending the staircase (Fig. 4.11). Using C2, the planner tends to produce plans that lead the robot to walk through the flat portion of the environment delimited by the two low walls (Fig. 4.12). Finally, using C3, the planner produces plans that lead the robot to walk far from the low walls which

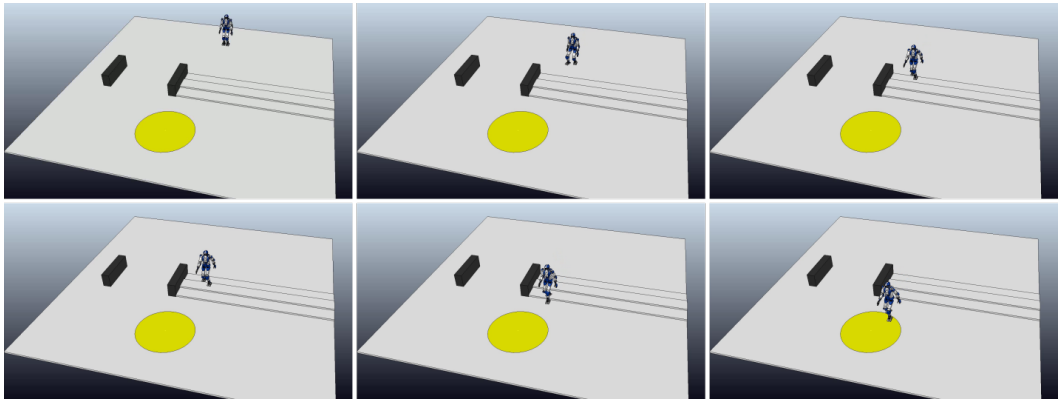

Figure 4.11. Scenario 3: minimizing the number of steps. The robot walks straight to the goal region, climbing and descending the staircase.

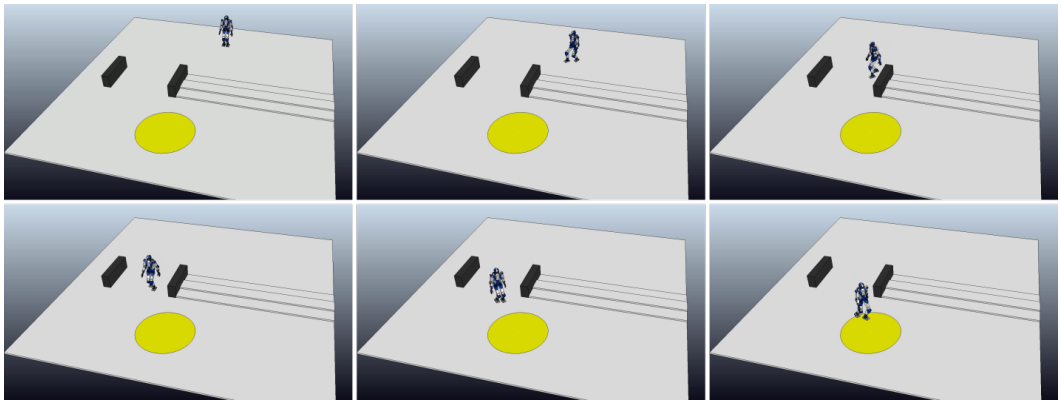

Figure 4.12. Scenario 3: minimizing the height variation. The robot reaches the goal region moving on the flat portion of the environment.

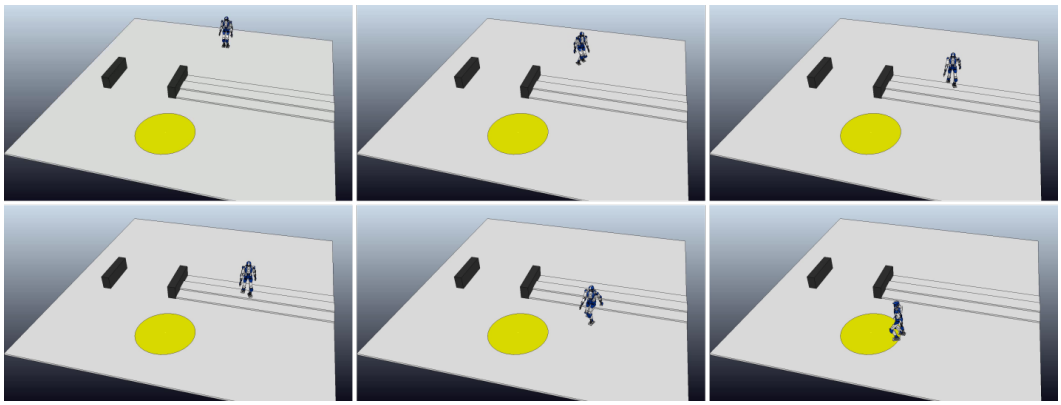

Figure 4.13. Scenario 3: maximizing the minimum clearance. The robot reaches the goal region walking far from the low black walls.

act as obstacles (Fig. 4.13); in most cases this requires to ascend and descend the staircase.

As expected, the improvement of plan quality comes at the expense of higher computational times. This is due to two facts. First, the optimal footstep planner

iters	cost	tree size (# vertexes)	planning time (s)	plan len. (# steps)	height var. (m)	minimum clearance (m)
1000	C1	853.45	2.108	31.40	0.264	0.482
	C2	856.95	2.109	33.20	0.264	0.664
	C3	867.60	2.125	31.70	0.288	0.563
3000	C1	2595.50	7.634	31.00	0.240	0.488
	C2	2575.05	7.439	38.25	0.120	0.423
	C3	2564.15	7.380	35.75	0.216	0.676
5000	C1	4102.40	14.464	29.60	0.312	0.750
	C2	4098.30	14.493	44.90	0.096	0.396
	C3	4131.35	14.072	41.10	0.336	1.068

Table 4.3. Optimal footstep planner performance data in the third scenario, varying the cost criterion and the number of iterations.

always performs $i_{\max}$ iterations, while the basic footstep planner terminates as soon as a solution is found (that in our simulations always happened before $i_{\max}$ is reached). Second, the two additional procedures for selecting the best parent of a new vertex and for the rewiring the tree introduce a considerable computational overhead.

In order to observe the asymptotic behavior of the proposed planner, we tested it in the third scenario varying the number of iterations ($i_{\max} = 1000, 3000, 5000$). Performance data collected in Table 4.3 clearly show that the algorithm improves the quality of the footstep plan, according to the chosen criterion, as the number of iterations increases.

4.4 Experiments

In this section, we present two experiments obtained testing the proposed framework on the NAO humanoid robot. In this case, rather than providing a performance assessment of the footstep planner (already presented via simulation results), we showcase the effectiveness of the proposed planning and control architecture on a real humanoid. To this purpose, the footstep planning module consists in the basic algorithm presented in Sect. 4.2.2.

In both experiments the robot has to reach a circular goal region of radius 0.1 m. The footstep planning module runs on an external computer which is equipped with an Intel Core i5 running at 2.7 GHz. We set $\alpha = 1$, $\Delta z_{\max} = 0.045$ m, $h_{\min} = 0.02$, $h_{\max} = 0.07$ and $\Delta h = 0.01$. According to the NAO kinematic characteristics, the catalogue U of primitives has been chosen as $U = \{(x, y, \theta) \in \{-0.06, 0, 0.06, 0.08, 0.10\} \times \{0.11, 0.12\} \times \{0, \frac{\pi}{12}\}\}$ in the case of right support (the catalogue for the case of left support is specular). Once planned, the sequences of footsteps and associated swing foot trajectories are sent to the robot via an ethernet cable through TCP. The gait generation module is implemented inside the B-Human RoboCup SPL team framework [91], and runs in real-time (at a control frequency of 100 Hz) on the NAO on-board CPU. We used $\omega = 6.68$ s^{-1}, the step duration is

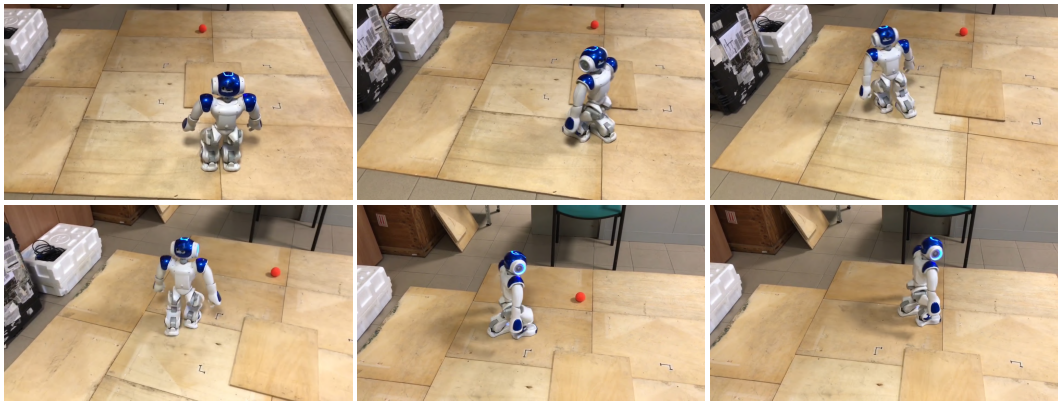

Figure 4.14. Experiment 1: the robot reaches the goal region avoiding collision with the central rectangular patch.

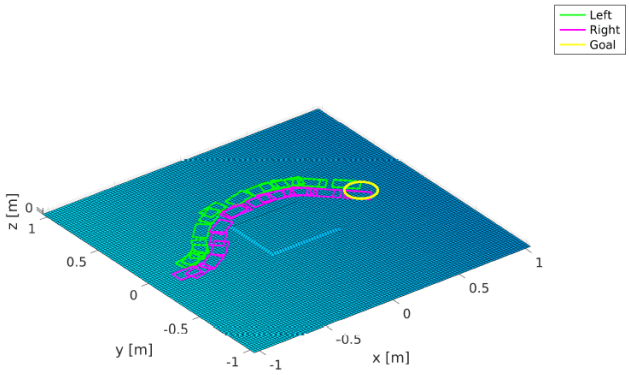

Figure 4.15. Experiment 1: the provided elevation map and the solution of the basic footstep planner.

$T_s = 0.48$ s of which 0.3 s of single support and 0.18 s of double support, and the vertical size of the ZMP constraint is set to $d_z^z = 0.05$ m.

In the first experiment, shown in Fig. 4.14, the goal region is centered at the red ball position. The elevation map $\mathcal{M}_z$ has been manually generated with a resolution of 0.02 m. The environment contains a rectangular patch having height 0.02 m at the center of a flat ground. Climbing on such central patch would require NAO to perform a large step leading the swing foot fully in contact with it (requirement R2). Due to its physical limitations, NAO can not perform such large steps. Then, the patch located at the center of the environment acts as a full-fledged obstacle for the robot. Fig. 4.15 shows the provided elevation map and the sequence of 31 footsteps that the planner generated in 0.070 s, building a tree of 488 vertexes. According to this sequence, under the action of the on-line gait generation module, NAO correctly reached the goal region avoiding collisions with the central patch.

In the second experiment, shown in Fig. 4.16, the goal region is located on top of a stair, that the robot must climb in order to complete the task. Differently from the previous case, the shape of such stair does not require large steps in order to be climbed. In this case, rather than directly providing the elevation map, we equipped the robot with a head-mounted Asus Xtion PRO Live camera through which it is

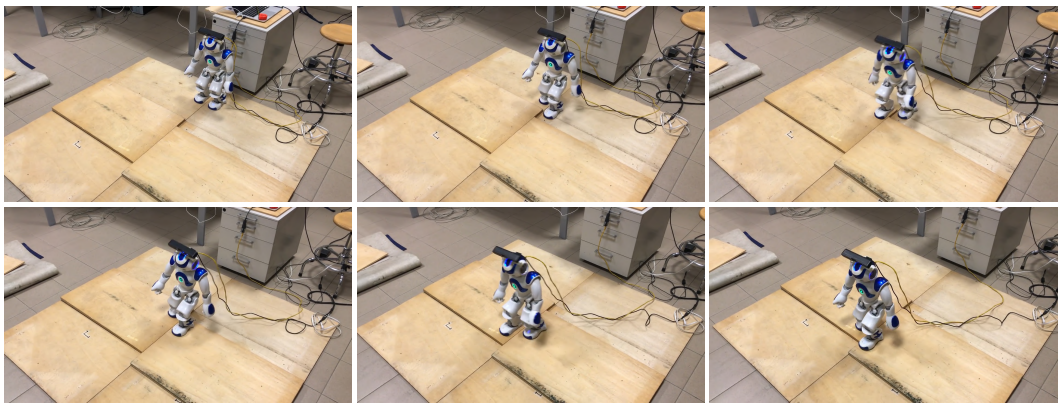

Figure 4.16. Experiment 2: the robot reaches the goal region climbing the stair.

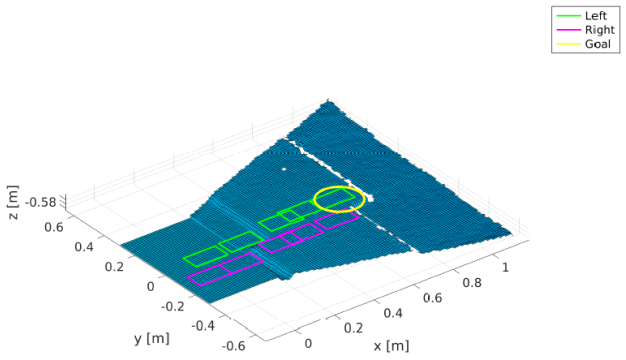

Figure 4.17. Experiment 2: the generated elevation map and the solution of the basic footstep planner.

allowed to acquire information about its surrounding before computing the footstep plan. Depth images collected by the camera are sent via a USB cable to a mapping module that runs on the same external computer running the footstep planning module. Such mapping module uses the framework proposed in [92] to generate an elevation map $\mathcal{M}_z$ with a resolution of 0.01 m. Fig. 4.17 shows the generated elevation map and the sequence of 10 footsteps that the planner generated in 0.331 s, building a tree of 454 vertexes. According to this sequence, under the action of the on-line gait generation module, NAO correctly climbed the stair and reached the goal region.

4.5 Conclusions

We presented an integrated architecture for planning and executing suitable walking motions for a humanoid robot that has to fulfil a walk-to task on uneven ground. This is obtained by combining an offline footstep planning module with an online gait generation module. The first module produces an appropriate sequence of footsteps leading to the goal region, while the second generates in real-time a suitable variable height CoM trajectory that is compatible with the planned footsteps and guaranteed to be bounded with respect to the ZMP evolution.

For the footstep planning module we have proposed two versions, both employing a randomized strategy. The first version proved to be able of generating feasible footstep plans in a short amount of time. The second version demonstrated its capability to compute, with some computational overhead, footstep plans of improved quality, accounting for different kinds of performance criteria.

The overall architecture, tested via simulations in V-REP with the HRP4 humanoid and via experiments with the NAO humanoid, revealed to be flexible, as it can embed constraints and can be applied to different environments without needing any pre-processing or heuristics definition.

This work can be extended along several directions. The main extension we are currently considering concerns the case of unknown environment for which we are developing a concurrent architecture (analogously to that proposed in Sect. 3.4) where planning, execution and SLAM take place simultaneously. In particular, the planning module will consist of a time-limited version of the proposed planners.

Other future work may address the case of environments with multiple floors and rough terrain where the assumption of piecewise-horizontal ground must be removed.

Chapter 5

Motion planning in the presence of soft task constraints

Robots are invariably required to execute tasks in a workspace that is populated by obstacles. If the robot is kinematically redundant with respect to a given task, it can perform it and simultaneously meet other basic requirements such as avoidance of collisions, joint limits, and so on.

As already discussed, tasks are expressed in terms of a certain set of coordinates, called task coordinates, which describe quantities related to, e.g., manipulation, navigation or perception. Often, the task is assigned as a desired path or trajectory for the task coordinates (as in the case of manipulation and visual tasks discussed in Sect. 3.1), resulting in an equality (*hard*) constraint. However, in industrial and service applications there are many situations where the task is better expressed by an inequality (*soft*) constraint; for example, this constraint may represent the fact that the desired path assigned to the task coordinates comes with a certain error tolerance. Fig. 5.1 shows an example of such scenario.

For the problem of generating collision-free robot motions in the presence of hard task constraints (TCMP) different methods have been proposed, either optimization- or sampling-based methods (see Chap. 2).

The first class of methods never guarantee completeness, while the second presents the drawback that task tracking is not enforced during the motion between adjacent samples, with the exception of the approach presented in [39], which relies on a motion generation scheme that guarantees continuous satisfaction of the task constraint. Building on this basic technique, a method for repeatable motion planning over cyclic tasks [93] and another for planning dynamically feasible motions in the presence of moving obstacles [94] have been proposed. All these planners are designed for the case of hard constraints.

Most motion planners for the case of soft task constraints rely on some sort of relaxation of the equality constraint represented by the assigned task path or trajectory [95, 96]. In [97, 98], planning is performed over an approximation of the constrained configuration space.

All the above methods attempt to satisfy only the soft task constraint throughout the motion. We argue, however, that it would be interesting (and presumably practical) to develop an opportunistic planner which is capable of satisfying the hard

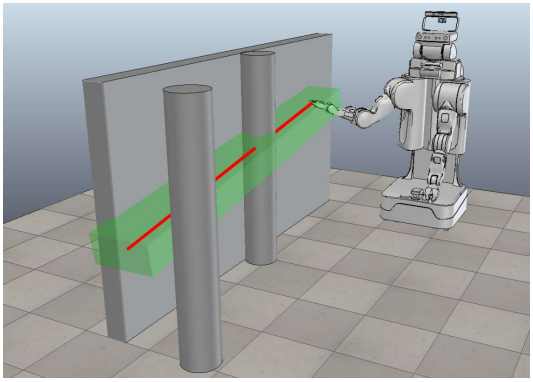

Figure 5.1. In the problem of interest, the robot is assigned a soft task specified by a desired path (red line) and a tolerance (green volume).

constraint whenever possible and of exploiting the available tolerance only when needed. In particular, deviations from the given task path should only take place in the presence of obstructions in the constrained configuration space, such as a narrow or closed passage. Narrow passages are a well-known problematic issue for sampling-based planners; among the most successful methods to handle them, one may mention Gaussian sampling [99] and the bridge test [100]; see [101, 102] for reviews on the topic. Very few works look at narrow passages in a task-constrained setting; one example is [103].

In this chapter we present a motion planner for the case of soft task constraints that is, for the first time, opportunistic in the sense defined above. To this end, it consists of the following components:

- a hard planner for generating collision-free motions that realize exactly the desired task path, essentially an adaptation of the control-based randomized method for TCMP in [39];
- a heuristic criterion to detect obstructions to the hard planner due to narrow/closed passages in the constrained configuration space;
- a soft planner for generating collision free-motions that are compliant with the soft task and allow to bypass the obstructions detected by the hard planner;
- another heuristic criterion to estimate when the obstructions to hard planning have been removed.

We present the method and the obtained results for the case of free-flying robots, i.e., robots that are not subject to nonholonomic constraints, and then discuss a possible extension to the case of humanoid robots.

This chapter is organized as follows. Sect. 5.1 provides a precise formulation of the considered planning problem. An overview of the proposed opportunistic planner is given in Sect. 5.2, while the hard and soft subplanners are described in Sects. 5.3 and 5.4, respectively. Several planning experiments for the PR2 mobile manipulator are shown in Sect. 5.5. Sect. 5.6 describes the planner extension to humanoids. Finally, in Sect. 5.7 we discuss some possible future developments.

5.1 Problem formulation

Consider a robot whose configuration $\boldsymbol{q}$ takes values in a n_q-dimensional configuration space $\mathcal{C}$. The robot moves in a workspace $\mathcal{W} \subseteq \mathbb{R}^3$ containing fixed obstacles. Denote by $\mathcal{O} \subset \mathcal{W}$ and $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$, respectively, the volume occupied by the obstacles and by the robot at configuration $\boldsymbol{q}$, and by $\mathcal{C}_{\text{free}}$ the free configuration space.

Assume that the robot is free-flying (i.e., its configuration can move arbitrarily in $\mathcal{C}$), so that its kinematic model consists of simple integrators. In order to plan paths, we use a geometric version [94] of such model, expressed as $\boldsymbol{q}' = \tilde{\boldsymbol{v}}$, where $()' = d()/ds$ denotes the derivative w.r.t. the path parameter s. This equation entails that the tangent vectors to any path $\boldsymbol{q}(s)$ in $\mathcal{C}$ can be chosen arbitrarily by specifying the (geometric) inputs $\tilde{\boldsymbol{v}}$.

The task is described in coordinates $\boldsymbol{y}$, taking values in an n_y-dimensional task space $\mathcal{Y}$. Coordinates $\boldsymbol{y}$ and $\boldsymbol{q}$ are related by the forward kinematic map $\boldsymbol{y} = \boldsymbol{k}(\boldsymbol{q})$, which at the tangent vector level becomes $\boldsymbol{y}' = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{q}'$, where $\boldsymbol{J}(\boldsymbol{q}) = d\boldsymbol{k}/d\boldsymbol{q}$ is the task Jacobian. We will assume that $n_q > n_y$, i.e., the robot is kinematically redundant for the assigned task.

In the situation of interest, shown in Fig. 5.1, the robot is assigned a *soft task* defined by

- the *desired task path* $\boldsymbol{y}_d(s)$, with the path parameter s taking values in $[0, 1]$ without loss of generality;
- the *tolerance* $\Delta\boldsymbol{y}(s)$, $s \in [0, 1]$, a positive n_t-vector that represents for each component the maximum admissible deviation of $\boldsymbol{y}$ from $\boldsymbol{y}_d$ at s.

The robot is allowed to exploit the tolerance whenever realizing the desired task exactly is difficult or impossible, due to the presence of narrow or closed passages in $\mathcal{C}$. The motion planner must be able to identify such situations automatically in order to act accordingly.

Let $\boldsymbol{e}(\boldsymbol{q}, s) = \boldsymbol{y}_d(s) - \boldsymbol{k}(\boldsymbol{q}(s))$ be the task error associated to configuration $\boldsymbol{q}$ at s. In the following, we say that a configuration $\boldsymbol{q}$ is *compliant with the soft task* if[1]

$$|\boldsymbol{e}(\boldsymbol{q}, s)| \leq \Delta\boldsymbol{y}(s), \text{ for some } s \in [0, 1]. \tag{5.1}$$

A configuration $\boldsymbol{q}$ is *compliant with the hard task* if $\boldsymbol{e}(\boldsymbol{q}, s) = 0$ for some $s \in [0, 1]$ (i.e., if it realizes one sample of the desired task path).

Soft-Task-Constrained Motion Planning is the problem of finding a configuration-space path $\boldsymbol{q}(s)$, $s \in [0, 1]$, such that for all s:

- $\boldsymbol{q}(s)$ is compliant with the soft task (deviations from the desired path are within the tolerance).
- $\mathcal{R}(\boldsymbol{q}(s)) \cap \mathcal{O} = \emptyset$ (collisions are avoided);
- self-collisions, singularities and joint limits are also avoided.

[1] This inequality and similar ones in this chapter are meant componentwise.

Although this is not made explicit in the above formulation, we would obviously like the solution to comply with the hard task *as much as possible*. Also, we assume that the initial configuration $\boldsymbol{q}_{\text{ini}}$ is assigned, with $\boldsymbol{k}(\boldsymbol{q}_{\text{ini}}) = \boldsymbol{y}_d(0)$, while the final configuration $\boldsymbol{q}_{\text{fin}} = \boldsymbol{q}(1)$ will result from planning.

Configurations that are compliant with the hard task make up a $(n_q - n_y)$-dimensional submanifold of $\mathcal{C}$, denoted by $\mathcal{C}_{\text{hard}}$, which naturally decomposes as a *foliation*. In fact, it is $\mathcal{C}_{\text{hard}} = \cup_{s\in[0,1]}\mathcal{L}(s)$, with the generic *leaf* defined as

$$\mathcal{L}(s) = \{\boldsymbol{q} \in \mathcal{C} : \boldsymbol{e}(\boldsymbol{q}, s) = 0\}.$$

The subset $\mathcal{C}_{\text{soft}}$ of configurations that are compliant with the soft task is instead n_q-dimensional. By letting

$$\mathcal{S}(s) = \{\boldsymbol{q} \in \mathcal{C} : |\boldsymbol{e}(\boldsymbol{q}, s)| \leq \Delta\boldsymbol{y}(s)\}$$

we can write $\mathcal{C}_{\text{soft}} = \cup_{s\in[0,1]}\mathcal{S}(s)$. However, this is not a foliation because the $\mathcal{S}$ subsets are not disjoint: a configuration will in general belong to multiple subsets $\mathcal{S}$. Clearly, we have $\mathcal{C}_{\text{hard}} \subset \mathcal{C}_{\text{soft}}$ and $\mathcal{L}(s) \subset \mathcal{S}(s)$, for all $s \in [0, 1]$.

5.2 Overview of the opportunistic planner

The proposed planner builds a tree $\mathcal{T}$ in $\mathcal{C}_{\text{soft}} \cap \mathcal{C}_{\text{free}}$, with configurations as vertexes and collision-free subpaths as edges. To this end, we make use of $N + 1$ samples of the desired task path $\boldsymbol{y}_d(s)$, corresponding to the equispaced sequence $\{s_0 = 0, s_1, ..., s_{N-1}, s_N = 1\}$. Henceforth, we use the shorthand notation $\mathcal{L}_i = \mathcal{L}(s_i)$ and $\mathcal{S}_i = \mathcal{S}(s_i)$.

$\mathcal{T}$ is grown by two (sub)planners that alternate depending on the context: the *Hard Planner* (HP) and the *Soft Planner* (SP). The basic difference between them is that HP works in $\mathcal{C}_{\text{hard}}$ and SP in $\mathcal{C}_{\text{soft}}$, i.e., subpaths generated by the first are compliant with the hard task, whereas those generated by the second are compliant with the soft task. Correspondingly, a vertex of $\mathcal{T}$ generated by HP will belong to a single $\mathcal{L}_i$, whereas a vertex generated by SP may belong to one or more $\mathcal{S}_i$, with $i = 1, ..., N$.

The pseudocode of the proposed planner is given in Algorithm 6. Construction of $\mathcal{T}$ starts by rooting it at the initial $\boldsymbol{q}_{\text{ini}}$. At the generic iteration, let h ($h =$

Algorithm 6: Opportunistic Planner

```
AddVertex(T, q_ini);
h ← 0;
repeat
    h ← HP(T, h);
    if h < N then
        h ← SP(T, h);
until h = ∅ or h = N;
if h = N then
    q_ini q_fin ← RetrievePath(T);
    return q_ini q_fin;
return ∅;
```

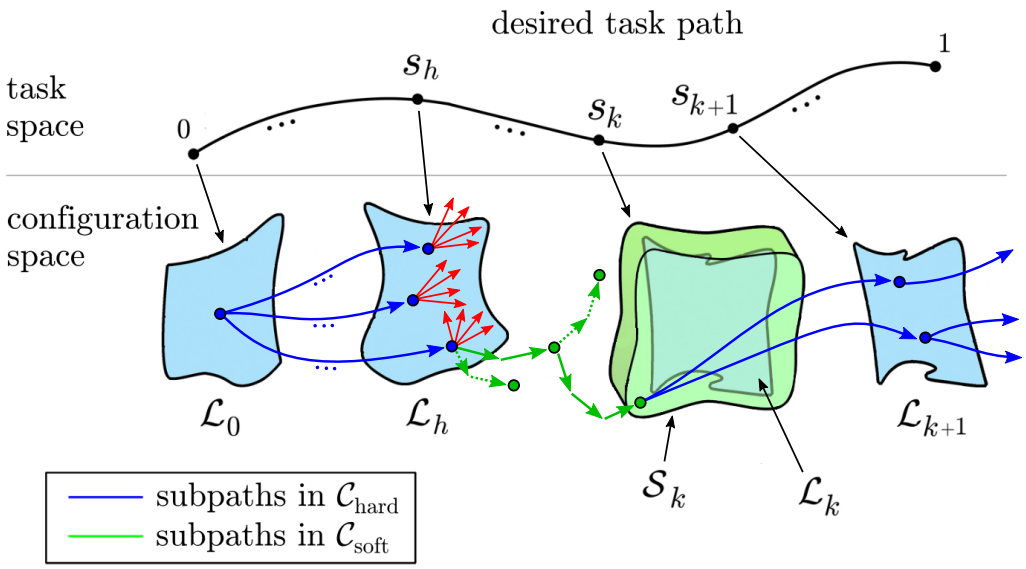

Figure 5.2. The tree $\mathcal{T}$ built by the opportunistic planner (solid blue/green). The portion of $\mathcal{T}$ up to $\mathcal{L}_h$ (solid blue) has been generated by HP and is contained in $\mathcal{C}_{\text{hard}}$. The red arrows originating at vertexes on $\mathcal{L}_h$ indicate a number of failed extension attempts, after which SP has been invoked to extend $\mathcal{T}$ in $\mathcal{C}_{\text{soft}}$. To this end, SP identifies the first task sample s_k where the obstruction has disappeared, and grows an auxiliary tree $\mathcal{T}_{\text{soft}}$ (dashed green) towards $\mathcal{S}_k$. When $\mathcal{T}_{\text{soft}}$ has reached $\mathcal{S}_k$, the subpath from $\mathcal{L}_h$ to $\mathcal{S}_k$ (solid green) is added as an edge to $\mathcal{T}$. At this point, control goes back to HP, and extension of $\mathcal{T}$ in $\mathcal{C}_{\text{hard}}$ is resumed (solid blue).

$0, \ldots, N-1$) be the *frontier index*, i.e., the index of the largest sample of s for which there exists a vertex $\boldsymbol{q}$ in T on $\mathcal{L}_h$ or $\mathcal{S}_h$.

First, HP is invoked in the attempt to extend $\mathcal{T}$ as much as possible in $\mathcal{C}_{\text{hard}}$. When HP stops, it returns an updated value of h. If $h < N$, it means that HP has heuristically identified an obstruction to extending $\mathcal{T}$ from $\mathcal{L}_h$ through subpaths in $\mathcal{C}_{\text{hard}}$; this may indicate that the extension is simply difficult, due to the presence of a narrow passage in $\mathcal{C}_{\text{hard}} \cap \mathcal{C}_{\text{free}}$, or actually impossible, as it happens when an obstacle occupies a portion of the desired task path. At this point, SP is invoked to allow extension of $\mathcal{T}$ from $\mathcal{L}_h$ in $\mathcal{C}_{\text{soft}}$, the rationale being that this may overcome the obstruction. When SP stops, it also returns an updated current value of h, now representing the index associated to the value of s where extension by HP is considered viable again; HP is invoked, and the procedure is repeated (see Fig. 5.2). SP returns $h = \emptyset$ if it does not succeed in extending $\mathcal{T}$ from $\mathcal{L}_h$. The inner loop (lines 3-7 of Algorithm 1) continues until $h = N$ or $h = \emptyset$. In the first case, a configuration $\boldsymbol{q}_{\text{fin}}$ exists in $\mathcal{T}$ such that $\boldsymbol{q}_{\text{fin}} \in \mathcal{L}_N$ or $\boldsymbol{q}_{\text{fin}} \in \mathcal{S}_N$ (depending on whether it has been generated by HP or SP), and a path $\overline{\boldsymbol{q}_{\text{ini}}\boldsymbol{q}_{\text{fin}}}$ can be extracted from $\mathcal{T}$. In the second case, the planner returns a failure.

A remark is in order here about the choice of N, i.e., the number of subsets $\mathcal{L}_i$ (or $\mathcal{S}_i$) used by our planner. While N has no impact on the accuracy with which the desired task path is realized, it is true that larger values of N allow a finer generation

of subpaths in $\mathcal{T}$, ultimately increasing the possibility of navigating the robot among obstacles. However, the size of the tree will grow accordingly, for vertexes will have to be placed on a larger number of subsets $\mathcal{L}_i$ (by HP) or $\mathcal{S}_i$ (by SP). The value of N must therefore represent a reasonable trade-off between maneuvrability of the robot and complexity of planning.

In the following sections, we describe in detail the structure of both the HP and SP planners.

5.3 Hard planner

HP is essentially an adaptation of the control-based planner proposed in [39], with the addition of a heuristic-based mechanism for detecting obstructions to further tree extension in $\mathcal{C}_{\text{hard}}$. The pseudocode of HP is given in Algorithm 7.

At each iteration, a random configuration $\boldsymbol{q}_{\text{rand}}$ is generated in $\mathcal{C}_{\text{hard}}$. The tree $\mathcal{T}$ is then searched for the closest vertex $\boldsymbol{q}_{\text{near}}$ to $\boldsymbol{q}_{\text{rand}}$; call s_j the path parameter value associated to the leaf $\mathcal{L}_j$ where $\boldsymbol{q}_{\text{near}}$ lies. At this point, a subpath starting from $\boldsymbol{q}_{\text{near}}$ and leading to a new configuration $\boldsymbol{q}_{\text{new}} \in \mathcal{L}_{j+1}$ is produced by numerical integration of $\boldsymbol{q}' = \tilde{\boldsymbol{v}}$ from s_j to s_{j+1} under the following motion generation scheme:

$$\tilde{\boldsymbol{v}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})(\boldsymbol{y}'_d + \boldsymbol{K}\boldsymbol{e}(\boldsymbol{q})) + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\tilde{\boldsymbol{w}}, \tag{5.2}$$

where $\boldsymbol{J}^{\dagger}$ is the pseudoinverse of the task Jacobian $\boldsymbol{J}$, $\boldsymbol{K}$ is a positive definite gain matrix, $\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}$ is the orthogonal projection matrix in the null space of $\boldsymbol{J}$, and $\tilde{\boldsymbol{w}}$ is a n_q-dimensional vector that represents a *residual* geometric input (it can be chosen arbitrarily without affecting task execution). To allow effective exploration of the planning space, $\tilde{\boldsymbol{w}}$ is randomly generated with bounded norm. Subpaths generated from $\mathcal{C}_{\text{hard}}$ via (5.2) are guaranteed to remain in $\mathcal{C}_{\text{hard}}$; whereas they converge exponentially to $\mathcal{C}_{\text{hard}}$ if $\boldsymbol{q}_{\text{near}}$ is outside it.

Once it has been generated, the subpath $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ is validated, i.e., it is checked for collision with obstacles, self-collisions, singularities[2] and violation of joint limits. If none of the above occurs, the subpath is *valid* and we add $\boldsymbol{q}_{\text{new}}$ and $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ to $\mathcal{T}$ as, respectively, a new vertex and edge. If $\boldsymbol{q}_{\text{new}}$ belongs to a leaf on which there are no other vertexes (i.e., if $\boldsymbol{q}_{\text{new}} \in \mathcal{L}_{h+1}$), the frontier index h is increased. If the subpath is not valid, the failure counter m_{fail} associated to $\boldsymbol{q}_{\text{near}}$ is increased.

The above extension procedure is iterated until the frontier index h reaches N, or an obstruction is detected to further extension of $\mathcal{T}$. As explained in the previous section, the latter may be due to a narrow or even closed passage in $\mathcal{C}_{\text{hard}} \cap \mathcal{C}_{\text{free}}$. To identify such situations, a heuristic criterion is used. In particular, HP will assume that an obstruction exists if both the following conditions are satisfied (see Fig. 5.3):

- the number m_{fron} of vertexes on the frontier leaf $\mathcal{L}_h$ has reached a threshold value $m_{\text{fron}}^{\max}$, indicating that $\mathcal{L}_h$ has been sufficiently explored;

[2] We have chosen to discard singular configurations in HP for two reasons. First, most singularities of redundant robots are avoidable, in the sense that the desired task path can be executed by a different joint space motion. The second reason is that, while it is true that a singularity-robust pseudoinverse would allow to go through unavoidable singularities, this would produce an error with respect to the desired path; by discarding singularities, we entrust SP to intervene and produce such deviation if necessary to solve the problem.

Algorithm 7: HP($\mathcal{T}$, h)

repeat
  $\boldsymbol{q}_{\text{rand}} \leftarrow$RandomConfig($\mathcal{C}_{\text{hard}}$);
  $\boldsymbol{q}_{\text{near}} \leftarrow$NearestVertex($\mathcal{T}, \boldsymbol{q}_{\text{rand}}$);
  $(\boldsymbol{q}_{\text{new}}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}) \leftarrow$Extend($\mathcal{T}, \boldsymbol{q}_{\text{near}}$);
  if Valid($\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$) **then**
    Add($\mathcal{T}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}, \boldsymbol{q}_{\text{new}}$);
    if $\boldsymbol{q}_{\text{new}} \in \mathcal{L}_{h+1}$ **then**
      $h \leftarrow h + 1$;
  else
    $m_{\text{fail}}(\boldsymbol{q}_{\text{near}}) \leftarrow m_{\text{fail}}(\boldsymbol{q}_{\text{near}}) + 1$;
until ($m_{\text{fron}} \geq m_{\text{fron}}^{\max}$ **and** $m_{\text{fail}}(\boldsymbol{q}_j) \geq m_{\text{fail}}^{\max}$, for all $\boldsymbol{q}_j \in \mathcal{L}_h$) **or** $h = N$;
return h;

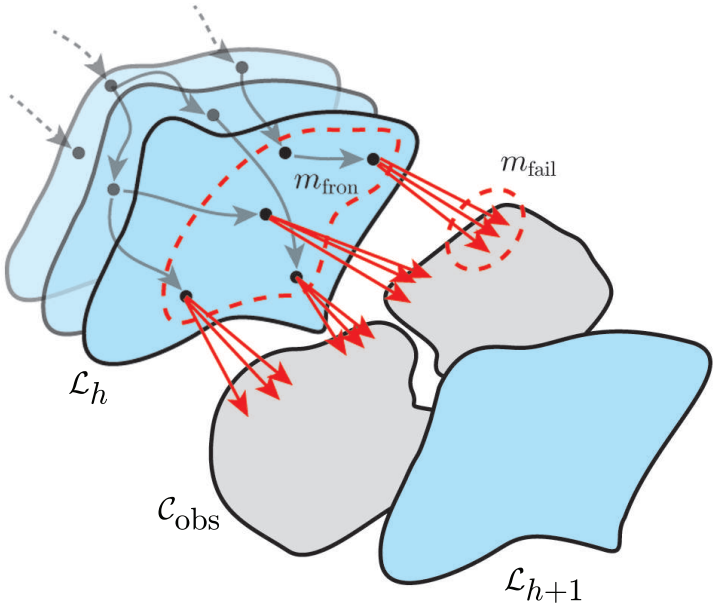

Figure 5.3. The proposed heuristic criterion detects an obstruction if (1) the number m_{fron} of vertexes on the frontier leaf has reached a threshold $m_{\text{fron}}^{\max}$, and (2) the number $m_{\text{fail}}(\boldsymbol{q}_j)$ of failed extension attempts from each such vertex $\boldsymbol{q}_j$ has reached a threshold $m_{\text{fail}}^{\max}$.

- for each vertex $\boldsymbol{q}_j$ on $\mathcal{L}_h$, the number of failed expansions $m_{\text{fail}}(\boldsymbol{q}_j)$ from $\boldsymbol{q}_j$ has reached a threshold value $m_{\text{fail}}^{\max}$, implying that a sufficient number of extensions have been attempted from the vertex (and therefore kinematic redundancy has been fully exploited).

Upon detecting an obstruction, HP returns control to the main planner with h as frontier index.

5.4 Soft planner

SP, whose pseudocode is given in Algorithm 8, is invoked when HP detects an obstruction to further extension of $\mathcal{T}$ in $\mathcal{C}_{\text{hard}}$ from $\mathcal{L}_h$. SP first identifies the index k ($k = h+1, \ldots, N$) associated to the first task sample s_k where the obstruction disappears, and then grows an auxiliary tree $\mathcal{T}_{\text{soft}}$ in $\mathcal{C}_{\text{soft}}$ to connect $\mathcal{L}_h$ to $\mathcal{S}_k$. Once a connecting subpath has been found, it is extracted from $\mathcal{T}_{\text{soft}}$ and added as an edge to $\mathcal{T}$. At this point, HP resumes planning in $\mathcal{C}_{\text{hard}}$ (Fig. 5.2).

Similarly to HP for detecting obstructions, also SP uses a heuristic criterion to identify k. In particular, a fixed number of inverse kinematics solutions is generated in $\mathcal{L}_{h+1}$. If the number m_{free} of collision-free configurations among them is larger than a given threshold $m_{\text{free}}^{\min}$, then $k = h+1$; otherwise, the procedure is repeated for $\mathcal{L}_{h+2}$, and so forth. If h reaches N, it means that SP will operate until task termination as planning in $\mathcal{C}_{\text{hard}}$ can never be resumed.

Once k has been identified, SP starts to grow an auxiliary tree $\mathcal{T}_{\text{soft}}$ in $\mathcal{C}_{\text{soft}}$, whose root is chosen at a random vertex $\boldsymbol{q}_h$ of $\mathcal{T}$ lying on $\mathcal{L}_h$. At each iteration, a random configuration $\boldsymbol{q}_{\text{rand}}$ is generated in $\mathcal{C}_{\text{free}}$ and its closest vertex $\boldsymbol{q}_{\text{near}}$ in $\mathcal{T}_{\text{soft}}$ is found. Then, the SP extension procedure (described in detail below) is invoked to produce a subpath $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ which is valid and complies with the soft task for increasing values of s. If successful, $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ and $\boldsymbol{q}_{\text{new}}$ are added to $\mathcal{T}_{\text{soft}}$ as an edge and a vertex, respectively. This procedure is iterated until one of the following conditions is met:

- $\boldsymbol{q}_{\text{new}} \in \mathcal{S}_k$, i.e., $\boldsymbol{q}_{\text{new}}$ is compliant with the soft task at s_k. In this case, the

Algorithm 8: SP($\mathcal{T}$, h)

```
k ← h;
repeat
    k ← k + 1;
    GenerateIKSolutions(m_sol, L_k);
    m_free ← CountCollisionFreeSolutions();
until m_free ≥ m_free^min or k = N;
q_h ← RandomVertex(T, L_h);
AddVertex(T_soft, q_h);
i ← 0;
repeat
    q_rand ← RandomConfig(C_free);
    q_near ← NearestVertex(T_soft, q_rand);
    (q_new, q_near q_new) ← ExtendSoft(q_near, q_rand, k);
    if q_new ≠ ∅ then
        Add(T_soft, q_near q_new, q_new);
    i ← i + 1;
until q_new ∈ S_k or i = i_max;
if q_new ∈ S_k then
    q_h q_new ← RetrievePath(T_soft);
    Add(T, q_h q_new, q_new);
    return k;
return ∅;
```

subpath $\overline{\boldsymbol{q}_h\boldsymbol{q}_{\text{new}}}$ is extracted from $\mathcal{T}_{\text{soft}}$ and added to $\mathcal{T}$ together with $\boldsymbol{q}_{\text{new}}$ as, respectively, a new edge and a new vertex.

- A maximum number $i_{\max}$ of extension attempts is exceeded. In this case, a failure is reported.

5.4.1 Soft tree extension

The SP extension procedure, whose pseudocode is given in Procedure 7, generates a subpath $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ in $\mathcal{C}_{\text{soft}}$ through a sequence of configurations that are valid and compliant with the soft task for increasing values of s. To this end, it uses a fine discretization of the $[s_h, s_k]$ interval with a step δs, producing the equispaced sequence $\{s_h, s_h + \delta s, \ldots, s_h + M \cdot \delta s = s_k\}$, with $M = (s_k - s_h)/\delta s$. Every new configuration generated during extension will be associated to a unique value of s within the above sequence.

Extension begins by taking a step of length η from $\boldsymbol{q}_{\text{near}}$ towards a random configuration $\boldsymbol{q}_{\text{rand}}$; let $\boldsymbol{q}_{\text{curr}}$ be the generated configuration and s_{curr} the associated parameter sample, i.e., the smallest value in the subsequence $\{s_{\text{near}} + \delta s, \ldots, s_k\}$ for which $\boldsymbol{q}_{\text{curr}}$ is compliant with the soft task:

$$s_{\text{curr}} = \min s \in \{s_{\text{near}} + \delta s, \ldots, s_k\} : \boldsymbol{q}_{\text{curr}} \in \mathcal{S}(s), \tag{5.3}$$

where s_{near} is the parameter sample associated to $\boldsymbol{q}_{\text{near}}$.

Then, an iterative procedure starts aimed at generating a subpath in $\mathcal{C}_{\text{soft}}$ from $\boldsymbol{q}_{\text{curr}}$ towards $\mathcal{S}_k$. At the generic iteration, a task error is computed for the current configuration $\boldsymbol{q}_{\text{curr}}$ giving a small increase δs to the associated path parameter value s_{curr}:

$$\boldsymbol{e}(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s) = \boldsymbol{y}_d(s_{\text{curr}} + \delta s) - \boldsymbol{k}(\boldsymbol{q}_{\text{curr}}). \tag{5.4}$$

Then, a descent direction for the task error in configuration space is computed as

$$\boldsymbol{d} = -\frac{\boldsymbol{J}^T(\boldsymbol{q}_{\text{curr}})\boldsymbol{e}(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s)}{\left\|\boldsymbol{J}^T(\boldsymbol{q}_{\text{curr}})\boldsymbol{e}(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s)\right\|}, \tag{5.5}$$

Procedure 7: ExtendSoft($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}, k$)

$\boldsymbol{q}_{\text{curr}} \leftarrow \boldsymbol{q}_{\text{near}} + \eta \frac{\boldsymbol{q}_{\text{rand}} - \boldsymbol{q}_{\text{near}}}{\|\boldsymbol{q}_{\text{rand}} - \boldsymbol{q}_{\text{near}}\|}$;
compute s_{curr} (5.3);
repeat
  compute $\boldsymbol{e}(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s)$(5.4);
  compute $\boldsymbol{d}$ (5.5);
  $\boldsymbol{q}_{\text{curr}} \leftarrow \boldsymbol{q}_{\text{curr}} + \eta\boldsymbol{d}$;
  compute s_{curr} (5.3);
until $s_{\text{curr}} = s_k$ **or** $s_{\text{curr}} = \emptyset$ **or** !Valid($\boldsymbol{q}_{\text{curr}}$);
if $s_{\text{curr}} = s_k$ **then**
  return $[\boldsymbol{q}_{\text{curr}}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{curr}}}]$;
return $[\boldsymbol{q}_{\text{curr}}.\text{parent}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{curr}}.\text{parent}}]$;

where $\boldsymbol{J}^T(\boldsymbol{q}_{\text{curr}})$ is the transpose of the task Jacobian. The current configuration is then updated by taking a step of length η in the direction $\boldsymbol{d}$, and the cycle continues.

The cycle is interrupted in the following cases:

1. the current configuration $\boldsymbol{q}_{\text{curr}}$ belongs to $\mathcal{S}_k$, i.e., $s_{\text{curr}} = s_k$;
2. $\boldsymbol{q}_{\text{curr}}$ is not compliant with the soft task;
3. $\boldsymbol{q}_{\text{curr}}$ is not valid.

In case 1, a subpath starting from $\boldsymbol{q}_h$ and leading to a configuration in $\mathcal{S}_k$ has been found, and is returned to SP together with the configuration itself. In cases 2-3, the subpath generated so far cannot be further extended without violating the validity requirements and the compliance with the soft task; therefore, only the portion of the subpath leading to the parent configuration of $\boldsymbol{q}_{\text{curr}}$ is returned to SP.

5.5 Planning experiments

We have implemented the proposed opportunistic planner in the V-REP simulation environment on an Intel Core i7-8700K CPU running at 3.7 GHz. The chosen robotic platform is the PR2 mobile manipulator, which consists of an omnidirectional base, a liftable torso, and two arms with 8 DOFs each.

We present planning experiments obtained in four different scenarios. The task is always assigned in terms of the position of the robot right end-effector, while the left arm is kept frozen. Hence, the robot configuration $\boldsymbol{q}$ consists of the planar position and orientation of the base, the torso height, and the joint coordinates of the right arm, for a total of $n_q = 12$ generalized coordinates.

The same parameters are used in all scenarios:

- the samples of the desired task path are $N + 1 = 11$;
- $\boldsymbol{K} = 10 \cdot \boldsymbol{I}_{3\times 3}$ in the motion generation (5.2), which is integrated with Euler method and a stepsize of 0.002;
- HP detects the presence of a planning obstruction by setting $m_{\text{fron}}^{\max} = m_{\text{fail}}^{\max} = 5$, while SP identifies its absence using $m_{\text{sol}} = 100$ and $m_{\text{free}}^{\min} = 20$;
- SP extension works with $\eta = 0.01$ and $\delta s = 0.02$.

The tolerance is specified in the local frame[3] of the desired task path, as this is the most simple and intuitive option for the user. Accordingly, compliance with the soft task at a certain configuration is evaluated using (5.1) by expressing the task error in that frame.

For each scenario, we report some snapshots from a solution (Figs. 5.4–5.7) as well as the evolution of the task error along that solution (Fig. 5.8). To better appreciate the quality of the generated motions, we invite the reader to watch the video available at the link `https://youtu.be/Z8Z9sJ-BWkU`, which shows animated clips of the solutions.

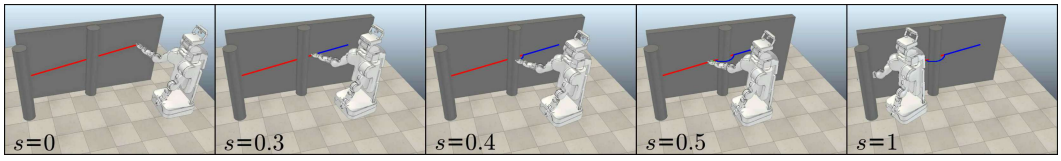

Figure 5.4. Planning scenario 1: snapshots from a solution. The desired and actual task paths are shown in red and blue, respectively. The robot leaves the desired path only when strictly necessary to avoid the pillar.

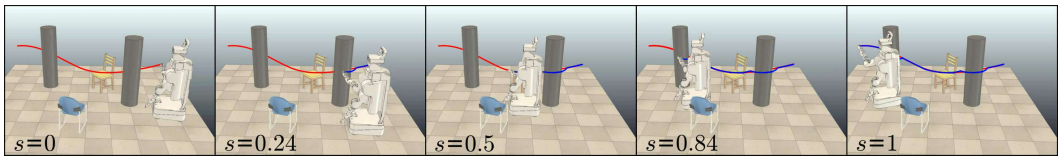

Figure 5.5. Planning scenario 2: snapshots from a solution. The task tolerance is exploited in correspondence of the two portions of the desired path that are obstructed by pillars.

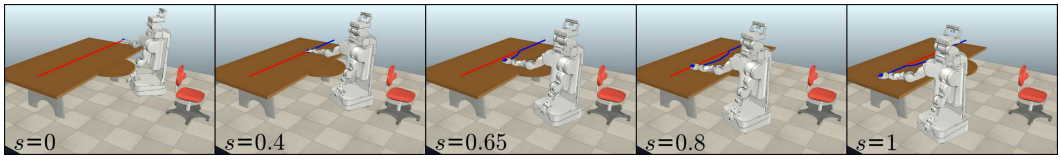

Figure 5.6. Planning scenario 3: snapshots from a solution. The robot carries an object from a location to another above the table, leaving the desired end-effector path only when strictly necessary to avoid collisions between its torso and the table.

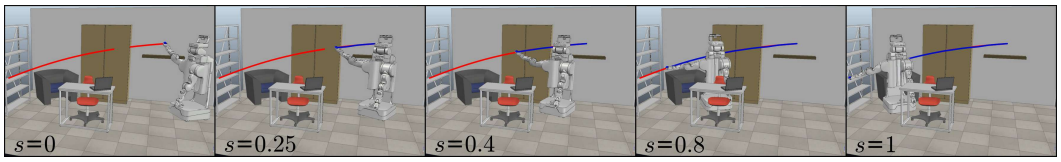

Figure 5.7. Planning scenario 4: snapshots from a solution. Although the environment is quite cluttered, the robot succeeds in carrying the object along the desired path, momentarily deviating from it only in order to avoid the cabinet.

In the first scenario (Fig. 5.4), the desired task path for the end-effector is a line passing through a pillar. The tolerance is specified as $\Delta \boldsymbol{y} = (0.07, 0.2, 0.1)$ m (corresponding to the green volume in Fig. 5.1). In the first part of the motion, HP is able to execute the desired task path (snapshots 1 and 2). In the vicinity of the pillar, HP detects an obstruction and SP is invoked; this leads to an end-effector path that deviates (snapshot 3) from the desired one for $s \in [0.3, 0.5]$, still remaining inside the available tolerance (Fig. 5.8). As soon as SP considers the obstruction to have disappeared, HP takes back control and the robot returns on the desired path (snapshot 5).

The second scenario (Fig. 5.5) is aimed at confirming that the opportunistic planner is able to leave and return to the desired task path multiple times. To

[3] In our implementation, such frame has the origin at $\boldsymbol{y}_d(s)$ and the x- and y-axes oriented, respectively, as $\boldsymbol{y}'_d(s) = (x'_d, y'_d, z'_d)$ and $(y'_d, -x'_d, 0)$; the z-axis is consequently defined. With this choice, the x-axis is always tangent to $\boldsymbol{y}_d(s)$ and oriented along the direction induced on $\boldsymbol{y}_d(s)$ by s.

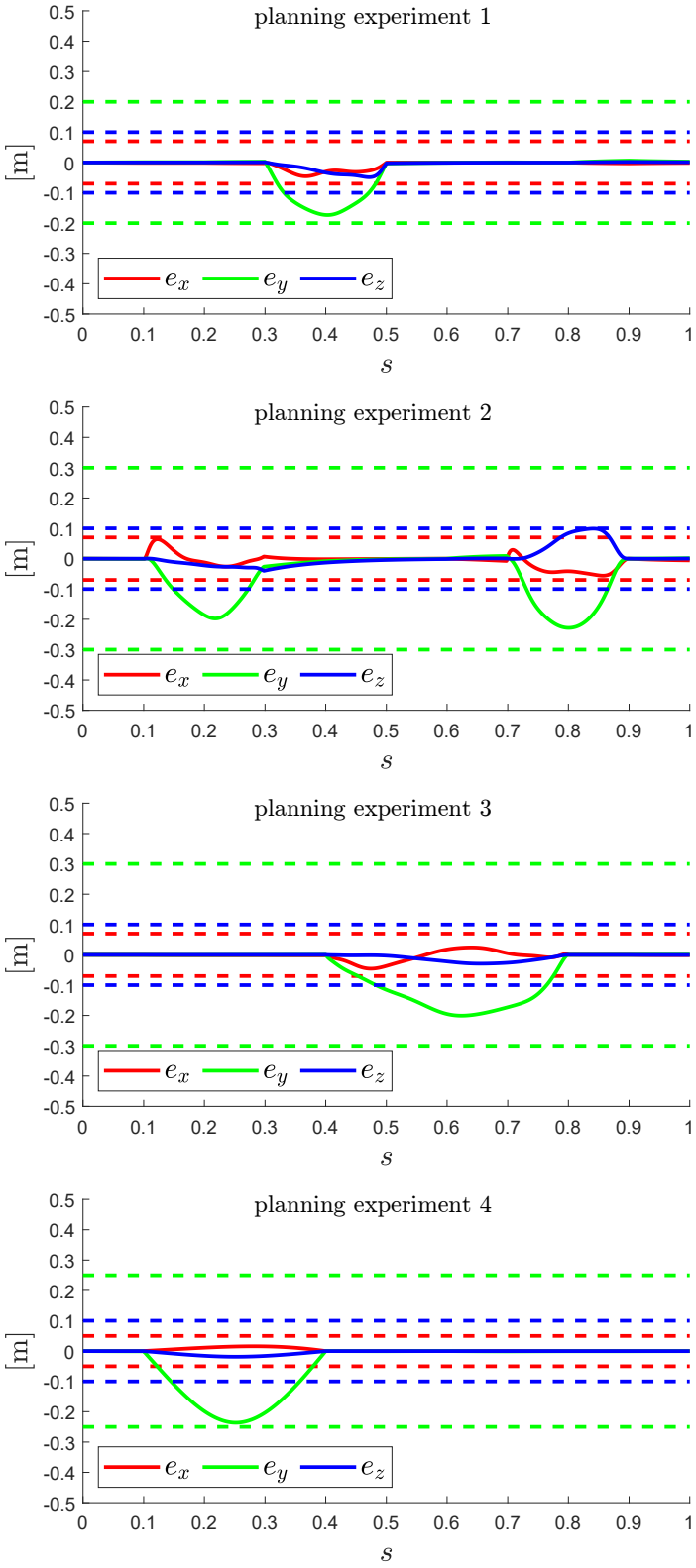

Figure 5.8. Evolution of the task error $\boldsymbol{e}$ in the four planning experiments. The dashed lines indicate the tolerance for each component.

scenario	planning time (s)	HP invocations	SP invocations	vertexes in T	collision checks
1	2.42	2	1	41	3098
2	7.51	3	2	54	6555
3	6.23	2	1	37	9065
4	13.64	2	1	43	17047

Table 5.1. Planner performance data.

this end, the robot end-effector is assigned a sinusoidal path that passes through two pillars, with the tolerance defined as $\Delta \boldsymbol{y} = (0.07, 0.3, 0.1)$ m. Snapshots 2 and 4 show that the robot correctly exploits the tolerance twice, for $s \in [0.1, 0.3]$ and $s \in [0.7, 0.9]$, while the desired path is realized everywhere else (see also Fig. 5.8).

In the third scenario (Fig. 5.6) the robot must execute a simple pick and place task, i.e., moving a ball from an initial to a final position on the table. In such a situation, the user may specify a very simple (and time-efficient) tentative task path, defined as the line segment joining the two locations above the pick and place positions. This path may be abandoned if necessary, but for safety reasons it is desirable that the object always remains above (but not in contact with) the table during the motion. Such requirements translate to a tolerance specified as $\Delta \boldsymbol{y} = (0.07, 0.3, 0.1)$ m. The results show that the desired path is perfectly executed at the start and at the end, with the robot retracting its end-effector for $s \in [0.4, 0.8]$ (snapshots 2-4) to avoid collision between its torso and the table. As indicated by Fig. 5.8, the task error is always within the tolerance region.

The final scenario (Fig. 5.7) also deals with a pick and place task, with the robot now required to move the ball from a shelf to a desired location on a bookcase. The task is specified through a curved desired path and a tolerance $\Delta \boldsymbol{y} = (0.05, 0.25, 0.1)$ m. An early portion of the desired path goes through a cabinet, while the second part requires the robot to navigate a very cluttered region. As in previous scenarios, the desired path is initially realized (snapshot 1) and, under the action of SP, briefly abandoned for $s \in [0.1, 0.4]$ in order to avoid the cabinet (snapshots 2 and 3). Once such obstruction has been removed, HP takes back control and brings back the robot to the desired path, staying on it until the end in spite of the very limited workspace clearance.

Table 5.1 reports some performance data averaged over 20 runs of the opportunistic planner in each scenario.

5.6 Extension to humanoid robots

In this section we present a possible extension of the described opportunistic planning strategy to the case of humanoid robots. Consider that the humanoid is assigned a soft task described in coordinates $\boldsymbol{y}$ that express the position and/or orientation of one of the hands. The complexity of the problem, in this case, is clearly higher than that of the case of a free-flying robot considered in Sect. 5.1. In fact, as discussed in Chap. 2, humanoids can displace their base only by stepping, hence suitable whole-body motions must be generated in order to complete the task.

To construct the tree $\mathcal{T}$, the proposed approach relies on the concept of CoM movement primitives introduced in Chap. 3. In particular, a vertex in $\mathcal{T}$ contains a configuration with an associated time instant and, in addition, the CoM primitive through which it has been generated. An edge between two vertexes is a whole-body motion that realizes a CoM movement selected from a set of primitives U and simultaneously accomplishes a portion of the task, i.e., it is compliant with the hard or soft task, depending on whether it has been generated via HP or SP, respectively.

While the structure of the main planner shown in Algorithm 6 remains identical, appropriate adaptations must be devised for the extension mechanisms of both HP and SP, which correspond to lines 4 and 13 of Algorithms 7 and 8, respectively.

In HP, once the configuration $\boldsymbol{q}_{\text{near}}$ is identified, its associated path parameter value s_j and time instant t_l are retrieved. At this point, a CoM movement primitive $\boldsymbol{u}_{\text{CoM}}$ is randomly selected from U among those that are admissible at $\boldsymbol{q}_{\text{near}}$. Let T_l be the duration of $\boldsymbol{u}_{\text{CoM}}$. Then, the motion generator described in Sect. 3.2.1 is invoked with the aim of generating a whole-body motion $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$, in the time interval $[t_l, t_l + T_l]$, which realizes the reference trajectories $\boldsymbol{y}^*_{\text{CoM}}$ and $\boldsymbol{y}^*_{\text{swg}}$ specified by $\boldsymbol{u}_{\text{CoM}}$, and is compliant with the hard task in the interval $[s_j, s_{j+1}]$. To this end, a continuous time history for the path parameter $s(t) : [t_l, t_l + T_l] \to [s_j, s_{j+1}]$ can be arbitrarily assigned, with the constraints that $s(t_l) = s_j$ and $s(t_l + T_l) = s_{j+1}$. For example, the simplest option consists in choosing a linear time history as

$$s(t) = s_j + \frac{t - t_l}{T_l}(s_{j+1} - s_j). \tag{5.6}$$

The motion generator proceeds by integrating the joint velocities produced by (3.4), where $\boldsymbol{y}_A = \boldsymbol{y}$, $\boldsymbol{y}^*_A = \boldsymbol{y}_d$, $\dot{\boldsymbol{y}}^*_A = \dot{\boldsymbol{y}}_d$ and the null-space vector $\boldsymbol{v}_0$ is chosen as in (3.7).

The extension procedure of SP (see Procedure 8) generates a sequence of whole-body motions, each one realizing a certain CoM primitive. To this purpose, it proceeds iteratively. At each iteration, a CoM movement primitive $\boldsymbol{u}_{\text{CoM}}$ is randomly selected from U among those that are admissible at $\boldsymbol{q}_{\text{prev}}$, i.e., the configuration generated at the previous iteration[4]. Let t_l and T_l be, respectively, the time instant associated to $\boldsymbol{q}_{\text{prev}}$ and the duration of $\boldsymbol{u}_{\text{CoM}}$. Then, the motion generator is invoked with the aim of generating a whole-body motion $\overline{\boldsymbol{q}_{\text{prev}}\boldsymbol{q}_{\text{curr}}}$, in the time interval $[t_l, t_l + T_l]$, which realizes the reference trajectories $\boldsymbol{y}^*_{\text{CoM}}$ and $\boldsymbol{y}^*_{\text{swg}}$ specified by $\boldsymbol{u}_{\text{CoM}}$, and is compliant with the soft task for increasing values of s. To this end, the motion generator proceeds by integrating the joint velocities produced by (3.5), appropriately setting the null-space vector $\boldsymbol{v}_0$.

At the generic integration step, let $\boldsymbol{q}_{\text{curr}}$ be the last generated configuration and s_{curr} its associated path parameter value computed as in (5.3), the null-space vector $\boldsymbol{v}_0$ is computed as

$$\boldsymbol{v}_0 = \eta \frac{\boldsymbol{q}_{\text{rand}} - \boldsymbol{q}_{\text{curr}}}{\|\boldsymbol{q}_{\text{rand}} - \boldsymbol{q}_{\text{curr}}\|} \tag{5.7}$$

during the first invocation of the motion generator, or as

$$\boldsymbol{v}_0 = \eta \boldsymbol{d} \tag{5.8}$$

with $\boldsymbol{d}$ computed as in (5.5), otherwise. The aim is to drive the motion generation towards the random configuration $\boldsymbol{q}_{\text{rand}}$ (5.7), or in the descent direction of the task

[4] At the first iteration $\boldsymbol{q}_{\text{prev}}$ coincides with $\boldsymbol{q}_{\text{near}}$

Procedure 8: PrimitiveBasedExtendSoft($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}, k$)

```
1  q_curr ← q_near;
2  repeat
3  |   q_prev ← q_curr;
4  |   u_CoM ←RandomPrimitive(U, q_prev);
5  |   q_prev q_curr ←MotionGenerator(q_prev, u_CoM);
6  |   compute s_curr (5.3);
7  until s_curr = s_k or s_curr = ∅ or !Valid(q_prev q_curr);
8  if s_curr = s_k then
9  |   return [q_curr, q_near q_curr];
10 return [q_prev, q_near q_prev];
```

error in the configuration space (5.8), without perturbing the exact execution of the primary locomotion task. The cycle is interrupted in one of the three cases described in Sect. 5.4.1. Accordingly, the sequence of generated whole-body motions that are valid and compliant with the soft task is returned to SP.

Some remarks are in order here about two possible limitations of the proposed approach.

- The parameter N must be carefully chosen in such a way to guarantee that each portion of the desired task path between two consecutive samples has a length compatible with the CoM primitives included in the catalogue U. Given a certain primitive, if this length is too large or too small w.r.t. the CoM displacement, the motion generation is more prone to failure, because some joints (especially of the involved arm) will be close to the their limits.

- The complex structure of humanoid robots makes inverse kinematics computations expensive. This may dramatically increase the planning time of the proposed approach, that needs to generate a fixed number of inverse kinematics solutions every time SP is invoked (line 4 of Algorithm 8). To tackle this problem, efficient inverse kinematics solvers should be employed.

5.7 Conclusions

We have considered the problem of planning collision-free motions for redundant robots in the presence of soft task constraints, specified by a desired path in task space with an associated tolerance. The objective was to devise a planner that can realize the desired path for as long as possible, exploiting the tolerance only when strictly needed to avoid a collision.

Our opportunistic approach alternates two subplanners: the first (HP) plans a robot motion that satisfies the hard constraint, until it detects an obstruction based on a heuristic criterion; when this happens, it invokes the second planner (SP), which is only required to satisfy the soft constraint and may therefore be able to bypass the obstruction, giving back control to HP as soon as possible. We implemented the method in V-REP for the PR2 mobile manipulator, presenting successful planning experiments in several scenarios. We have also discussed a possible extension to the

case of humanoid robots. Implementing and testing the proposed extension is part of our future work.

Our approach can also be further developed along several lines, such as *(i)* devising an automatic procedure for tuning the planner parameters, in particular N; *(ii)* extending the method to generic nonholonomic robots; *(iii)* taking into account moving obstacles; *(iv)* generating solutions that are optimal w.r.t. a given performance criterion.

Chapter 6

Multi-contact motion planning

Thanks to their versatile whole-body structure, humanoid robots have the potential for achieving complex tasks that require to sequentially establish multiple contacts with the environment. Differently from tasks that can be achieved through biped locomotion, such as those considered in Chaps. 3 and 4, the tasks considered in this chapter require the simultaneous interaction between different bodies of the robot (not only the feet) and the environment. For example, climbing a staircase using a handrail to keep balance requires to establish contacts with both feet and one hand.

As anticipated in Chap. 2, the problem of generating appropriate motions for achieving this kind of tasks is known as multi-contact motion planning problem. To deal with the problem complexity, most existing approaches adopt the contact-before-motion paradigm: first, a sequence of contact combinations (called stances) is planned, and then a whole-body motion that realizes such sequence, while guaranteeing balance and collision avoidance, is computed.

The first sub-problem is particularly challenging due to the acyclic nature of the sequence of contact combinations, differently from the case of biped locomotion in which the identity of the foot in contact with the ground regularly alternates between the right and left foot. Early approaches, such as [49], deal with such complexity by first creating a stance-adjacency graph based on a set of predesigned possible contacts between robot and environment points, and then searches the graph to find an appropriate sequence of stances. Other methods (e.g, [50, 51]) avoid the specification of predesigned potential contact points, thus allowing contacts anywhere in the environment. These methods first find (via standard RRT or PRM) a guide path for a free-floating robot model that does not collide with obstacles, but keeps the limbs sufficiently close to them. Then, the sequence of stances is computed using a best-first search that exploits the guide path to generate configurations in contact with the environment. A more recent work [104] improves the search efficiency by precomputing a feasible set of configurations for each kinematic chain of the robot. Approaches based on graph or best-first search require the specification of heuristics that drive the search. Such heuristics are generally task and/or scenario dependent, and thus need to be carefully designed according to the considered situation. Moreover, precomputation of potential stances and feasible configurations of limbs are typically very expensive procedures.

In this chapter we propose a method for planning multi-contact motions for a

humanoid robot. The presented method adopts the contact-before-motion paradigm, as in the discussed previous works. The main contribution of this chapter consists in a novel randomized strategy for planning the sequence of stances, which does not require the specification of predesigned potential contacts between robot and environment points, the design of heuristics, or any kind of precomputation.

The work presented in this chapter was made (and is currently in progress) in collaboration with the Humanoids & Human Centered Mechatronics (HHCM) group of the Istituto Italiano di Tecnologia (IIT).

This chapter is organized as follows. Sect. 6.1 introduces some basic notions and assumptions used throughout the chapter. Sect. 6.2 formally describes the motion planning problem and gives an overview of the proposed approach. The strategy proposed for generating the sequence of stances is described in Sect. 6.3, while the generation of whole-body motions throughout such sequence is discussed in Sect. 6.4. Sect. 6.5 presents preliminary results obtained applying the proposed planner to the stand up task of the COMAN+ humanoid. Conclusions and future work are discussed in Sect. 6.6.

6.1 Background

The configuration of a humanoid robot is fully described by

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}_{\text{fb}} \\ \boldsymbol{q}_{\text{jnt}} \end{pmatrix}$$

where $\boldsymbol{q}_{\text{fb}} \in SE(3)$ is the pose of a reference frame attached to a certain link[1] of the robot w.r.t. the inertial world frame and $\boldsymbol{q}_{\text{jnt}}$ is the n-vector of joint angles. Then, the configuration space $\mathcal{C}$ is $(6+n)$-dimensional.

Throughout this chapter, it is assumed that a given subset of h robot points, henceforth referred to as *end-effectors*, is allowed to establish non-coplanar, fixed, point contacts with the environment, i.e., the position of an end-effector in contact with a certain point of the environment is fully constrained, differently from other kinds of contact modes (e.g., sliding or rolling contacts). We will indicate such subset as $\mathcal{H} = \{\mathtt{e}_1, \ldots, \mathtt{e}_h\}$, where each element $\mathtt{e}_i$ is a robot point identifier. As an example, consider the case of a humanoid robot: $\mathcal{H}$ may include only (a representative point of) the feet ($h = 2$), the feet and the hands ($h = 4$), and possibly additional links (e.g., the knees), depending on the task that the robot is required to perform. Position coordinates of a specific end-effector $\mathtt{e}_i$ are related to configuration coordinates by a forward kinematic map $\boldsymbol{p}_{\mathtt{e},i} = \boldsymbol{k}(\mathtt{e}_i, \boldsymbol{q})$, which at differential level becomes $\dot{\boldsymbol{p}}_{\mathtt{e},i} = \boldsymbol{J}_i \dot{\boldsymbol{q}}_{\text{jnt}}$, where $\boldsymbol{J}_i = \boldsymbol{J}(\mathtt{e}_i, \boldsymbol{q})$ is the Jacobian matrix of $\boldsymbol{p}_{\mathtt{e},i}$ w.r.t. $\boldsymbol{q}$.

A *contact* is described by a triple $c = \langle \mathtt{e}, \boldsymbol{p}_{\text{c}}, \boldsymbol{f}_{\text{c}} \rangle$, where $\mathtt{e}$ is the end-effector in contact, $\boldsymbol{p}_{\text{c}}$ is the point of the environment where the contact is established, and $\boldsymbol{f}_{\text{c}} \in \mathbb{R}^3$ is the contact force exerted at $\boldsymbol{p}_{\text{c}}$ by the environment on $\mathtt{e}$. A *stance* is a set $\sigma = \{c_1, \ldots, c_k\}$ of $k \leq h$ contacts. A stance σ uniquely defines a set $\mathcal{A} = \{\mathtt{e}_1, \ldots, \mathtt{e}_k\} \subseteq \mathcal{H}$ of *active* end-effectors, a set $P_{\text{c}} = \{\boldsymbol{p}_{\text{c},1}, \ldots, \boldsymbol{p}_{\text{c},k}\}$ of contact

[1] Note that this is a more general option w.r.t. that adopted in Chap. 3 where such frame, in view of the use of CoM primitives, was attached to the CoM.

positions, and a vector $\boldsymbol{F}_{\mathrm{c}} = (\boldsymbol{f}_{\mathrm{c},1}^T, \ldots, \boldsymbol{f}_{\mathrm{c},k}^T)^T \in \mathbb{R}^{3k}$ of contact forces (and vice versa).

A pair $\langle \sigma, \boldsymbol{q} \rangle$ consisting of a stance σ and a feasible configuration $\boldsymbol{q}$ defines a *multi-contact state*. For a configuration $\boldsymbol{q}$ to be *feasible* at a stance σ, it must realize all the contact positions specified by σ, i.e., $\boldsymbol{k}(\mathtt{e}_i, \boldsymbol{q}) = \boldsymbol{p}_{\mathrm{c},i}$ for all $c_i \in \sigma$, and guarantee static balance[2].

A common choice (see [49]) to evaluate static balance of a humanoid, avoiding the computation of the torques exerted at the joints, is to consider the robot being constituted by a single rigid body (or equivalently that it has no torque limits). Under this assumption, static balance at a multi-contact state $\langle \sigma, \boldsymbol{q} \rangle$ is guaranteed if (i) the contact forces compensate the gravitational forces and (ii) all the contacts remain fixed.

Condition (i) is expressed by the Centroidal Statics model

$$m\boldsymbol{g} + \boldsymbol{G}(\boldsymbol{q})\boldsymbol{F}_{\mathrm{c}} = \boldsymbol{0} \tag{6.1}$$

where m is the total mass of the robot, $\boldsymbol{g} = (\boldsymbol{g}_a^T, \boldsymbol{0}_{3\times1}^T)^T \in \mathbb{R}^6$ with $\boldsymbol{g}_a$ the gravity acceleration, and matrix $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{6\times3k}$ has the form

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{I}_{3\times3} & \cdots & \boldsymbol{I}_{3\times3} \\ \boldsymbol{S}(\boldsymbol{p}_{\mathrm{CoM}} - \boldsymbol{p}_{\mathrm{c},1}) & \cdots & \boldsymbol{S}(\boldsymbol{p}_{\mathrm{CoM}} - \boldsymbol{p}_{\mathrm{c},k}) \end{pmatrix} \tag{6.2}$$

with $\boldsymbol{S}$ a skew-symmetric matrix, and $\boldsymbol{p}_{\mathrm{CoM}}$ the CoM position at $\boldsymbol{q}$. Model (6.1) can be obtained by considering the Centroidal Dynamics model ([52]) under quasi-static conditions, i.e., $\dot{\boldsymbol{q}} = \ddot{\boldsymbol{q}} = \boldsymbol{0}$.

Condition (ii) is satisfied as long as, for each contact $c_i \in \sigma$, the contact force $\boldsymbol{f}_{\mathrm{c},i}$ lies inside the Coulomb friction cone $\mathcal{F}(\boldsymbol{f}_{\mathrm{c},i}, \boldsymbol{n}_{\mathrm{c},i}, \mu_i)$ directed by the unit normal $\boldsymbol{n}_{\mathrm{c},i}$ at point $\boldsymbol{p}_{\mathrm{c},i}$, i.e.,

$$\boldsymbol{f}_{\mathrm{c},i} \cdot \boldsymbol{n}_{\mathrm{c},i} > \bar{f} \text{ and } \|\boldsymbol{f}_{\mathrm{c},i}^t\|_2 \leq \mu_i(\boldsymbol{f}_{\mathrm{c},i} \cdot \boldsymbol{n}_{\mathrm{c},i}) \tag{6.3}$$

where $\bar{f} \geq 0$, $\boldsymbol{f}_{\mathrm{c},i}^t$ is the tangential component of $\boldsymbol{f}_{\mathrm{c},i}$, and μ_i is the static friction coefficient at $\boldsymbol{p}_{\mathrm{c},i}$. In the following we will shortly indicate such condition as $\boldsymbol{f}_{\mathrm{c},i} \in \mathcal{F}(\boldsymbol{f}_{\mathrm{c},i}, \boldsymbol{n}_{\mathrm{c},i}, \mu_i)$.

Given a stance σ, all configurations that are feasible at σ make up a set $\mathcal{F}_\sigma$ that is a submanifol of $\mathcal{C}$. Two stances σ and σ' are *adjacent* if:

- σ and σ' differs from a single contact, i.e., σ' can be reached by either removing $(\sigma \supset \sigma')$ or adding a contact $(\sigma \subset \sigma')$ from σ;
- $\mathcal{F}_\sigma \cap \mathcal{F}_{\sigma'} \neq \emptyset$, i.e., there exists (at least) one configuration $\boldsymbol{q}$, called *transition configuration*, that belongs to both $\mathcal{F}_\sigma$ and $\mathcal{F}_{\sigma'}$. In particular, if $\sigma \subset \sigma'$ $(\sigma \supset \sigma')$, $\boldsymbol{q}$ is a transition configuration if it realizes all the contact positions specified by σ' (σ) and satisfies the static balance conditions using the contacts in σ (σ').

[2]For challenging tasks, such as the stand up task considered in Sect. 6.5, it is reasonable to allow the robot to perform only static motions.

6.2 Problem and approach

In the situation of interest, a humanoid robot is assigned a multi-contact loco-manipulation task, i.e., it must move within the environment by establishing multiple non-coplanar contacts with it. Examples of these tasks include crawling under low obstacles, climbing/descending stairs using a handrail, and standing up exploiting the environment as support.

We assume that the environment is known and static, such that its geometry can be provided as a point cloud $\mathcal{P}$, and that it is composed by smooth surfaces, such that the unit normal $\boldsymbol{n}$ pointing from the environment to the robot can be readily computed at any point $\boldsymbol{p} \in \mathcal{P}$ whenever needed.

In our formulation, the task is specified as a desired final stance σ_{fin} that the robot must reach starting from its multi-contact state $\langle \sigma_{\text{ini}}, \boldsymbol{q}_{\text{ini}} \rangle$ at the initial time instant t_{ini}. To complete the assigned task, the robot is required to perform a sequence of whole-body motions throughout a sequence of stances. Both sequences must be autonomously decided by the robot.

The motion planning problem consists in finding a configuration space trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, that leads to the desired final stance σ_{fin}, i.e., $\boldsymbol{q}_{\text{fin}} = \boldsymbol{q}(t_{\text{fin}})$ realizes all the contact positions specified by σ_{fin}, and satisfies the following three requirements:

R1 A part from contacts between the end-effectors and the environment, collisions and self-collisions are avoided.

R2 Bounds on the joint (1) positions, (2) velocities and (3) accelerations are respected.

R3 The robot maintains static balance at all time instants.

To solve the described problem, we propose a strategy consisting in two sequential (sub)planners: the *multi-contact state planner* and the *whole-body planner*.

The first planner computes a sequence of $N + 1$ multi-contact states leading to the desired final stance σ_{fin}

$$S = \{\langle \sigma_0, \boldsymbol{q}_0 \rangle, \langle \sigma_1, \boldsymbol{q}_1 \rangle, \ldots, \langle \sigma_{N-1}, \boldsymbol{q}_{N-1} \rangle, \langle \sigma_N, \boldsymbol{q}_N \rangle\}$$

where $\langle \sigma_0, \boldsymbol{q}_0 \rangle = \langle \sigma_{\text{ini}}, \boldsymbol{q}_{\text{ini}} \rangle$ and $\langle \sigma_N, \boldsymbol{q}_N \rangle = \langle \sigma_{\text{fin}}, \boldsymbol{q}_{\text{fin}} \rangle$[3]. Moreover, consecutive multi-contact states $\langle \sigma_{i-1}, \boldsymbol{q}_{i-1} \rangle$ and $\langle \sigma_i, \boldsymbol{q}_i \rangle$ in S, $i = 1, \ldots, N$, are such that σ_{i-1} and σ_i are adjacent.

The second planner computes the subtrajectories $\boldsymbol{q}_i(t)$, $t \in [t_{i-1}, t_i]$, appropriately assigning their timing laws, between configurations $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$ in consecutive multi-contact states of S. The concatenation of subtrajectories $\boldsymbol{q}_i(t)$, $i = 1, \ldots, N$, provides the overall trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, in $\mathcal{C}$.

In the following we separately discuss the two planners.

[3]Note that, in our formulation, t_{fin}, $\boldsymbol{q}_{\text{fin}}$, and N are not assigned and will be automatically determined by the planner.

6.3 Multi-contact state planner

This section presents a randomized method for planning the sequence of multi-contact states S which leads to the desired final stance σ_{fin}. Each element $\langle \sigma_i, \boldsymbol{q}_i \rangle$ in S, $i = 1, \ldots, N$, is such that requirements R1, R2.1, R3 are satisfied, and the stance σ_i is adjacent to the stance σ_{i-1} contained in the previous element.

The proposed multi-contact state planner, whose pseudocode is given in Algorithm 6, uses a RRT-like strategy to iteratively construct a tree $\mathcal{T}$ in the search space. In this tree, a vertex $v = \langle \sigma, \boldsymbol{q} \rangle$ consists of a multi-contact state satisfying R1, R2.1, R3. An edge between two vertexes v and v' indicates that stances σ and σ' are adjacent. At the beginning, the tree $\mathcal{T}$ is rooted at vertex $v_{\text{ini}} = \langle \sigma_{\text{ini}}, \boldsymbol{q}_{\text{ini}} \rangle$.

The generic iteration of the algorithm starts by selecting an end-effector e_{rand} and a point $\boldsymbol{p}_{\text{rand}}$. The algorithm is allowed to randomly choose between exploration and exploitation, to bias the growth of the tree towards unexplored regions of the search space and the goal, respectively. In the first case, e_{rand} and $\boldsymbol{p}_{\text{rand}}$ are randomly chosen from the set $\mathcal{H}$ and the workspace, respectively. In the second case, a contact c_{rand} is randomly selected from σ_{fin}, and e_{rand} and $\boldsymbol{p}_{\text{rand}}$ are retrieved from it.

Then, the algorithm assigns to each vertex v in $\mathcal{T}$ a probability that is inversely proportional to the Euclidean distance $\|\boldsymbol{p}_{\text{rand}} - \boldsymbol{k}(\mathsf{e}_{\text{rand}}, \boldsymbol{q})\|_2$ between end-effector e_{rand} at $\boldsymbol{q}$ (the configuration specified by v) and $\boldsymbol{p}_{\text{rand}}$. The resulting probability distribution is used to randomly choose a vertex $v_{\text{near}} = \langle \sigma_{\text{near}}, \boldsymbol{q}_{\text{near}} \rangle$ of $\mathcal{T}$ for a tree expansion attempt.

Once v_{near} has been selected, the algorithm decides whether to attempt an expansion of $\mathcal{T}$ from v_{near} by removing or adding a contact with e_{rand}. This choice is determined by the presence/absence of e_{rand} in the set $\mathcal{A}_{\text{near}}$ of active end-effectors at σ_{near}. According to the decision taken, a candidate set $\mathcal{A}_{\text{cand}}$ of active end-effectors is generated, and a candidate position $\boldsymbol{p}_{\text{cand}}$ for the added contact (if any) is chosen. More precisely:

- If e_{rand} is active at σ_{near} ($\mathsf{e}_{\text{rand}} \in \mathcal{A}_{\text{near}}$), $\mathcal{A}_{\text{cand}}$ is generated by removing e_{rand} from $\mathcal{A}_{\text{near}}$ ($\mathcal{A}_{\text{cand}} = \mathcal{A}_{\text{near}} \setminus \{\mathsf{e}_{\text{rand}}\}$), and $\boldsymbol{p}_{\text{cand}}$ is left unspecified.
- If e_{rand} is not active at σ_{near} ($\mathsf{e}_{\text{rand}} \notin \mathcal{A}_{\text{near}}$), $\mathcal{A}_{\text{cand}}$ is generated by adding e_{rand} to $\mathcal{A}_{\text{near}}$ ($\mathcal{A}_{\text{cand}} = \mathcal{A}_{\sigma}^{\text{near}} \cup \{\mathsf{e}_{\text{rand}}\}$), and $\boldsymbol{p}_{\text{cand}}$ is chosen within the reachable workspace $\mathcal{W}$ of the end-effector e_{rand} at configuration $\boldsymbol{q}_{\text{near}}$. In particular, $\mathcal{W}$ is defined as the set of points of the point cloud $\mathcal{P}$ that lie inside a sphere of radius r centered at the position of e_{rand} at $\boldsymbol{q}_{\text{near}}$[4], and $\boldsymbol{p}_{\text{cand}}$ is chosen as the point in $\mathcal{W}$ that is closest to $\boldsymbol{p}_{\text{rand}}$.

At this point, the algorithm builds a candidate set $P_{\text{c}}^{\text{cand}} = \{\boldsymbol{p}_{\text{c},1}, \ldots, \boldsymbol{p}_{\text{c},k}\}$, with $k = |\mathcal{A}_{\text{cand}}|$, of contact positions associated to the end-effectors specified in $\mathcal{A}_{\text{cand}}$. In particular, the generic contact position $\boldsymbol{p}_{\text{c},i} \in P_{\text{c}}^{\text{cand}}$ associated to the end-effector $\mathsf{e}_i \in \mathcal{A}_{\text{cand}}$ chosen as

$$\boldsymbol{p}_{\text{c},i} = \begin{cases} \boldsymbol{p}_{\text{cand}}, & \text{if } \mathsf{e}_i = \mathsf{e}_{\text{rand}} \\ \boldsymbol{k}(\mathsf{e}_i, \boldsymbol{q}_{\text{near}}), & \text{otherwise.} \end{cases}$$

[4]More sophisticated methods for approximating the reachable workspace (e.g., [105, 106]) can be involved without affecting the overall planning strategy.

Algorithm 9: Multi-Contact State Planner

$v_{\text{ini}} \leftarrow \langle \sigma_{\text{ini}}, \boldsymbol{q}_{\text{ini}} \rangle$;
AddVertex($\mathcal{T}$, v_{ini});
$i \leftarrow 0$;
repeat
  $\mathsf{e}_{\text{rand}} \leftarrow$ PickRandomEndEffector($\mathcal{H}$);
  $\boldsymbol{p}_{\text{rand}} \leftarrow$ PickRandomPoint($\mathbb{R}^3$);
  $v_{\text{near}} \leftarrow$ FindNearestVertex($\mathcal{T}$, e_{rand}, $\boldsymbol{p}_{\text{rand}}$);
  if $\mathsf{e}_{\text{rand}} \in \mathcal{A}_{\text{near}}$ **then**
    $\mathcal{A}_{\text{cand}} \leftarrow \mathcal{A}_{\text{near}} \setminus \{\mathsf{e}_{\text{rand}}\}$;
    $\boldsymbol{p}_{\text{cand}} \leftarrow \emptyset$;
  else
    $\mathcal{A}_{\text{cand}} \leftarrow \mathcal{A}_{\text{near}} \cup \{\mathsf{e}_{\text{rand}}\}$;
    $\mathcal{W} \leftarrow$ ComputeReachableWorkspace($\boldsymbol{q}_{\text{near}}$, e_{rand});
    $\boldsymbol{p}_{\text{cand}} \leftarrow$ PickRandomPoint($\mathcal{W}$, $\boldsymbol{p}_{\text{rand}}$);
  end
  $P_{\text{c}}^{\text{cand}} \leftarrow$ BuildContactPositions($\mathcal{A}_{\text{cand}}$, $\boldsymbol{q}_{\text{near}}$, $\boldsymbol{p}_{\text{cand}}$, e_{rand});
  $\langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle \leftarrow$ MultiContactStateGenerator($\mathcal{A}_{\text{cand}}$, $P_{\text{c}}^{\text{cand}}$);
  if $\langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle \neq \langle \emptyset, \emptyset \rangle$ **then**
    $\boldsymbol{q}_{\text{tran}} \leftarrow$ TransitionConfigurationGenerator(σ_{near}, σ_{new});
    if $\boldsymbol{q}_{\text{tran}} \neq \emptyset$ **then**
      $v_{\text{new}} \leftarrow \langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle$;
      AddVertex($\mathcal{T}$, v_{new}, v_{near});
    end
  end
  $i \leftarrow i + 1$;
until $\sigma_{\text{new}} = \sigma_{\text{fin}}$ **or** $i = i_{\max}$;
if $\sigma_{\text{new}} = \sigma_{\text{fin}}$ **then**
  $S \leftarrow$ RetrieveSolution($\mathcal{T}$);
  return S;
end
return $\emptyset$;

With the aim of producing a new multi-contact state $\langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle$ - realizing the candidate contact positions $P_{\text{c}}^{\text{cand}}$ for the end-effectors in $\mathcal{A}_{\text{cand}}$, and satisfying R1, R2.1, R3 - the multi-contact state generator (described in Sect 6.3.1) is invoked. If it succeeds, adjacency of σ_{new} with respect to σ_{near} must be verified. To this end, the algorithm invokes the transition configuration generator (described in Sect. 6.3.2) to check whether there exists a configuration $\boldsymbol{q}_{\text{tran}}$ that belongs to both feasible sets $\mathcal{F}_{\sigma}^{\text{near}}$ and $\mathcal{F}_{\sigma}^{\text{new}}$ at σ_{near} and σ_{new}, respectively. If the test is passed, a new vertex $v_{\text{new}} = \langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle$ is added to the tree $\mathcal{T}$ as a child of v_{near}.

Planning terminates when the desired final stance is reached, i.e., $\sigma_{\text{new}} = \sigma_{\text{fin}}$, or a maximum number $i_{\max}$ of iterations has been executed. In the first case, the branch joining the root vertex v_{ini} to v_{new} represents the sequence S; then, it is extracted from $\mathcal{T}$ and provided to the whole-body planner. In the second case, the planner returns a failure.

6.3.1 Multi-contact state generator

The multi-contact state generator, whose pseudocode is given in Procedure 9, receives in input a candidate set $\mathcal{A}_{\text{cand}} = \{\mathtt{e}_1, \ldots, \mathtt{e}_k\}$ of k active end-effectors, together with their reference contact positions $P_{\text{c}}^{\text{cand}} = \{\boldsymbol{p}_{\text{c},1}, \ldots, \boldsymbol{p}_{\text{c},k}\}$. It is in charge of generating a multi-contact state $\langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle$ that realizes the reference contact positions $P_{\text{c}}^{\text{cand}}$ for the end-effectors in $\mathcal{A}_{\text{cand}}$, and satisfies requirements R1, R2.1, R3.

The first step consists in generating a candidate configuration $\boldsymbol{q}_{\text{cand}}$ that realizes the reference contact positions $P_{\text{c}}^{\text{cand}}$ for the end-effectors in $\mathcal{A}_{\text{cand}}$, and satisfies requirements R1-R2.1. To this end, it repeatedly invokes the Inverse Kinematics (IK) solver (described in Sect. 6.3.3). As it will be clear in the following, the IK solver is non-deterministic, hence different invocations have in general different outcomes. Such loop is left when the IK solver finds a solution or a predefined time budget ΔT_{IK} runs out. In the last case, a failure is returned to the multi-contact state planner, which resumes control.

If $\boldsymbol{q}_{\text{cand}}$ is successfully generated, let $\boldsymbol{p}_{\text{CoM}}^{\text{cand}}$ be the CoM position at $\boldsymbol{q}_{\text{cand}}$. The Centroidal Statics (CS) solver (described in Sect. 6.3.3) is invoked. It computes the contact forces $F_{\text{c}}^{\text{cand}} = (\boldsymbol{f}_{\text{c},1}^T, \ldots, \boldsymbol{f}_{\text{c},k}^T)^T$, with $\boldsymbol{f}_{\text{c},i}$ the contact force at $\boldsymbol{p}_{\text{c},i}$, and the CoM position $\boldsymbol{p}_{\text{CoM}}^{\text{new}}$, as close as possible to $\boldsymbol{p}_{\text{CoM}}^{\text{cand}}$, which guarantee satisfaction of requirement R3.

At this point, the procedure attempts to generate a new configuration $\boldsymbol{q}_{\text{new}}$ that - in addition to realize the reference contact positions $P_{\text{c}}^{\text{cand}}$ for the end-effectors in $\mathcal{A}_{\sigma}^{\text{cand}}$, and satisfy requirements R1-R2.1 - achieves the CoM position $\boldsymbol{p}_{\text{CoM}}^{\text{new}}$ as well. To this purpose, a loop similar to that described above is performed. Note that, this time, the generated configuration $\boldsymbol{q}_{\text{new}}$ (if any) is statically balanced by construction, thus requirement R3 is satisfied as well.

Procedure 9: MultiContactStateGenerator($\mathcal{A}_{\text{cand}}$, $P_{\text{c}}^{\text{cand}}$)

```
1  t_IK ← 0;
2  repeat
3  |   q_cand ← IKSolver(A_cand, P_c^cand);
4  |   t_IK ← UpdateTime();
5  until q_cand ≠ ∅ or t_IK ≥ ΔT_IK;
6  if q_cand ≠ ∅ then
7  |   [F_c^cand, p_CoM^new] ← CSSolver(P_c^cand, p_CoM^cand);
8  |   t_IK ← 0;
9  |   repeat
10 |   |   q_new ← IKSolver(A_cand, P_c^cand, p_CoM^new);
11 |   |   t_IK ← UpdateTime();
12 |   until q_new ≠ ∅ or t_IK ≥ ΔT_IK;
13 |   if q_new ≠ ∅ then
14 |   |   σ_new ← BuildStance(A_cand, P_c^cand, F_c^cand);
15 |   |   return ⟨σ_new, q_new⟩;
16 |   end
17 end
18 return ⟨∅, ∅⟩;
```

In case generation of a statically balanced $\boldsymbol{q}_{\text{new}}$ succeed, the corresponding stance

σ_{new} is constructed as $\sigma_{\text{new}} = \{c_1^{\text{new}}, \ldots, c_k^{\text{new}}\}$, where $c_i^{\text{new}} = \langle \mathsf{e}_i, \boldsymbol{p}_{\text{c},i}, \boldsymbol{f}_{\text{c},i} \rangle$, with each element appropriately retrieved from $\mathcal{A}_{\text{cand}}$, $P_{\text{c}}^{\text{cand}}$, and $\boldsymbol{F}_{\text{c}}^{\text{cand}}$, respectively. The resulting multi-contact state $\langle \sigma_{\text{new}}, \boldsymbol{q}_{\text{new}} \rangle$ is returned to the multi-contact state planner. Instead, if generation of a statically balanced $\boldsymbol{q}_{\text{new}}$ failed, a failure is returned to the multi-contact state planner.

6.3.2 Transition configuration generator

The transition configuration generator, whose pseudocode is given in Procedure 10, receives in input two stances σ_{near} and σ_{new}. It is in charge of generating a configuration $\boldsymbol{q}_{\text{tran}}$ that satisfies requirements R1-R2.1-R3, and is feasible at both σ_{near} and σ_{new}.

The first step consists in retrieving the set $\mathcal{A}$ of active end-effectors, together with their corresponding contact positions P_{c}, from the stance composed by more contacts; positions P'_{c} of the contacts that are active at the stance composed by less contacts are retrieved as well.

The transition configuration generator proceeds analogously to the multi-contact state generator, with the only difference that configurations are generated (via the IK solver) so as to realize the contact positions P_{c} for the end-effectors in $\mathcal{A}$, while static balance (considered via the CS solver) is obtained by involving only the contacts at the positions P'_{c}. As before, depending on whether $\boldsymbol{q}_{\text{tran}}$ is successfully generated or not, the procedure returns, respectively, $\boldsymbol{q}_{\text{tran}}$ or a failure to the multi-contact state planner.

Procedure 10: TransitionConfigurationGenerator(σ_{near}, σ_{new})

```
if σ_near ⊂ σ_new then
    A ← A_new; P_c ← P_c^new; P'_c ← P_c^near;
else
    A ← A_near; P_c ← P_c^near; P'_c ← P_c^new;
end
t_IK ← 0;
repeat
    q_cand ← IKSolver(A, P_c);
    t_IK ← UpdateTime();
until q_cand ≠ ∅ or t_IK ≥ ΔT_IK;
if q_cand ≠ ∅ then
    [F_c^cand, p_CoM^tran] ← CSSolver(P'_c, p_CoM^cand);
    t_IK ← 0;
    repeat
        q_tran ← IKSolver(A, P_c, p_CoM^tran);
        t_IK ← UpdateTime();
    until q_tran ≠ ∅ or t_IK ≥ ΔT_IK;
    return q_tran;
end
return ∅;
```

6.3.3 Inverse Kinematics and Centroidal Statics solvers

The IK solver receives in input a set $\bar{\mathcal{A}} = \{\mathtt{e}_1, \ldots, \mathtt{e}_k\}$ of k end-effectors, a set $\bar{P}_\mathrm{c} = \{\bar{\boldsymbol{p}}_{\mathrm{c},1}, \ldots, \bar{\boldsymbol{p}}_{\mathrm{c},k}\}$ of associated reference positions, and (possibly) a reference CoM position $\bar{\boldsymbol{p}}_\mathrm{CoM}$. It aims at finding a configuration $\boldsymbol{q}$ that achieves as much as possible the reference positions for the end-effectors and the CoM (if any), and satisfies requirements R1-R2.1. To this end, the IK solver randomly picks a configuration $\boldsymbol{q}_\mathrm{rand}$ from $\mathcal{C}$ and, starting from it, proceeds iteratively by integrating the joint velocities produced by solving the following unconstrained QP problem

$$\min_{\dot{\boldsymbol{q}}_\mathrm{jnt}} \sum_{i=1}^{k} \|\boldsymbol{J}_i \dot{\boldsymbol{q}}_\mathrm{jnt} - \boldsymbol{K}\boldsymbol{e}_i\|^2_{2,w_i^\mathrm{IK}} + \|\boldsymbol{J}_\mathrm{CoM}\dot{\boldsymbol{q}}_\mathrm{jnt} - \boldsymbol{K}\boldsymbol{e}_\mathrm{CoM}\|^2_{2,w_\mathrm{CoM}^\mathrm{IK}}. \tag{6.4}$$

Here, $\boldsymbol{e}_i$, $i = 1, \ldots, k$, and $\boldsymbol{e}_\mathrm{CoM}$ are the tracking errors; $\boldsymbol{K}$ is a positive definite gain matrix; $w_\mathrm{CoM}^\mathrm{IK}$ is set to zero if $\bar{\boldsymbol{p}}_\mathrm{CoM}$ is unspecified. Integration stops when the value of the cost function becomes smaller than a predefined threshold or a maximum number of iterations is reached. In the first case, the last generated configuration $\boldsymbol{q}$ is checked for requirements R1-R2.1. If the test is passed, $\boldsymbol{q}$ is returned. In all the other cases, a failure is reported[5].

The CS solver, that consists in an adaptation of the method proposed in [107], receives in input a set $\bar{P}_\mathrm{c} = \{\bar{\boldsymbol{p}}_{\mathrm{c},1}, \ldots, \bar{\boldsymbol{p}}_{\mathrm{c},k}\}$ of k reference contact positions and a reference CoM position $\bar{\boldsymbol{p}}_\mathrm{CoM}$. In output, it provides the contact forces $\boldsymbol{F}_\mathrm{c} = (\boldsymbol{f}_{\mathrm{c},1}^T, \ldots, \boldsymbol{f}_{\mathrm{c},k}^T)^T$ and the CoM position $\boldsymbol{p}_\mathrm{CoM}$ which guarantee that the robot maintains static balance. To this end, the following NLP problem is solved

$$\begin{aligned} \min_{\boldsymbol{p}_\mathrm{CoM}, \boldsymbol{F}_\mathrm{c}} \quad & \|\boldsymbol{p}_\mathrm{CoM} - \bar{\boldsymbol{p}}_\mathrm{CoM}\|^2_{2,w_\mathrm{CoM}^\mathrm{CS}} + \|\boldsymbol{F}_\mathrm{c}\|^2_{2,w_\mathrm{F}^\mathrm{CS}} \\ \text{subject to} \quad & m\boldsymbol{g} + \boldsymbol{G}(\boldsymbol{p}_\mathrm{CoM})\boldsymbol{F}_\mathrm{c} = \boldsymbol{0}, \\ & \boldsymbol{f}_{\mathrm{c},i} \in \mathcal{F}(\boldsymbol{f}_{\mathrm{c},i}, \boldsymbol{n}_{\mathrm{c},i}, \mu_i) \text{ for } i = 1, \ldots, k. \end{aligned} \tag{6.5}$$

The cost function in (6.5) includes a first term which attempts to bring the CoM position $\boldsymbol{p}_\mathrm{CoM}$ as close as possible to the reference position $\bar{\boldsymbol{p}}_\mathrm{CoM}$, and a second term for regularization purposes. The constraints enforce the conditions (i) and (ii) of Sect. 6.1, respectively, to guarantee static balance. Note that, in the first constraint we have explicitly shown the dependence of matrix $\boldsymbol{G}(\boldsymbol{q})$ on the decision variable $\boldsymbol{p}_\mathrm{CoM}$ only (that determines the nonlinearity of the constraint), as the contact positions specified by $\bar{P}_\mathrm{c}$ yield in (6.2) skew-symmetric matrices of the form $\boldsymbol{S}(\boldsymbol{p}_\mathrm{CoM} - \bar{\boldsymbol{p}}_{\mathrm{c},i})$. Finally, in the second constraint, $\bar{f}$ in (6.3) serves as a design parameter.

[5] Note that, according to (6.4), end-effectors not included in $\bar{\mathcal{A}}$ can assume arbitrary positions. This might potentially lead to the generation of unnatural configurations. To avoid this fact, a suitable option (that we used in our implementation) consists in adding in (6.4) a third term that attempts to keep the $h - k$ end-effectors not included in $\bar{\mathcal{A}}$ as close as possible to their positions attained at configuration $\boldsymbol{q}_\mathrm{near}$ from which the invocation of the IK solver originated. For sake of clarity, we omit explicit discussion of such term, which will be similar to the first in (6.4), but with lower weights.

6.4 Whole-body planner

The whole-body planner is in charge of computing the timed subtrajectories $\boldsymbol{q}_i(t)$, $t \in [t_{i-1}, t_i]$, between configurations $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$ contained in consecutive multi-contact states in the sequence S found by the multi-contact state planner. The overall trajectory $\boldsymbol{q}(t)$, $t \in [t_{\text{ini}}, t_{\text{fin}}]$, representing the solution for the planning problem will be the concatenation of these subtrajectories.

Given two consecutive multi-contact states $\langle \sigma_{i-1}, \boldsymbol{q}_{i-1} \rangle$ and $\langle \sigma_i, \boldsymbol{q}_i \rangle$ in S, the configuration space trajectory to be computed must start at $\boldsymbol{q}_{i-1} \in \mathcal{F}_{\sigma,i-1}$, and end at $\boldsymbol{q}_i \in \mathcal{F}_{\sigma,i}$, passing through a transition configuration $\boldsymbol{q}_{i-1}^{\text{tran}} \in \mathcal{F}_{\sigma,i-1} \cap \mathcal{F}_{\sigma,i}$. Such transition configuration may be stored in the tree produced by the multi-contact state planner in the same vertex containing $\boldsymbol{q}_i$ at the time of its creation, and appropriately retrieved together with the sequence S (for sake of illustration, we avoid explicit discussion of this operation).

In order to find such trajectory we adopt an adaptation of the approach proposed in [108] which works in two phases that we briefly recall in the following (the reader is referred to the original paper for further details).

In the first phase the aim is to find a discrete sequence S_q of configurations leading from $\boldsymbol{q}_{i-1}$ to $\boldsymbol{q}_i$, and passing through a transition configuration $\boldsymbol{q}_{i-1}^{\text{tran}}$. To this end, a simple RRT-based planner is invoked two times in order to produce, respectively, the portion of S_q joining $\boldsymbol{q}_{i-1}$ to $\boldsymbol{q}_{i-1}^{\text{tran}}$, and $\boldsymbol{q}_{i-1}^{\text{tran}}$ to $\boldsymbol{q}_i$. At each iteration, the planner uses a projection mechanism (see [34] for details) to bring the generated configuration in the appropriate submanifold of $\mathcal{C}$, i.e., $\mathcal{F}_{\sigma,i-1}$ and $\mathcal{F}_{\sigma,i}$ for the first and second portion of S_q, respectively. By construction, all configurations in S_q satisfy requirements R1, R2.1, R3.

In the second phase, a time-optimal trajectory satisfying the additional requirements R2.2-R2.3 (respect of joint velocities and acceleration limits) is computed through an optimization-based interpolation using polynomial splines.

6.5 Preliminary results

In this section, we present preliminary results that we obtained applying the multi-contact state planner proposed in Sect. 6.3. The chosen robotic platform is the COMAN+ humanoid robot built at Istituto Italiano di Tecnologia. COMAN+ has 28 degrees of freedom, weights 70 kg and is 1.7 m tall. Moreover, COMAN+ is torque-controlled.

For the IK and CS solvers described in Sect. 6.3.3 we have used, respectively, the whole-body inverse kinematics framework named CartesI/O [109], and the framework presented in [107] which is based on IFOPT, an Eigen-based C++ interface to the Non-linear Programming solver IPOPT [110].

In the considered scenario (see first snapshot in Fig. 6.1), the robot is initially in a quadrupedal configuration, with both feet and hands in contact with the ground. A wall is located nearby the robot, in front of it. The task assigned to the robot consists in standing up from its initial posture. In particular, the task is defined as reaching a final stance specifying desired contact positions for both feet and hands, respectively, on the ground and on the wall. To fulfil the task, the robot can exploit

contacts with any point of the environment using both feet and hands. Then, the set $\mathcal{H}$ contains four end-effectors. In particular, COMAN+ has two rectangular, flat feet and two spherical hands. We placed a local reference frame on each of these end-effectors. For each foot, the local frame has origin at the center of the sole and is oriented so that the xy plane is parallel to the sole. For each hand, the local frame has origin at a point designed as tip (this can be chosen arbitrarily on the sphere) and is oriented so that the xy plane is tangent to the sphere. To increase the chances of the IK solver to find collision-free configurations, we have included in (6.4) an additional term that aims at keeping the z-axis of each active end-effector frame parallel to the normal at the contact point. Note that, differently from the spherical hands which can establish full-fledged point contacts, the feet actually establish surface contacts. In the presented preliminary results, they are approximated to point contacts. The explicit inclusion of surface contacts in the multi-contact state planner is part of our future work (see Sect. 6.6).

The point cloud $\mathcal{P}$ modeling both the ground and the wall has been generated with a resolution of 0.1 m. We set the radiuses of the feet and hands workspaces to 0.3 m and 0.6 m, respectively. The time budget ΔT_{IK} used to search for an inverse kinematics solution in the multi-contact state and transition configuration generators is set to 3 s. The CS solver uses an identical static friction coefficient $\mu = 0.5$ at all point of the environment, and a force threshold $\bar{f}$ of 15 and 65 N for contacts involving hands and feet, respectively.

Since the proposed multi-contact state planner is randomized, to assess its performance we ran it ten times on the considered scenario. The produced sequence

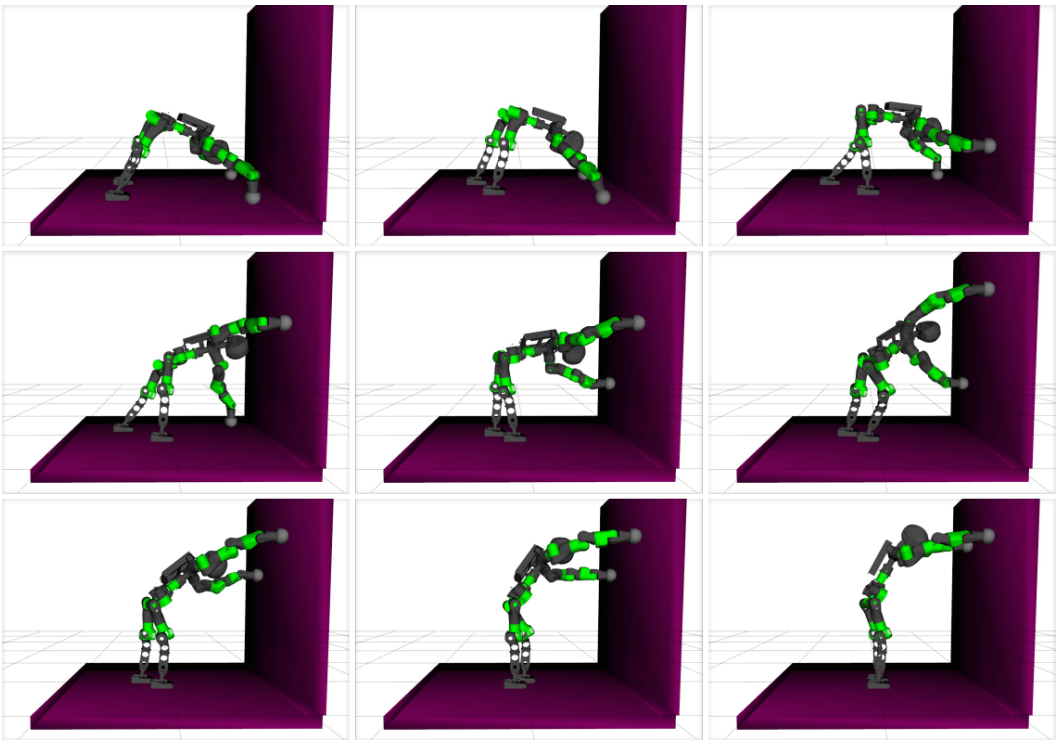

Figure 6.1. Stand up task: snapshots from a solution.

of multi-contact states contains on average 23 elements. The average number of needed iterations, number of vertexes in the final tree and time required to find a solution are, respectively, 537, 152 and 271 s. Fig. 6.1 shows some snapshots from a typical solution: the robot correctly exploits the environment as support to fulfil the assigned stand up task.

6.6 Conclusions

We presented a randomized strategy for planning humanoid multi-contact motions. Our strategy consists in two sequential planners. The first generates a sequence of multi-contact states leading to the desired final stance, while the second computes a configuration space trajectory realizing such sequence. We showed preliminary results regarding the application of the proposed planner to the stand up task of the COMAN+ humanoid.

For the first planner, i.e., the multi-contact state planner, we have proposed a novel approach that thanks to its randomized strategy does not require to predesign potential contacts between robot and environment points, to specify task-oriented heuristics, or precomputations (e.g., of stance-adjacency graphs or feasible configurations of the various kinematic chains). We believe that this is an advantage over existing approaches.

The results are promising and open up to multiple further developments. Our current work focuses on the two following aspects.

- Extension of the proposed planning strategy to the case (of more practical interest) of surface contacts, i.e., when a surface of a robot link is in contact with a surface of the environment (e.g., a foot on the ground). A surface contact fully constrain the pose (position and orientation) of the involved end-effector, and can be modelled using a set of at least three point contacts located on the vertexes of the surface. Accordingly, the CS solver described in Sect. 6.3.3 must consider, for each surface contact, the corresponding contact point set. Furthermore, each vertex in the tree must maintain, for each surface contact, the resultant contact wrench that can be readily computed from the forces (at the corresponding contact point set) computed by (6.5).

- Integration of the proposed planning strategy in a complete multi-contact planning and control architecture that will allow the experimental validation on the real humanoid. The complementary multi-contact controller, which is in charge of realizing the motions computed by the proposed multi-contact planner, must compute the torque commands for the COMAN+ actuators. It shall combine a contact forces distribution module ([111]) with an impedence controller. The first is in charge of realizing the reference contact forces (that are planned by our planner simultaneously to the configuration space trajectory and can be directly retrieved from the stances in the generated multi-contact state sequence) at best, while providing reactive balance capabilities. The second is responsible for tracking the reference joint positions and velocities resulting from the planned configuration space trajectory. Proprioceptive feedback must be used for both control objectives.

Other future work will address the improvement of the efficiency of the multi-contact state planner. A possibility that we want to explore is to implement a bi-directional version of the planner, where two trees rooted, respectively, at the initial and final stance are grown until they share a common stance. Another possibility consists in designing a unique IK-and-CS solver that can produce in a single stage a configuration satisfying requirements R1-R2.1-R3. Finally, incorporating performance criteria in the planning stage, as done in Chap. 4 for the footstep planning problem, is another interesting direction that we want to investigate.

Chapter 7

Safe human-humanoid coexistence

The most recent paradigms for adopting robotic technologies in applications emphasize the role of collaboration between robots and humans. Since collaboration implies sharing a common environment, safety concerns immediately become relevant. In industrial contexts, these have been addressed through the introduction of lightweight, compliant manipulators [112] and the development of new techniques for safe coexistence and interaction with humans [113]. Clearly, similar issues arise in service applications; for example, [114] give a survey of human-aware robot navigation and [115] provide a review of recent research on safety for domestic robots.

One of the most essential safety layers in a robot is related to the avoidance of obstacles, static or dynamic, which can be implemented using an appropriate motion planning strategy (see Chap. 2). Recently, researchers have started looking at this issue in the context of safe human-robot coexistence and interaction [116, 117, 118]. These methods, however, are almost invariably devoted to fixed-base manipulators or wheeled robots.

On the other hand, there exists a growing interest in the use of humanoids for assembly operations where access by fixed-base manipulators or wheeled robots is impossible. For example, a recent EU H2020 research project targeted the adoption of humanoids in aeronautic manufacturing [119]. Indeed, there is a widespread view that humanoids represent a rather natural choice of platform in environments that are specially designed to accommodate humans. Whether one agrees or not, there is no doubt that the challenging problem of safe deployment of humanoid robots needs be addressed.

The design of safety layers for humanoids must account for their unique characteristics: in particular, the fact that they can displace their base only through steps and that balance must be maintained at all times during motion [120]. One of the first works showing a humanoid (ASIMO) safely navigating an environment populated by dynamic obstacles via vision-based replanning was due to [53]. More recent results range from reactive Model Predictive Control techniques [55, 56] to full-fledged whole-body motion planners like that presented in Chap. 3.

Another important body of research related to the reliability of humanoids originated from the 2015 DARPA Robotics Challenge, in which research teams

competed to effectively control humanoids in environments designed to emulate a real-world disaster scenario [121, 122]. Many planning and control techniques developed in this context by the participating teams proved to be effective [123], providing strong inspiration for achieving robust operation of humanoids.

All the above works, to which many more could be added [124], focus however on a single issue which is considered relevant for reliable operation, such as robust locomotion or collision avoidance. Furthermore, humans are generally absent in the considered scenario (as in the case of the whole-body planning framework presented in Chap. 3), and tasks are often executed in supervised autonomy with the aid of specifically designed user interfaces (as it happens in the DARPA Challenge [125]). In the literature, there is no general study that looks at the safety problem from a global viewpoint in order to design a holistic framework for achieving safe operation of humanoids.

In this chapter we propose a complete framework for the safe deployment of humanoid robots in environments that may contain humans. Proceeding from some general guidelines, we propose several safety behaviors, classified in three categories:

- override behaviors, which stop the execution of the current task to account for the presence of an immediate danger, leading to a state from which normal operation can only be resumed after human intervention;
- temporary override behaviors, that take control of the robot for the limited amount of time necessary to address safety concerns, after which task execution can be automatically resumed;
- proactive behaviors, which are aimed at increasing the overall level of safety by an adaptation or enhancement of the robot activity, without interrupting the current task.

Activation and deactivation of these behaviors is triggered by information coming from the robot sensors and is handled by a state machine. To allow this, the state of the robot is identified by the current context (essentially, the task the robot is executing) and all active behaviors.

We also discuss the implementation of our safety framework in a reference control architecture, showing in particular that all behaviors related to locomotion can be efficiently realized in an MPC setting. Two humanoid platforms are used to show the performance of the proposed method, i.e., HRP-4 in simulation and NAO in experiments.

This chapter is organized as follows. Sect. 7.1 briefly reviews the existing safety standards for robots, while Sect 7.2 formulates some general guidelines for safe deployment of humanoids. In Sect. 7.3 we enunciate the robot sensing capabilities assumed by our framework. An overview of the proposed safety behaviors is given in Sect. 7.4, while Sect. 7.5 provides a detailed description of each behavior. Sect. 7.6 presents the state machine that orchestrates activation and deactivation of the behaviors. A reference control architecture is outlined in Sect. 7.7, and the implementation of the proposed behaviors within such architecture is discussed in Sect. 7.8. Simulation and experimental results are presented in Sects. 7.9 and 7.10, respectively. Sect. 7.11 presents additional results and discusses limitations and

possible extensions of the proposed method. Sect. 7.12 offers some concluding remarks.

7.1 Safety standards

Standards codify a set of practices that inform the design and operation of technologies. A product does not necessarily have to follow international standards as, unlike laws and regulations, these are not mandatory. However, they often provide a guarantee of compliance with regulations which otherwise can be quite hard to accommodate. It is therefore advantageous whenever possible to follow an existing standard, as this simplifies the design process.

The main international standards are published by either IEC or ISO committees. The first body focuses on standards related to electronics, while the second covers the remaining areas. Standards are divided in three categories.

- *Type A* standards provide basic rules and guidelines for machine safety (e.g., ISO 12100 “Basic Concepts, Design Principles" and ISO 14121 “Principles of Risk Assessment").
- *Type B* standards are further divided into two subtypes: B1 covers aspects such as safety distances or ergonomic principles (e.g., ISO 13857 “Safety Distances"). B2 describes rules concerning protective equipment for different applications (e.g., IEC 13850 “Emergency Stop").
- *Type C* standards refer to specific kinds of machines or areas of application, such as robots, and describe practical requirements and precautionary measures relating to all significant risks (e.g., see ISO 10218 “Robots and Robotic Devices - Safety Requirements for Industrial Robots"). Type C standards refer to Type A and B standards for generalities but may deviate from them when needed by the application.

The fundamental standard for industrial robot safety is ISO 10218, which highlights three particular aspects.

First, the standard suggests that means must be provided for the control and/or the release of hazardous energy stored in the robot. Examples of energy storage sources are batteries, springs and gravity.

The second aspect concerns safety stops. All robotic systems should have both a protective and an emergency stop function. Protective stops are used for risk reduction and can be activacted and deactivated automatically. Emergency stops are used in dangerous situations and require manual intervention.

Finally, ISO 10218 defines conditions for safe human-robot collaborative operation, identifying in particular four modes:

- *Safety-rated monitored stop.* The robot must stop whenever a human enters the shared workspace. This method does not allow collaborative work but only coexistence in the workspace. The robot may resume automatic operation when the human leaves.

- *Hand guiding.* The human can move the robot by physically interacting with it, e.g, to allow simplified path/point teaching. When provided, hand guiding equipment (such as a joystick) shall be located close to the end-effector. This mode must provide an emergency stop, an enabling device and safety-rated monitored speed limit.

- *Speed and separation monitoring.* The robot progressively reduces its speed as the human approaches. Failure to maintain the desired relative speed or separation distance will trigger a protective stop.

- *Power and force limiting.* In this mode, humans and robots can safely interact with little or no additional safety components because the robot force/power are bounded by design or control. Safe bounds are determined following ISO 10218-2 and ISO/TS 15066.

Almost all the above mentioned standards are specifically devised for industrial fixed-base manipulators, with only few exceptions addressing the case of wheeled mobile robots. No existing standard explicitly considers humanoid robots, whose peculiar nature must be properly taken into account.

7.2 Safety guidelines

With an eye to the standards discussed in the previous section, we provide here a qualitative description of the guidelines that inspired the design of our safety behaviors. In particular, we argue that the following recommendations should be followed for safe operation of a humanoid robot.

- *Watch what you're doing.* The robot should watch its main area of operation. When performing a manipulation task, it will therefore look at its hand(s) and/or at the object to be manipulated. When performing a locomotion task, it should direct its gaze towards the area where it is about to step.

- *Be on the lookout.* If the robot is idle, then it should scan the surroundings to identify possible sources of danger. In particular, if a moving object (e.g., a human) is detected, the robot should keep an eye on it.

- *Evade if you can.* When a moving object approaches, the humanoid robot should perform an evasive action, if this can be done safely.

- *Halt if you must.* In a situation of clear and present danger, the robot should terminate any activity and stop as soon as possible.

- *Beware of obstacles.* In the vicinity of unexpected objects, robot velocities and forces should be modified, scaled down or even zeroed in order to reduce potential damage in the case of a collision.

- *Look for support.* When locomotion is expected to be challenging (e.g., on stairs), the robot should try to establish additional contact with the environment (e.g., with a handrail). The possibility of improving balance by adding

contacts should also be considered whenever a non-negligible risk of falling is detected.

- *Protect yourself.* In the imminence of a potentially damaging event, such as an unavoidable fall, the robot should act so as to minimize damage to itself and/or the environment.

Some of these guidelines will directly result into one or more safety behaviors (Sects. 7.4 and 7.5) that are activated to improve the level of safety when necessary. Other guidelines must also be taken into account at the basic planning/control stage. For example, *watch what you're doing* generates a behavior aimed at increasing the level of safety (*scan*) but also calls for visual-servoed manipulation/locomotion strategies during normal operation; the *look for support* guideline is reflected in a safety behavior (*add_contact*) but has also consequences at the planning stage (e.g., generation of stair climbing motions must include handrail grasping and releasing).

7.3 Sensing assumptions

We now specify which information must be made available by the robot sensory system (or by external sensing devices) to implement the proposed safety framework. We shall not discuss in any detail the perception processes that provide such information.

Throughout this chapter, it is assumed that at any time instant the humanoid robot has a certain level of *awareness* about its own state and the surrounding environment, defined as follows:

- The robot knows whether there are *unexpected objects* (i.e., objects that are not present in the available map of the environment) within a perception area $\mathcal{P}$, whose extension depends on the specific sensory system, and in particular can measure the position (e.g., range and bearing) of the nearest point on each

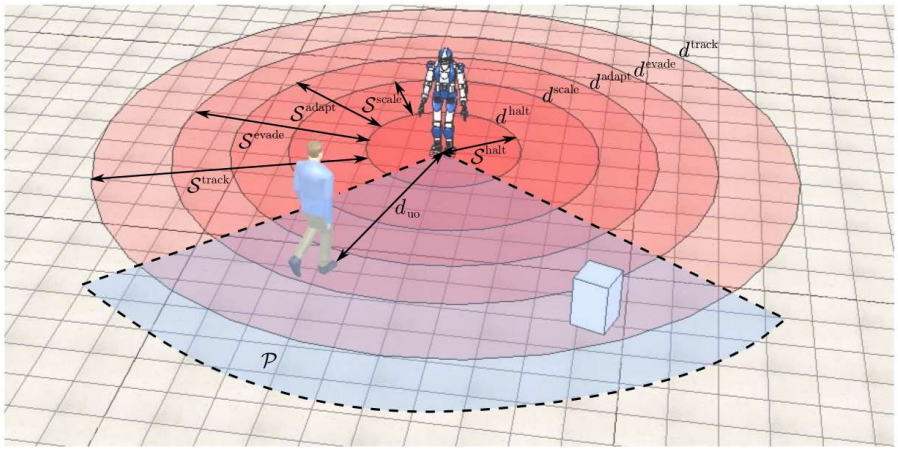

Figure 7.1. The humanoid robot can identify unexpected objects within a perception area $\mathcal{P}$ and measure the minimum distance d_{uo} to the closest of them. Also shown are the various safety areas (with the associated thresholds) defined in Sect. 7.5.2.

such object. As a consequence, it can compute the distance d_{uo} to the closest unexpected object in $\mathcal{P}$ (see Fig. 7.1). If there is no unexpected object in $\mathcal{P}$, d_{uo} is set to ∞. Techniques for identifying unexpected objects are typically based on a comparison between the predicted and the actual scene; for example, [126] consider change detection in images.

- The robot is able to establish whether the closest unexpected object in $\mathcal{P}$ is *moving* or *stationary* (e.g., a human walking vs. some misplaced furniture). In practice, this can be done by looking at significant variations of d_{uo} [127] once the effect of the robot's own motion has been removed. The f_{mo} flag is used to specify whether the unexpected object is moving or not.

- The robot can detect *unexpected contacts* with the environment, indicated by the f_{uc} flag. Depending on the contact detection method, other information may be available, such as the location of the contact point on the robot body or the interaction force [128].

- The robot knows the current risk of fall, represented by r_{fall}. For example, r_{fall} can be estimated from the position of the Zero Moment Point based on inertial measurements [129].

- The robot knows the location of *contact surfaces* that can be reached without stepping from its current posture. Contact surfaces are surfaces (or points) of the environment with which the robot may safely establish a contact for additional support [130]. The existence of reachable contact surfaces is encoded in the f_{cs} flag.

- The robot knows l_{battery}, its current battery level.

7.4 Overview of safety behaviors

This section provides a general description of the behaviors adopted by the humanoid robot to increase the level of safety for itself and the environment, which may include humans. In particular, three categories of safety behaviors are introduced, i.e., *override*, *temporary override* and *proactive*. We explain the idea behind each behavior and the situation in which it will be activated. A formal description, with detailed triggering conditions for each case, will be given in the next section.

7.4.1 Override behaviors

Override behaviors stop the execution of the current task and lead to a state from which normal operation can only be resumed after human intervention. They are intended as a way to react to unexpected and dangerous situations from which it would not be safe (or even possible) to resume the task automatically. We define two override behaviors:

- *halt*: In many situations, robot operation becomes critical: for example, when the battery level l_{battery} is too low, or when the distance d_{uo} to an unexpected moving obstacle goes below a certain threshold. In these cases, the *stop if*

you must guideline indicates that the robot should immediately abort any task. The *halt* behavior is an emergency stop procedure which interrupts any operation as quickly as possible. However, one should keep in mind that a humanoid robot, especially during locomotion, must stop in such a way to maintain balance.

- *self-protect*: While it is obviously desirable to avoid falling altogether, there are several reasons which might lead to a loss of balance, for example a hardware/software fault, or an unexpected collision. To properly handle this event, one can design a *self-protect* behavior to be activated during falls, as suggested by the *protect yourself* guideline. In practice, when the robot detects an unrecoverable loss of balance, it must immediately adopt measures aimed at minimizing damage to itself and the environment.

7.4.2 Temporary override behaviors

Some events which cause safety concerns require the robot to stop task execution for a limited amount of time. As soon as these concerns have been properly handled, task execution can resume automatically. In particular, this is the case of the following behaviors:

- *stop*: This behavior is activated when external circumstances suggest to stop walking as a precaution, in application of the *beware of obstacles* guideline. This is for example the case of an unexpected object moving in the vicinity of the robot; in this situation, locomotion is interrupted and only resumed if the object leaves the area. The *stop* behavior is also used for transitioning from normal operation to other safety behaviors (e.g., to *track*) and vice versa (e.g., from *evade*). Note that *stop* differs from *halt* because it is more graceful and does not require human intervention to restart the robot.

- *evade*: Following the *evade if you can* guideline, if an unexpected moving object tends to approach the robot, this performs an evasive maneuver to avoid collision. At the end of the maneuver, the robot can resume normal operation as soon as the object does not constitute a threat anymore.

- *add_contact*: This behavior, which descends from the *look for support* guideline, allows the robot to establish new contacts for additional support whenever it is standing and the risk of fall is estimated to be non-negligible. In fact, we consider intrinsically risky trying to establish a new contact while the robot is walking. Besides, if the robot is ascending or descending stairs, additional contact with a handrail should have already been taken into account at the planning stage.

- *track*: If an unexpected moving object has been detected in its vicinity, the robot keeps its gaze directed at it, as suggested by the *be on the lookout* guideline. Note that this behavior can only be activated when the robot is idle or performing an observation task (see Sect. 7.5.1): in any other case, averting the gaze from the current task can be dangerous (*watch what you're doing*

guideline). In particular, if the robot is walking, it will need to stop before starting to track the object.

7.4.3 Proactive behaviors

Proactive behaviors are actions intended to increase the overall safety level by calling for an adaptation or enhancement of the current robot activity. They include:

- *scan*: During manipulation or locomotion, the robot keeps its gaze directed at the main area of operation, as suggested by the *watch what you're doing* guideline. If the robot is idle, then it scans its surroundings, in application of the *be on the lookout* guideline.
- *adapt_footsteps*: During locomotion, the robot is in general controlled via high-level directives, such as tracking a reference velocity, or reaching a specific location in the workspace. The *adapt_footsteps* behavior, inspired to the *beware of obstacles* guideline, allows the robot to locally modify its footstep plan to avoid collision with stationary unexpected objects in its path.
- *scale_velocity-force*: If a nearby unexpected object is detected during manipulation, the robot must decrease all velocities and forces associated to the current manipulation task to reduce the risk of collision or the associated damages, as indicated by the *beware of obstacles* guideline.

7.5 Behavior-based safety framework

The activation and deactivation of the various safety behaviors is triggered by the information coming from the sensory system as described in Sect. 7.3 and depends on the current context (see Fig. 7.2).

In this section, we define the possible contexts and discuss the various safety areas used for behavior activation. Then, we describe each behavior in detail. Transitions to, from and among safety behaviors are actually controlled by a state machine, in which states are defined as a context followed by one or more active behaviors. The structure of the state machine will be discussed in Sect. 7.6.

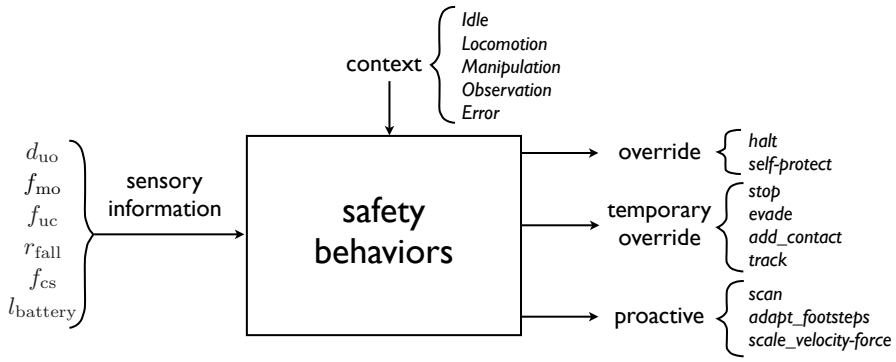

Figure 7.2. The activation of safety behaviors depends on the current context and is triggered by sensory information.

7.5.1 Contexts

The robot *contexts*[1] characterize what the robot is doing at a certain time instant. We identify five robot contexts:

- *Idle.* The humanoid is standing in double support at a fixed position and not performing any particular task.
- *Locomotion.* The humanoid is moving in the environment by taking steps. This includes walking, multi-contact locomotion and ascending/descending stairs.
- *Manipulation.* The humanoid is standing and executing a manipulation task that does not require any stepping.
- *Observation.* The humanoid is standing and executing a high-level observation task (e.g., find an object on a table). No locomotion or manipulation task is simultaneously being executed.
- *Error.* The robot is on hold until restarted by human intervention.

The first four contexts are associated to *normal operation*[2] (i.e., no safety behavior is active, except for *scan*) but also to operation under temporary override or proactive safety behaviors. *Error* is the only *emergency* context, to which the robot is released from override behaviors (*halt*, *self-protect*); from this context, normal operation cannot be resumed automatically.

7.5.2 Safety areas and thresholds

All safety behaviors are triggered by a certain measured quantity becoming larger or smaller than some given threshold. The most relevant of these quantities is the distance d_{uo} between the robot and the closest unexpected object, for which we specify five thresholds, i.e., in decreasing order: d^{track}, d^{evade}, d^{adapt}, d^{scale}, d^{halt}. As shown in Fig. 7.1, these thresholds on d_{uo} implicitly define five (partially overlapping) annular areas[3] around the robot:

- $\mathcal{S}^{\text{track}}$, defined by $d^{\text{halt}} \leq d_{\text{uo}} \leq d^{\text{track}}$. If the robot is not performing any locomotion or manipulation task (context *Idle* or *Observation*) and an unexpected moving object enters $\mathcal{S}^{\text{track}}$, the robot will start tracking it visually. If the robot is walking (context *Locomotion*), it will have to stop before starting to track the object (walking without watching the stepping area would be unsafe).

[1] The definition of contexts could also take into account the various environments: for example, *Locomotion on flat ground* could be different from *Locomotion on stairs*, e.g., in terms of acceptable risk of fall.

[2] Note that an *Idle* context can be included in a mission plan as an intentional pause, e.g., at the transition between different phases or for debugging purposes.

[3] The definition of areas around the robot is also suggested by *proxemics*, the study of the use of space in social interaction [131]. Its goal is to describe and characterize the distances between humans in different social contexts, and the way these are established and perceived. In robotics, proxemics can serve as a basis to model robot behavior in interaction with humans [132], which is especially relevant with humanoid robots as they are designed to allow for natural and comfortable interactions.

- $\mathcal{S}^{\text{evade}}$, defined by $d^{\text{halt}} \leq d_{\text{uo}} \leq d^{\text{evade}}$. If the robot is not performing any task (context *Idle*) and an unexpected moving object enters $\mathcal{S}^{\text{evade}}$, the robot will execute an evasion maneuver.

- $\mathcal{S}^{\text{adapt}}$, defined by $d^{\text{halt}} \leq d_{\text{uo}} \leq d^{\text{adapt}}$. If a walking robot (context *Locomotion*) detects an unexpected stationary object in $\mathcal{S}^{\text{adapt}}$, it will adapt the footstep plan to avoid collisions.

- $\mathcal{S}^{\text{scale}}$, defined by $d^{\text{halt}} \leq d_{\text{uo}} \leq d^{\text{scale}}$. If the robot is performing a manipulation task (context *Manipulation*) and an unexpected moving object is detected in $\mathcal{S}^{\text{scale}}$, the robot will reduce all velocities and forces associated to the current manipulation task.

- $\mathcal{S}^{\text{halt}}$, defined by $d_{\text{uo}} \leq d^{\text{halt}}$. This is the innermost safety area. If any unexpected object is detected in $\mathcal{S}^{\text{halt}}$, the robot will terminate all operations, irrespective of the current context.

As for the estimated risk of fall, as soon as r_{fall} becomes significant ($r^{\text{low}} \leq r_{\text{fall}} \leq r^{\text{high}}$), the robot will establish a new contact for additional support, provided that a suitable surface is available ($f_{\text{cs}} = \text{TRUE}$). If a fall is considered inevitable ($r_{\text{fall}} > r^{\text{high}}$), the robot will take measures aimed at minimizing damages.

Finally, operations are terminated also when the battery level l_{battery} is too low ($l_{\text{battery}} \leq l^{\text{low}}$).

Fig. 7.3 summarizes the activation of behaviors as a function of the values of d_{uo}, r_{fall} and l_{battery}.

The reader may have noticed that the safety behaviors related to collision avoidance (such as *stop*, *evade*, *halt*) discussed so far are only driven by the distance between the robot and the unexpected object, and do not take into account their relative velocity. This choice was made for the following reasons:

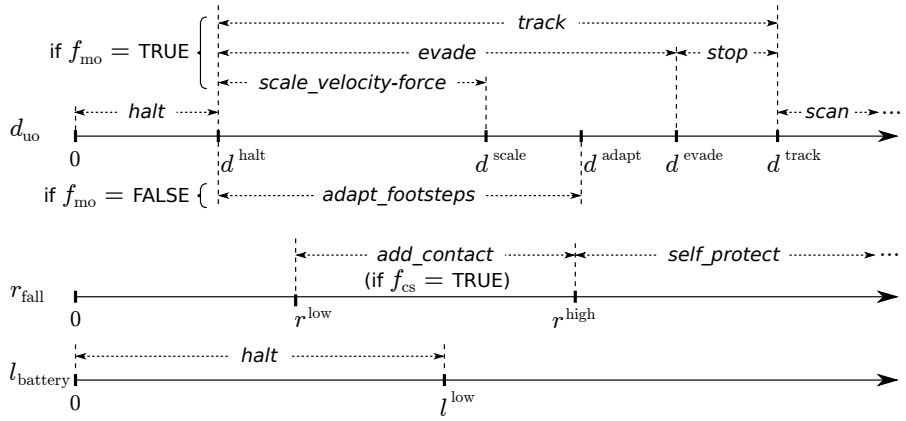

Figure 7.3. The different thresholds used for quantities d_{uo} (top), r_{fall} (center) and l_{battery} (bottom), each with its associated behaviors. Consider that the actual activation of a behavior depends on the current context (not shown here). The only situation which does not appear in this representation is the triggering of *halt* due to an unexpected contact ($f_{uc} = \text{TRUE}$).

- Velocity information may not be available, or in any case it may be computationally more costly to obtain. Indeed, in Sect. 7.3 we have assumed that the sensory system only provides distance measurements.
- If an unexpected obstacle enters a safety area (e.g., $\mathcal{S}^{\text{evade}}$), its relative velocity w.r.t. the robot is certainly directed towards the half-plane containing the robot (and tangent to the area). Therefore, our choice corresponds to a conservative viewpoint in which any such relative velocity is considered dangerous, independently from its specific orientation and magnitude.
- Working out an extension of the method for the case in which relative velocity is measured and used by the collision avoidance behaviors is relatively easy, see Sect. 7.11.3.

7.5.3 Definitions of behaviors

The formal definition of each safety behavior requires the specification of:

- one or more *contexts* from which the behavior can be activated;
- a *trigger*, indicating which particular event or piece of information will cause the behavior to activate, along with specific actions that occur upon triggering, such as deactivating other behaviors;
- the *action*, i.e., which activities are associated to the behavior;
- a *release*, which is an event or piece of information that causes the behavior to deactivate, again including specific actions that occur upon deactivation.

In the following, we define override, temporary override and proactive behaviors in this order.

halt

- **Context**: *Idle*, *Locomotion*, *Manipulation*, *Observation*.
- **Trigger**: $d_{\text{uo}} \leq d^{\text{halt}}$ (T1) OR $f_{\text{uc}} = \text{TRUE}$ (T2) OR $l_{\text{battery}} \leq l^{\text{low}}$ (T3).

  Triggering permanently deactivates any active behavior and inhibits all others from activating, except for *self-protect* in case of a fall.
- **Action**: Depends on the context and the trigger:
  - If the context is *Idle* and the trigger is T1 or T3, the robot will augment its support polygon and/or assume a low-impact configuration (e.g., by folding its arms).
  - If the context is *Idle* and the trigger is T2, the robot will decrease joint stiffness on the kinematic chain where the contact has occurred, provided that the latter is on the upper body. Otherwise, the robot will simply maintain its current posture.

  - If the context is *Locomotion*, *Manipulation* or *Observation* the robot will abort the task and stop any motion as soon as possible, regardless of the trigger.

- **Release**: When the action is completed.
  Upon release, context is changed to *Error*.

self-protect

- **Context**: *Idle*, *Locomotion*, *Manipulation*, *Observation*.
- **Trigger**: $r_{\text{fall}} > r^{\text{high}}$.
  Triggering permanently deactivates any active behavior and inhibits all others from activating.
- **Action**: The robot will abort the task and act so as to minimize the potential damage to itself and/or the environment. To this end, several aspects must be considered, including *(i)* how to fall, i.e., which internal posture to assume before impact to preserve robot integrity *(ii)* where to fall, i.e., how to choose the landing surfaces so as to avoid fragile objects.
- **Release**: When the action is completed.
  Upon release, context is changed to *Error*.

stop

- **Context**: *Locomotion*.
- **Trigger**: $d^{\text{evade}} < d_{\text{uo}} \leq d^{\text{track}}$ AND $f_{mo} = \text{TRUE}$.
- **Action**: The robot will stop walking, ending in a double support configuration. This is done before starting to track (and possibly later evade) an unexpected moving object; or at the end of an evasion maneuver.
- **Release**: When the action is completed.
  Upon release, context is changed to *Idle*.

evade

- **Context**: *Idle*
- **Trigger**: $d^{\text{halt}} < d_{\text{uo}} \leq d^{\text{evade}}$ AND $f_{\text{mo}} = \text{TRUE}$.
  Triggering changes context to *Locomotion*.
- **Action**: The robot will execute a reactive evasion maneuver so as to increase the distance to the unexpected moving object.
- **Release**: When $d_{\text{uo}} > d^{\text{evade}}$.
  Upon release, *stop* is activated in order to interrupt the evasion maneuver.

add_contact

- **Context**: *Idle*, *Manipulation*, *Observation*.
- **Trigger**: $r^{\text{low}} \leq r_{\text{fall}} \leq r^{\text{high}}$ AND $f_{\text{cs}} = \text{TRUE}$.
  Upon triggering, context is changed to *Idle* if it was *Manipulation* or *Observation*. Moreover, *track* is deactivated if active, and *evade* is inhibited from activation.
- **Action**: The robot will interrupt the task (if the context was *Manipulation* or *Observation*), and select and establish contact with an additional support point on the available surfaces.
- **Release**: When the action is completed.

track

- **Context**: *Idle*, *Observation*.
- **Trigger**: $d^{\text{halt}} < d_{\text{uo}} \leq d^{\text{track}}$ AND $f_{\text{mo}} = \text{TRUE}$.
  Triggering deactivates *scan* and changes the context to *Idle* if it was *Observation*.
- **Action**: The robot will interrupt any observation task (if the context was *Observation*) and direct its gaze at the closest unexpected object moving in $\mathcal{S}^{\text{track}}$.
- **Release**: When $d_{\text{uo}} > d^{\text{track}}$.
  Upon release, *scan* is activated.

scan

- **Context**: *Idle*, *Manipulation*, *Locomotion*.
- **Trigger**: Active by default unless *track* is active.
- **Action**: Depends on the context:
  - If the context is *Idle*, the robot will scan the surrounding environment.
  - If the context is *Locomotion*, the robot will scan the path ahead.
  - If the context is *Manipulation*, the robot will scan the area of operation.
- **Release**: Never.

adapt_footsteps

- **Context**: *Locomotion*.
- **Trigger**: $d^{\text{halt}} < d_{\text{uo}} \leq d^{\text{adapt}}$ AND $f_{\text{mo}} = \text{FALSE}$.

- **Action**: The robot will modify the current footstep plan as needed to avoid the closest unexpected object standing in the scene.
- **Release**: When $d_{\text{uo}} > d^{\text{adapt}}$.

scale_velocity-force

- **Context**: *Manipulation*.
- **Trigger**: $d^{\text{halt}} < d_{\text{uo}} \leq d^{\text{scale}}$ AND $f_{\text{mo}} = \text{TRUE}$.
- **Action**: Robot velocities and/or forces associated to the current task will be reduced.
- **Release**: When $d_{\text{uo}} > d^{\text{scale}}$.

Note the following points.

- Override behaviors force the robot to abort any task and deactivate/inhibit all other kinds of behavior.
- Under temporary override behaviors, any task is interrupted for a limited period of time.
- Among temporary override behaviors, *add_contact* deactivates other behaviors that would cause a conflict of context (*evade*) or steal an essential sensory resource (*track*); for the same latter reason, *track* deactivates *scan*, and *scan* cannot be activated when the context is *Observation*.
- When temporary override behaviors are released, the context will always be *Idle* with the *scan* behavior active. It is important to note that *in this condition control goes back to normal operation*: the robot is ready to resume any task that was interrupted, under prompting by the supervisory module that activates and sequences tasks.
- Finally, proactive behaviors do not interrupt active tasks.

7.6 State machine

In the proposed framework, activation and deactivation of safety behaviors are handled by a state machine. In particular, the *state* of the robot is uniquely identified by the current context and all active behaviors, and denoted as Context/*behavior_1*/*behavior_2*/..., with behaviors listed in the order of activation.

The complete list of the states is the following:

- Idle/*scan*
- Idle/*track*

- Idle/*scan*/*add_contact*
- Idle/*halt*
- Idle/*self-protect*
- Locomotion/*scan*
- Locomotion/*scan*/*adapt_footsteps*
- Locomotion/*scan*/*stop*
- Locomotion/*track*/*evade*
- Locomotion/*track*/*evade*/*adapt_footsteps*
- Locomotion/*track*/*stop*
- Locomotion/*halt*
- Locomotion/*self-protect*
- Manipulation/*scan*
- Manipulation/*scan*/*scale_velocity-force*
- Manipulation/*halt*
- Manipulation/*self-protect*
- Observation/
- Observation/*halt*
- Observation/*self-protect*
- Error/

Fig. 7.4 gives a complete representation of the state machine. Transition from one state to another is instantaneous and corresponds to activation or deactivation of some behavior. For example, the transition from Locomotion/*scan* to Locomotion/*scan*/*adapt_footsteps* corresponds to the activation of the proactive behavior *adapt_footsteps*: the robot was walking and scanning the stepping area when an unexpected stationary object was detected in $\mathcal{S}^{\text{adapt}}$, so that it became necessary to adapt the footstep plan. Similarly, the transition from Locomotion/*track*/*evade* to Locomotion/*track*/*stop* corresponds to the deactivation of the temporary override behavior *evade*, which automatically triggers *stop*: the robot was performing an evasion maneuver which became unnecessary when the moving unexpected obstacle left $\mathcal{S}^{\text{evade}}$, so that motion was stopped. Note that *track* is still active in the final state after the transition because the object will still be in $\mathcal{S}^{\text{track}}$ after leaving $\mathcal{S}^{\text{evade}}$ (see Fig. 7.1).

The only *normal operation* states are Idle/*scan*, Locomotion/*scan*, Manipulation/*scan* and Observation/, in which no safety behavior is active apart from the

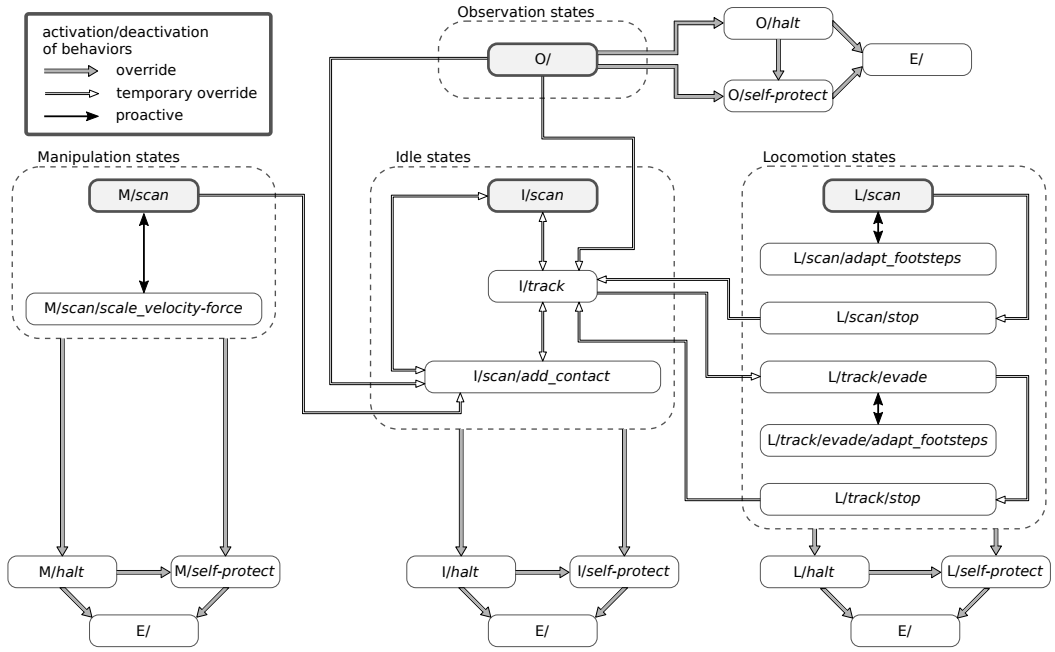

Figure 7.4. A representation of the state machine governing the transitions among the proposed safety behaviors. For compactness, contexts within states are denoted only by their initial (I, M, L, O or E). A transition originating from the boundary of a dashed box (e.g., the box of Idle states) indicates that the transition may originate from any state in the box. Normal operation states (i.e., states where no safety behavior is active apart from the default behavior *scan*) are shown in gray.

default proactive behavior *scan*. In these states, control is entirely committed to the realization of the desired task.

Finally, examination of Fig. 7.4 confirms that:

- override behaviors lead to the Error/ state;
- temporary override behaviors ultimately lead to the Idle/*scan* state, that is a normal operation state from which the original task can be resumed;
- proactive behaviors (apart from *scan*) simply disappear in the end, leading back to the state from which they were activated.

7.7 Control architecture

The implementation of the safety behaviors, which will be discussed in some detail in the next section, will obviously depend on the specific control architecture of the robot. In view of this, we describe here a possible structure which will be used in the rest of the chapter for illustration.

Fig. 7.5 shows a general overview[4] of the control architecture. Note that safety

[4]The control architecture of humanoids invariably includes a stabilization module, not present in Fig. 7.5, whose role is to guarantee balancing of the robot in all situations. This low-level module is not discussed since here it bears no relevance to our safety framework.

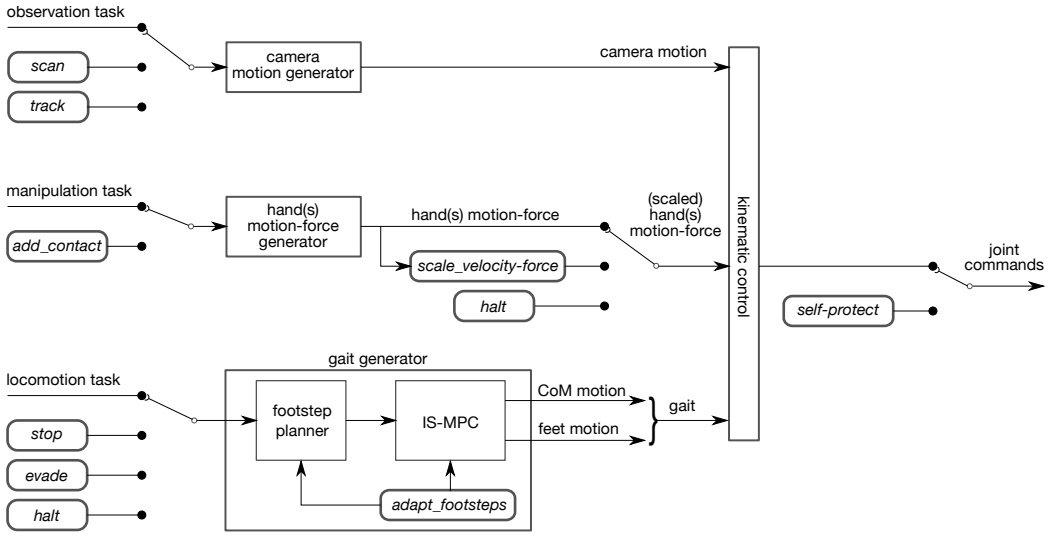

Figure 7.5. The considered control architecture. The safety behaviors blocks appear as either signal generators or modifiers.

behaviors blocks appear in this scheme as either reference signal generators (*scan*, *track*, *add_contact*, *stop*, *evade*, *halt*, *self-protect*) or signal modifiers (*scale_velocity-force*, *adapt_footsteps*). Note the particular placement of the *self-protect* behavior, which will take control of the robot directly at the joint level. Activation and deactivation of the behaviors is handled in the background by the state machine described in the previous section.

In the rest of this section, we describe the core components of the control architecture without reference to the safety behaviors. The specific way in which each behavior is embedded in the architecture will be discussed in the next section.

7.7.1 Camera motion generator

The camera motion generator is primarily (i.e., during normal operation) in charge of producing a suitable reference motion for the camera when an observation task is being executed (context *Observation*). An image-based visual servoing scheme [133] is used to move the camera so as to track a reference signal representing the motion of a desired feature in the image plane, e.g., for finding an object in the scene.

7.7.2 Hand(s) motion-force generator

During normal operation, the hand(s) motion-force generator is in charge of producing a suitable motion-force reference for the hand(s) when a manipulation task is being executed (context *Manipulation*). For example, when a certain grasp must be executed, the module will plan a continuous motion of the hand(s) so as to reach the associated Cartesian posture.

7.7.3 Gait generator

The primary role of the gait generator is to produce suitable reference motions for the robot Center of Mass (CoM) in order to execute a locomotion task (context *Locomotion*). Since such a module is specific to humanoid robots, we will describe it in some detail, considering for compactness the case of flat ground. The scheme at the core of the module can however be easily extended to work on non-flat terrains, in particular to deal with the presence of stairs, for example by adopting a complete online version of the approach presented in Chap. 4.

The gait generator that we consider is based on the work presented in [68]. In particular, it makes use of two sequential modules running in real time (Fig. 7.5). The first module, i.e., the footstep planner, generates a sequence of timed footsteps to realize high-level omnidirectional velocity commands v_x, v_y, ω. The second module, i.e., Intrinsically Stable MPC (IS-MPC), produces a trajectory of the robot CoM and feet such that (1) the Zero Moment Point (ZMP) is always within the support polygon, thus guaranteeing that balance is maintained, and (2) that the CoM trajectory is bounded with respect to the ZMP trajectory, implying internal stability. Both modules require the solution of QP problems.

The high-level velocity commands v_x, v_y, ω that drive gait generation are known over a *preview horizon* $T_p = P \cdot \delta$ of P sampling intervals — each of duration δ — in the future. The footstep generation module plans footsteps over the same horizon, whereas IS-MPC plans CoM and ZMP trajectories over a *control horizon* $T_c = C \cdot \delta$. It is assumed that $P \geq C$, so that IS-MPC can take full advantage of the preview information.

At each sampling instant t_k, the footstep planner receives in input the high-level reference velocities over the preview horizon, i.e., from t_k to $t_k + T_p = t_{k+P}$.

First, the footstep timing over T_p is determined[5] in the form $\mathcal{T}_s^k = \{T_s^1, \ldots, T_s^F\}$, where T_s^j is the step duration between the $(j-1)$-th and the j-th footstep, taken from the start of the single support phase to the next. Since the step duration is variable, the number F of footsteps falling within T_p may change with t_k.

Then, a sequence of F footsteps $(\hat{X}_f^k, \hat{Y}_f^k, \Theta_f^k)$ over the same interval is generated:

$$\begin{aligned} \hat{X}_f^k &= (\hat{x}_f^1 \ \ldots \ \hat{x}_f^F)^T \\ \hat{Y}_f^k &= (\hat{y}_f^1 \ \ldots \ \hat{y}_f^F)^T \\ \Theta_f^k &= (\theta_f^1 \ \ldots \ \theta_f^F)^T, \end{aligned}$$

where $(\hat{x}_f^j, \hat{y}_f^j, \theta_f^j)$ is the pose[6] (position + orientation) of the j-th footstep. To this end, we inject the high-level reference velocities into the following omnidirectional motion model

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix} \tag{7.1}$$

[5] Step timing is chosen by a simple heuristic based on the velocity commands [68].

[6] The hat on the position coordinates indicates that these are candidates values which will be later adjusted by IS-MPC. Orientations are final because their inclusion in the MPC formulation would make the problem nonlinear.

and then distribute the footsteps around the resulting trajectory in accordance with the timing $\mathcal{T}_s^k$. This is done by solving a sequence of two QP problems; the first computes the footstep orientations

$$\begin{aligned} &\min_{\Theta_f^k} \sum_{j=1}^{F} (\theta_f^j - \theta_f^{j-1} - \int_{t_s^{j-1}}^{t_s^j} \omega(\tau) d\tau)^2 \\ &\text{subject to} \quad |\theta_f^j - \theta_f^{j-1}| \leq \theta_{\max}, \end{aligned} \tag{7.2}$$

while the second computes the footstep positions

$$\begin{aligned} &\min_{\hat{X}_f^k, \hat{Y}_f^k} \sum_{j=1}^{F} (\hat{x}_f^j - \hat{x}_f^{j-1} - \Delta x^j)^2 + (\hat{y}_f^j - \hat{y}_f^{j-1} - \Delta y^j)^2 \\ &\text{subject to kinematic constraints.} \end{aligned} \tag{7.3}$$

In (7.2), $\theta_{\max}$ is the maximum allowed rotation between two consecutive footsteps, while t_s^j is the timestamp of the j-th footstep. In (7.3), $(\hat{x}_f^0, \hat{y}_f^0)$ is the known position of the support foot at t_k and Δx^j, Δy^j are given by

$$\begin{pmatrix} \Delta x^j \\ \Delta y^j \end{pmatrix} = \int_{t_s^{j-1}}^{t_s^j} R_\theta \begin{pmatrix} v_x(\tau) \\ v_y(\tau) \end{pmatrix} d\tau \pm R_j \begin{pmatrix} 0 \\ \ell/2 \end{pmatrix},$$

where R_θ, R_j are the rotation matrices associated respectively to $\theta(\tau)$ and θ_j^f, ℓ is the chosen coronal distance between consecutive footsteps, and the sign of the second term alternates for left/right footsteps. The *kinematic constraints* in the second problem are built to guarantee that the footsteps are kinematically feasible for the specific humanoid being considered.

Once the timed footstep plan is available, the IS-MPC module is called to generate the CoM motion and simultaneously adjust the footstep positions. To this purpose, we employ the version with *automatic footstep placement* of IS-MPC described in Appendix A.1. To ensure that the CoM trajectory does not diverge w.r.t. the ZMP, IS-MPC incorporates an explicit stability constraint which involves the future evolution, referred to as the *tail*, of the ZMP after the control horizon, which are unknown and must be conjectured to obtain a causal version of the constraint. In particular, one may set the tail to zero (*truncated tail*) if the high-level velocity commands indicate that the robot should immediately stop, or surmise the value of such velocities on the basis of the preview information encoded in the footstep plan (*anticipative tail*).

Overall, the described gait generator provides a reactive scheme that is suitable not only for regular gaits but also for executing sudden avoidance maneuvers, as well as temporary or emergency stops – all actions which are required in our safety framework.

7.8 Implementation of behaviors

In this section we discuss the implementation of safety behaviors inside the control architecture of Fig. 7.5. The main focus will be on those behaviors that are related to locomotion, which are specific to humanoids. For the others we shall simply provide pointers to existing work.

7.8.1 *halt*

The *halt* behavior realizes an emergency stop in the presence of immediate threats. As explained in Sect. 7.5.3, the behavior is declined differently depending on the context and/or trigger. Here, we consider the case in which the context is *Locomotion*.

Activation of *halt* is realized via two mechanisms:

- the high-level velocity commands v_x, v_y, ω are immediately set to zero;
- as a consequence, the truncated tail is used in the stability constraint of IS-MPC.

7.8.2 *self-protect*

As shown by Fig. 7.5, the *self-protect* behavior takes command of the robot at the joint level and therefore overrides all the preceding control architecture. When an impending fall is detected, this behavior acts so as to minimize the potential damage to itself and/or the environment. In the following, we refer the reader to some relevant works in this direction.

Choosing how to fall means assuming a proper configuration for reducing the effect of the impact with the floor. [134, 135] present a controller that limits the impact force, based on the idea of lowering the CoM as soon as possible by crouching or knee-bending; [136] use a similar approach to manage forward or backward falls. [137] compute whole-body trajectories aimed at minimizing damage due to falling through an optimization-based control strategy.

In addition to lowering the impact force, strategies for absorbing the impact are also useful. A control method that combines robot reconfiguration and post-impact compliance is proposed by [138]: during the falling phase, the robot is kept away from fall singularities, i.e., postures in which impact forces would be poorly absorbed. After the impact, compliance control is activated, with the motors behaving as spring-dampers.

Besides how to fall, it is also important to choose where to fall. [139] introduce a controller which changes the fall direction in order to avoid specific objects or parts of the environment. This method was extended to multiple objects by [140].

7.8.3 *stop*

The *stop* behavior temporarily interrupts locomotion in preparation for tracking a moving object, or at the end of an evasive maneuver.

Activation of *stop* is realized via two mechanisms:

- the high-level velocity commands v_x, v_y, ω go from their current value to zero over a fixed *arrest time*;
- as a consequence, the anticipative tail is used in the stability constraint of IS-MPC.

Once the motion has been stopped, the state becomes Idle/*track* (see Fig. 7.4). If the unexpected moving object leaves $\mathcal{S}^{\text{track}}$, the robot goes to the normal operation state Idle/*scan*, where the original locomotion task can be resumed.

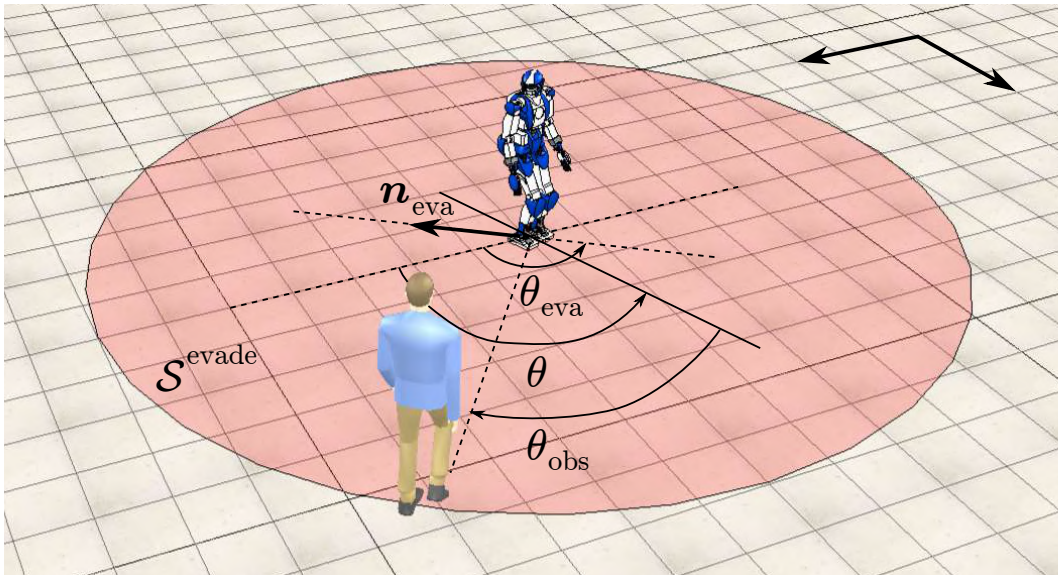

Figure 7.6. The geometry of evasion. A moving object enters $\mathcal{S}^{\text{evade}}$. The chosen direction of evasion is $\boldsymbol{n}_{\text{eva}}$.

The different effect of *stop* vs. *halt* will be illustrated via simulation in the next section.

7.8.4 *evade*

The *evade* behavior is realized by sending to the robot high-level velocity commands for avoiding an unexpected moving object that has entered the $\mathcal{S}^{\text{evade}}$ area.

To devise such commands we use the technique presented in [57]. Consider the geometry of the problem as shown in Fig. 7.6. The angle θ_{obs} under which the robot sees the moving object is directly measured by the robot (see the first assumption in Sect. 7.3). The chosen direction of evasion is represented by $\boldsymbol{n}_{\text{eva}}$, with the robot moving backwards so as to keep the object in its field of view.

While the humanoid can in principle move in any direction with motion model (7.1), studies on humans [141] indicate that time-efficient locomotion requires the orientation of the body to be tangent to the path, similarly to what happens in nonholonomic mobile robots. For this reason, we adopt the unicycle as template model for evasive maneuvers:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega, \end{aligned} \tag{7.4}$$

where x, y, θ denote now the position and orientation of the unicycle, and v, ω are its driving and steering velocity inputs. The latter are chosen as

$$v = -\bar{v} \tag{7.5}$$

$$\omega = k\,(\theta_{\text{eva}} - \theta), \tag{7.6}$$

While the driving velocity is set to a constant negative value, the steering velocity forces the unicycle (7.4) to align smoothly with the desired orientation, chosen as[7] $\theta_{\text{eva}} = \theta + \theta_{\text{obs}} - \text{sign}(\theta_{\text{obs}}) \cdot \pi/2$. As a result, we obtain $\omega = k\,(\theta_{\text{obs}} - \text{sign}(\theta_{\text{obs}}) \cdot \pi/2)$, which can be implemented using only on-board measurements. Alternatively, the proportional control law (7.6) may be replaced with

$$\omega = k\,\text{sign}(\theta_{\text{eva}} - \theta), \tag{7.7}$$

to make the evader perform the evasion maneuver with a constant curvature radius.

The final step is to send the control inputs (7.5–7.6) to the gait generator, with the adjustment $v_x = v\cos\theta$, $v_y = v\sin\theta$ (while ω is unchanged).

An observation is in order about obstacle avoidance during evasion maneuvers. Since evasion is a safety behavior, it leads the humanoid to an area which was not contemplated in the original motion plan. Obstacles in this area should therefore be considered as *unexpected*, regardless of their being represented or not in the environment map. With this rationale, *any* obstacle inside $\mathcal{S}^{\text{adapt}}$ will trigger the *adapt_footsteps* behavior during an evasion maneuver (see the first simulation in the next section).

7.8.5 *add_contact*

The *add_contact* behavior can be activated in *Idle*, *Observation* and *Manipulation*. The current task is interrupted and context is immediately changed to *Idle* to allow the robot to establish an additional contact. This requires first choosing a posture where one robot body (typically, a hand) is in contact with a reachable contact surface, whose existence is indicated by the f_{cs} flag. The hands motion-force generator module is then invoked to plan a continuous motion-force reference that will achieve the chosen desired posture.

When one or more contact surfaces are reachable, it is necessary to decide which contact to choose. Clearly, it is essential that the added contact improves balance. To this end, one may use concepts such as the generalization of ZMP support areas to the case of multiple non-coplanar contacts [143, 130]. These aspects have also been studied in the more general field of multi-contact planning for locomotion and manipulation [144, 145, 146]; an interesting summary is given by [48].

We emphasize that additional contacts may also be exploited outside the safety framework, i.e., during normal operation. For example, if the robot is required to climb a staircase it is sensible to take advantage of the handrail if available. In this case, however, the appropriate contacts must be integrated in the locomotion/manipulation task being executed.

7.8.6 *scan* and *track*

Both the *scan* and *track* behaviors are realized by invoking the camera motion generation module.

The *scan* behavior uses an artificial image feature as a reference signal. If the context is *Idle*, the artificial feature moves cyclically throughout the surrounding

[7] This evasion strategy is called *move aside*, as the humanoid moves (backwards) in a direction that is orthogonal to the object line of approach; other strategies are possible [142].

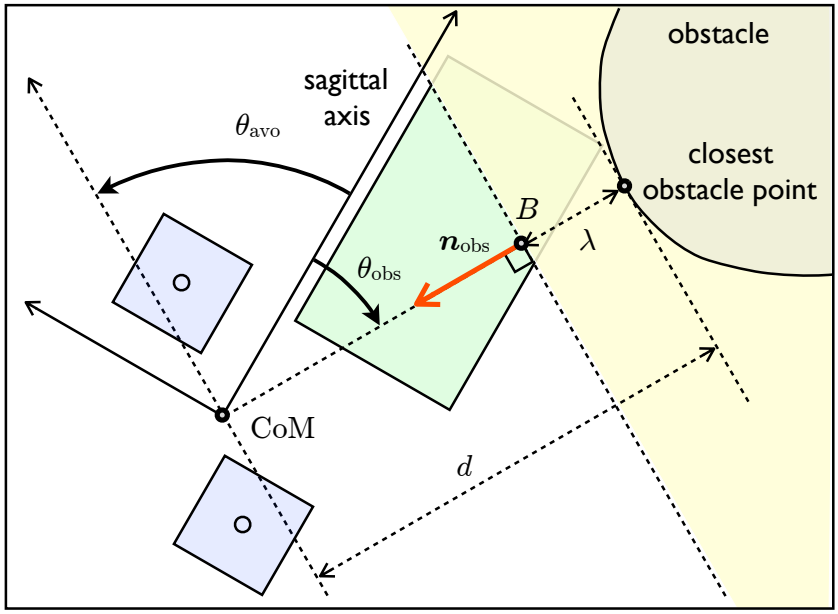

Figure 7.7. The geometry of footstep adaptation with the definition of the relevant quantities. The current robot placement is defined by its CoM. Also shown are the current footstep locations (light blue), the kinematically feasible zone (green) and the forbidden zone (yellow) due to the presence of the obstacle for the next footstep.

area so as to achieve complete coverage. In *Locomotion* or *Manipulation*, the feature is fixed at the center of the specific area of interest.

In the *track* behavior, the image feature will be directly the closest point on the unexpected moving object that has triggered the behavior.

7.8.7 *adapt_footsteps*

If the context is *Locomotion* and a stationary unexpected object is detected in $\mathcal{S}^{\text{adapt}}$, the *adapt_footsteps* behavior is invoked. This is realized, using an adaptation of the technique we presented in [147], by minor modifications of the two modules that constitute the gait generator, i.e., the footstep planner and IS-MPC.

Refer to Fig. 7.7 for the geometry of the problem and the definition of the relevant quantities. In particular, the range d and bearing θ_{obs} are directly measured by the robot (again, see the first assumption in Sect. 7.3), while θ_{avo} is defined as

$$\theta_{\text{avo}} = \theta_{\text{obs}} - \text{sign}(\theta_{\text{obs}}) \cdot \pi/2.$$

Within the footstep planner, the first modification is in the QP problem used to compute the footstep orientations from ω (see (7.2)), whose cost function is replaced by

$$\sum_{j=1}^{F} (\theta_f^j - \theta_f^{j-1} - \int_{t_s^{j-1}}^{t_s^j} \omega(\tau) d\tau)^2 + k_{\text{obs}} \frac{w(\theta_{\text{obs}})}{d^2} \left(\theta_f^j - \theta_{\text{avo}}\right)^2.$$

With respect to the original cost function, this contains an additional term that induces an alignment of the foosteps with θ_{avo}, i.e., with the tangent half-line

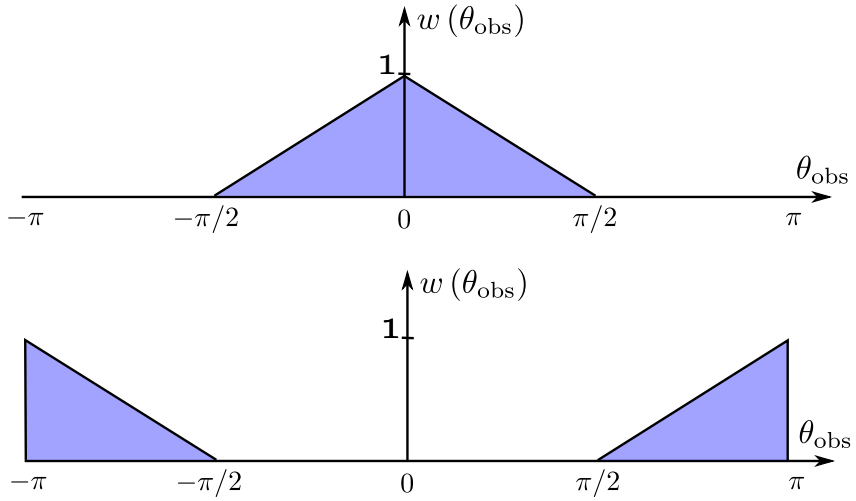

Figure 7.8. The weight function $w(\theta_{\text{obs}})$. Top: if the robot is walking forward. Bottom: if the robot is walking backwards.

originating at the closest object point (see Fig. 7.7). This second term is modulated through a scaling factor k_{obs} by the inverse of the squared distance and by the weight function $w(\theta_{\text{obs}})$ defined in Fig. 7.8. The idea here is that when the robot moves forward only obstacles lying in its front half-plane should be considered; whereas when moving backwards (e.g., during an evasion maneuver) footstep adaptation is only required to avoid obstacles behind the robot.

The second modification in the footstep planner is in the QP problem used to compute the footstep positions (see (7.3)), to which a collision avoidance constraint is added. With reference again to Fig. 7.7, consider the point $B = (x_B, y_B)$ located along the line connecting the CoM with the closest object point, at a safety distance λ from the latter, and draw the normal to the same line through B. The half-plane beyond this line (in yellow in Fig. 7.7) is a forbidden zone[8] for the footstep locations. This constraint is easily written as

$$\mathbf{n}_{\text{obs}}^{\mathbf{T}} \left\{ \begin{pmatrix} x_{\text{f}}^{j} \\ y_{\text{f}}^{j} \end{pmatrix} - \begin{pmatrix} x_B \\ y_B \end{pmatrix} \right\} \geq \mathbf{0} \tag{7.8}$$

with $\mathbf{n}_{\text{obs}}$ the unit vector defined in Fig. 7.7.

The same collision avoidance constraint must obviously be enforced also in the IS-MPC module which will determine the final footstep positions. Thus, constraint (7.8) is additionally included in the QP problem of IS-MPC with automatic footstep placement described in Appendix A.1.

Wrapping up, we may say that *adapt_footsteps* takes into account the presence of unexpected stationary objects in the robot path at two levels: in the cost function of the footstep orientation QP, and through the introduction of a collision avoidance constraint in the footstep position QP as well as in IS-MPC. As shown in the next

[8]Turning the obstacle into a half-plane is necessary to convexify the obstacle avoidance constraint. This might appear to be a conservative choice, but it should be noted that the position of the half-plane is updated at each iteration, so the robot will eventually be able to go around the obstacle.

two sections, both via simulations and experiments, this strategy is effective for collision-free locomotion.

7.8.8 *scale_velocity-force*

If a moving obstacle enters $\mathcal{S}^{\text{scale}}$ while the robot is in the *Manipulation* context, the *scale_velocity-force* behavior is activated. Both the hand velocity and the interaction forces are reduced for enhancing the level of safety. Studies are available in which velocity/force bounds are derived taking into account the dynamic properties of the robot as well as the possibility of human injury [148].

7.9 Simulations

In this section we provide simulated demonstrations of the proposed safety framework. The used robot is HRP-4. The robot has been equipped with a depth camera for gathering range and bearing information about the obstacles. All simulations are performed in the V-REP environment, enabling dynamic simulation via the Newton Dynamics engine.

The first simulation is designed so as to bring up several safety concerns in sequence, with the objective of illustrating how the state machine orchestrates transitions between behaviors. Snapshots of the simulation are shown in Fig. 7.9. Only three of the safety areas defined in Sect. 7.5.2 are shown around the robot, i.e., $\mathcal{S}^{\text{track}}$, $\mathcal{S}^{\text{evade}}$ and $\mathcal{S}^{\text{adapt}}$, respectively with $d^{\text{track}} = 5$ m, $d^{\text{evade}} = 3$ m, and $d^{\text{adapt}} = 1.5$ m. The remaining areas $\mathcal{S}^{\text{scale}}$ and $\mathcal{S}^{\text{halt}}$ are not shown because not relevant.

At the beginning, the robot is standing, not performing any task, and scanning the environment (state Idle/*scan*). At $t = 9$ s, a human enters $\mathcal{S}^{\text{track}}$. This event triggers the *track* behavior, and the robot starts following the human with its camera (state Idle/*track*). At $t = 25$ s, the human enters $\mathcal{S}^{\text{evade}}$, triggering the *evade* behavior: the robot initiates an evasion maneuver while still tracking the human (state Locomotion/*track*/*evade*). At $t = 29$ s, while the robot is still performing the evasion maneuver, a stationary object (the yellow cylinder) enters $\mathcal{S}^{\text{adapt}}$; the *adapt_footsteps* behavior is activated and the footstep plan is modified to avoid collision (state Locomotion/*track*/*evade*/*adapt_footsteps*). When the human leaves $\mathcal{S}^{\text{evade}}$, at $t = 37$ s, the *stop* behavior is invoked to interrupt the evasion maneuver (state Locomotion/*track*/*stop*); motion is terminated at $t = 40$ s (state Idle/*track*). The robot quits tracking the human when he leaves $\mathcal{S}^{\text{track}}$ at $t = 44$ s (state Idle/*scan*).

At $t = 49$ s the second part of the simulation begins. The robot is commanded to reach a goal in the workspace (bullseye mark). Accordingly, the context switches to *Locomotion* and appropriate high-level reference velocities are sent to the gait generator (state Locomotion/*scan*). Since the yellow object is still in $\mathcal{S}^{\text{adapt}}$, the *adapt_footsteps* behavior is immediately activated (state Locomotion/*scan*/*adapt_footsteps*). Once the object goes outside $\mathcal{S}^{\text{adapt}}$, at $t = 62$ s, the robot can walk directly towards the goal (state Locomotion/*scan*). At $t = 83$ s, another stationary object (the green cuboid) enters $\mathcal{S}^{\text{adapt}}$, and again collision is avoided by footstep adaptation (state Locomotion/*scan*/*adapt_footsteps*). As soon

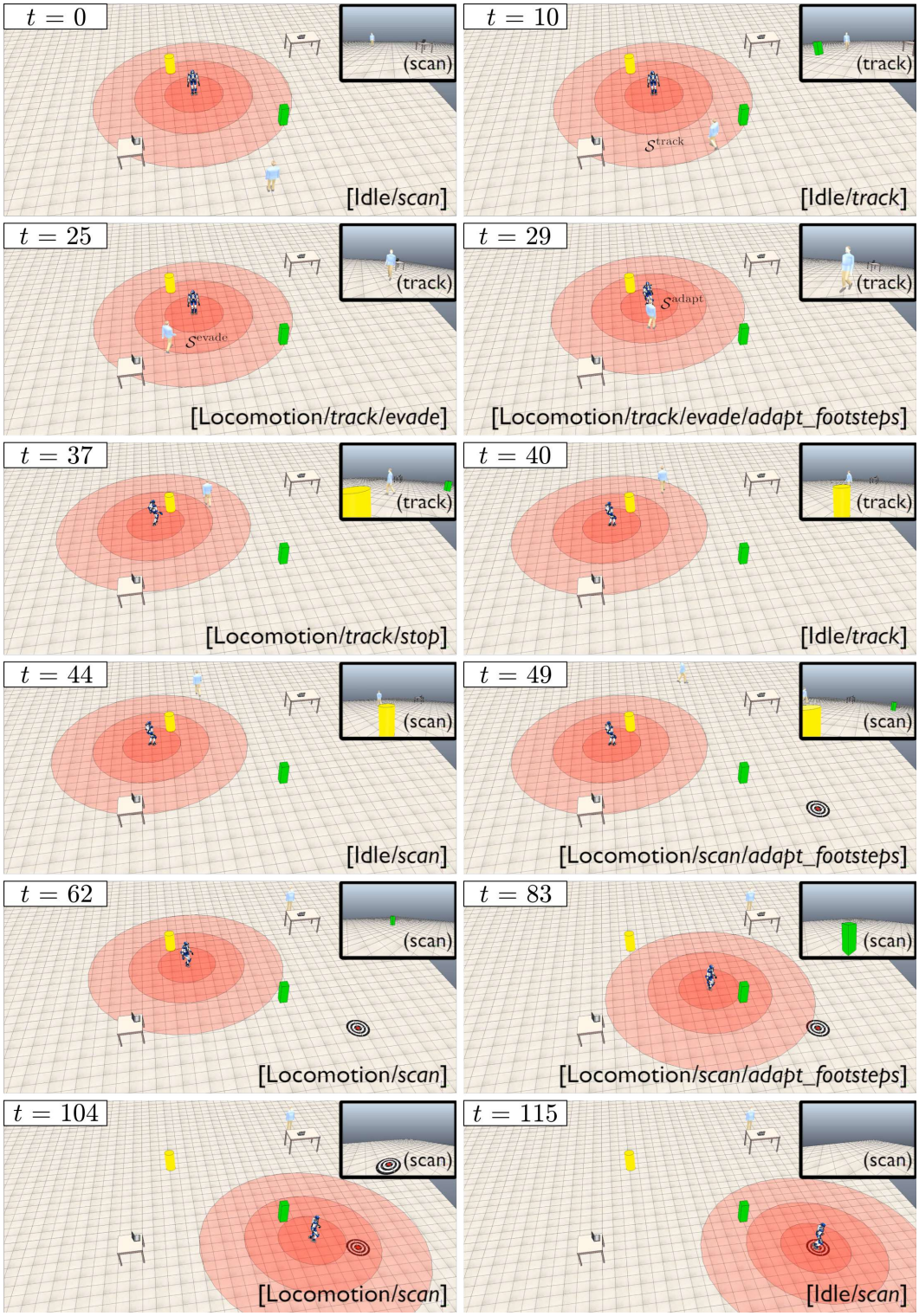

Figure 7.9. Simulation 1: A scenario leading to several safety behaviors being activated in sequence. Snapshots correspond to transitions between states. The humanoid camera view is shown in the upper right corner.

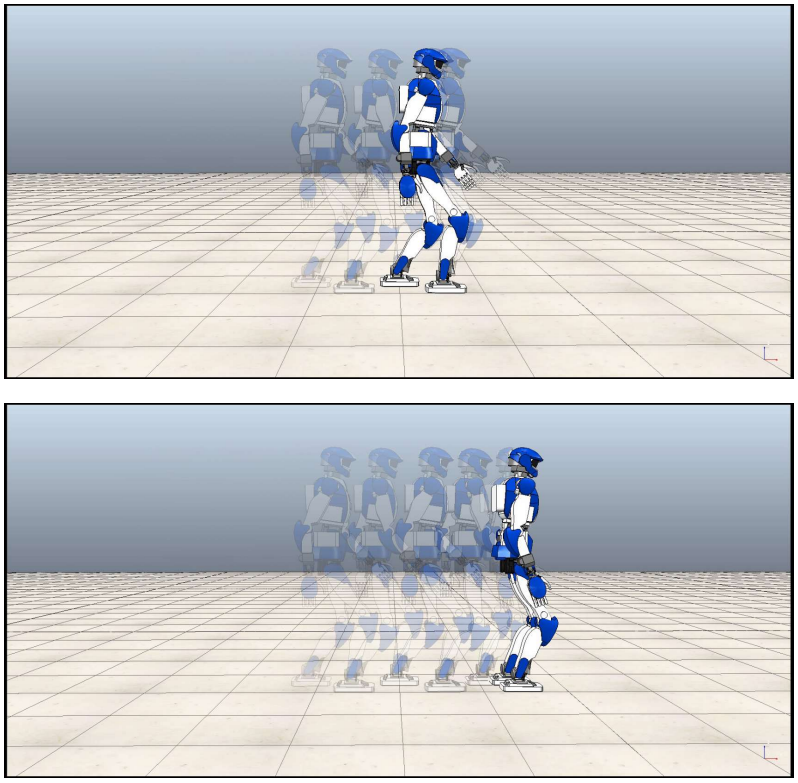

Figure 7.10. Simulation 2: Stroboscopic motion of the robot using *halt* (top) vs *stop* (bottom).

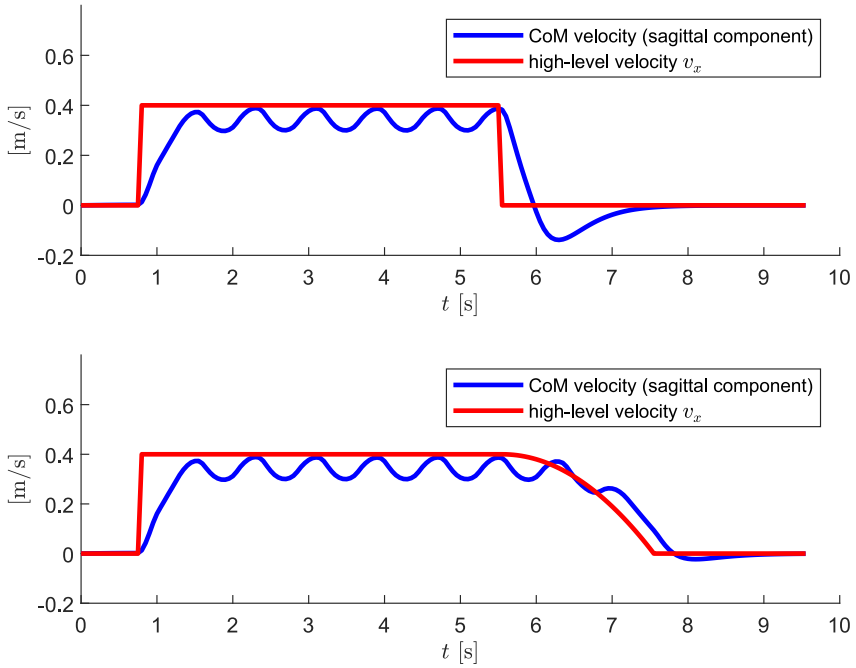

Figure 7.11. Simulation 2: Velocity profiles with *halt* (top) and *stop* (bottom).

as the green object leaves $\mathcal{S}^{\text{adapt}}$ ($t = 104$ s), the robot resumes normal walking (state Locomotion/*scan*) until it reaches the desired goal, where it stops (state Idle/*scan*). Note that the final stop is not the result of a safety behavior; rather, it is produced directly by the gait generator in response to the high-level references velocities vanishing at the goal.

The second simulation, shown in Figs. 7.10–7.11, is aimed at highlighting the difference between the *halt* and *stop* behaviors (see Sects. 7.8.1 and 7.8.3, respectively). The robot is walking normally (state Locomotion/*scan*), with a reference sagittal velocity $v_x = 0.4$ m/s, when the *halt* and *stop* behaviors are respectively triggered during a double support phase. The arrest time used by *stop* is 2 s. As expected, the results indicate that with *halt* the robot stops immediately, almost bouncing back; whereas a much smoother finish is obtained using *stop*.

7.10 Experiments

Experiments were simply designed to showcase different safety behaviors on an actual humanoid platform. Indeed, we were more interested in a 'proof of concept' rather than a quantitative performance evaluation, also considering the fact that the results will be in any case dependent on the specific platform.

In particular, we implemented the proposed approach on NAO. An Asus Xtion PRO Live camera has been mounted over the robot head for measuring depth. Both the camera motion generator and the gait generator run in real-time on the on-board CPU at a control frequency of 100 Hz.

In the first experiment, shown in Fig. 7.12, we use another NAO controlled through a gamepad as an unexpected moving object that pursues our robot. At the beginning, the robot is standing, not performing any task, and scanning the environment (state Idle/*scan*). At $t = 4$ s, the pursuer enters $\mathcal{S}^{\text{track}}$ (we set $d^{\text{track}} = 1$ m). This triggers the *track* behavior (state Idle/*track*). At $t = 7$ s, the pursuer enters $\mathcal{S}^{\text{evade}}$ ($d^{\text{evade}} = 0.6$ m) and the *evade* behavior is activated (state Locomotion/*track*/*evade*). When the pursuer leaves $\mathcal{S}^{\text{evade}}$, at $t = 16$ s, the *stop* behavior is invoked to interrupt the evasion maneuver (state Locomotion/*track*/*stop*); motion is terminated at $t = 17$ s (state Idle/*track*). The robot quits tracking the pursuer when it leaves $\mathcal{S}^{\text{track}}$ at $t = 20$ s (state Idle/*scan*).

The second experiment, shown in Fig. 7.13, focuses on the *adapt_footsteps* behavior. The robot is following a reference sagittal velocity $v_x = 0.08$ m/s (state Locomotion/*scan*) when, at $t = 6$ s an unexpected stationary object (wooden panel) enters $\mathcal{S}^{\text{adapt}}$ (we set $d^{\text{adapt}} = 0.9$ m). Successful obstacle avoidance is produced by footstep adaptation (state Locomotion/*scan*/*adapt_footsteps*), which is deactivated at $t = 9$ s when the obstacle is no more perceived (state Locomotion/*scan*). Note that the panel on the left flank does not trigger *adapt_footsteps* because it never enters $\mathcal{S}^{\text{adapt}}$. The robot then resumes walking in the desired direction.

Although we just reported results from two experiments, a similar successful performance was consistently achieved in our trials, also thanks to the reliability of the underlying MPC-based controller. Clearly, such performance is possible as long as the necessary sensory information is made available to the robot; in this end, robust perception strategies are an essential prerequisite for safety.

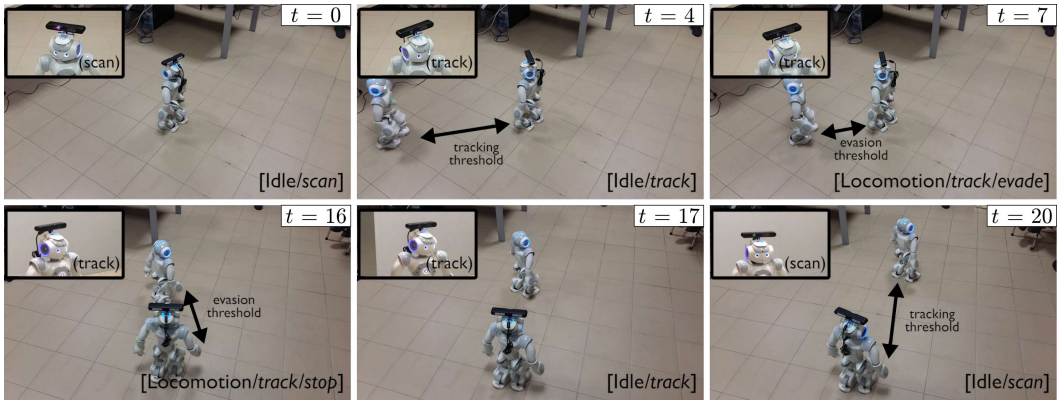

Figure 7.12. Experiment 1: A NAO robot executing an evasion maneuver triggered by another NAO used as a moving object. Snapshots correspond to transitions between states. A close-up of the robot head is shown in the upper left corner.

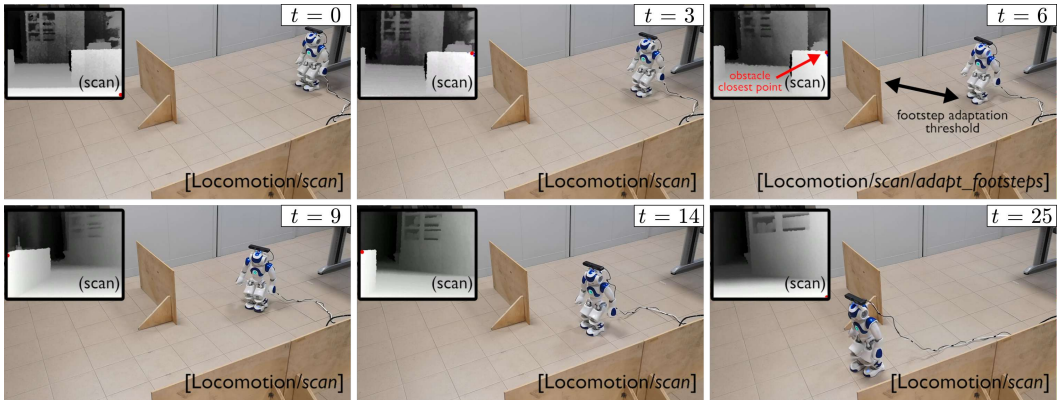

Figure 7.13. Experiment 2: A NAO robot avoiding an unexpected obstacle through footstep adaptation. The humanoid camera view is shown in the upper right corner.

7.11 Discussion

The objective of this section is to provide some additional analysis and details about the proposed method.

7.11.1 Effect of safety on performance

The simulation and experimental results of the last two sections show that the proposed framework effectively increases the overall level of safety, ultimately protecting the robot as well as its co-workers. Obviously, this safety improvement will come at a cost, i.e., a deterioration of performance (in terms of, e.g., time needed to complete a task) due to the more cautious attitude of the robot.

To evaluate the above aspect in detail, we have performed a campaign of simulations focusing on a scenario where an HRP-4 humanoid must execute a walk-to locomotion task in a 25×25 m area. A variable number (1, 3, 5 or 10) of humans walking at 0.2 m/s cross at random the path of the robot. To increase the robot's chances of detecting and avoiding the humans, an omnidirectional camera has been

number of humans	success rate	minimum time	maximum time	average time	success rate without safety framework
1	100%	84.75	111.22	89.64	90%
3	100%	84.75	143.23	113.1	70%
5	100%	100.20	148.10	113.59	70%
10	90%	103.95	163.75	128.37	60%

Table 7.1. HRP-4 executing a walk-to locomotion task in the presence of a variable number of humans: performance data over 10 runs for each scenario.

added to its sensory equipment. As a consequence (see Sect. 7.11.3), the safety behaviors involved in the simulations are *stop*, *evade* and *halt*, for which we have used the following parameters: $d^{\text{evade}} = 3$ m, $d^{\text{halt}} = 1$ m, $\bar{v} = -0.3$ m/s and $k = 0.2$. A simulation is stopped and a failure is recorded when the *halt* behavior is triggered, leading the robot to an Error state. Table 7.1 summarizes the outcome of 10 runs for each scenario, in terms of success rate (how many times the robot was able to complete the task) and completion time (minimum, maximum and average). For comparison, each simulation was also performed without the safety framework, adding however to the control architecture a standard obstacle avoidance module based on artificial potentials [149]; in this case, failure means that collision with a human could not be avoided.

The results in Table 7.1 confirm that our safety framework allows the robot to complete the task in the large majority of cases, even in the presence of many moving humans. As a counterpart, there is a limited increase in the average time needed to complete the task (around 44% going from 1 to 10 humans). Note that the success rate is much lower in the absence of the framework, due to collisions between the robot and the humans.

7.11.2 Limitations of the method

While the results presented so far are clearly positive, one must acknowledge that there are limitations to the proposed method.

1. Robot contexts (Sect. 7.5.1) are separated. For example, the possibility that the humanoid performs a manipulation task while walking is not considered here. Our motivation for excluding such cases is twofold: on the one hand, we are assuming that sensory resources are limited, so that it may be impossible to adequately monitor both the manipulation and the walking area; on the other hand, it is rather obvious that focusing on one task at a time allows to maximize safety. However, an extension allowing execution of simultaneous tasks should be in principle relatively easy to design: one can simply add the combined contexts (e.g., *Loco-manipulation*) to the list and adapt the definition of behaviors to the new contexts.

2. We are looking at the safety problem from the viewpoint of a single robot. If multiple humanoids, all equipped with the proposed framework, are sharing

the same environment, each of them will see the others as unexpected moving obstacles, and perform evasion maneuvers whenever required. While this may not be necessarily optimal, it should be considered that a proper multi-robot safety framework would inevitably require some degree of centralization and inter-robot communication, which may negatively affect the reactiveness of the single robot and the robustness of the safety framework.

3. We are looking at the safety problem in a context of pure coexistence between robots and humans, in the sense that physical collaboration between them is not allowed. This simplifying assumption may be however appropriate for many current applications, especially in industrial contexts where current regulations *de facto* exclude human-humanoid collaboration [119]. However, there is no doubt that in the future such possibility should and will be allowed, making the design of safety frameworks considerably more challenging. Still, we believe that the structure of the proposed approach, based on the definition of override/temporary override/proactive behaviors orchestrated by a state machine, provides a valid template for such extension.

7.11.3 Adaptations

Although the safety guidelines proposed in Sect. 7.2 are completely general, our safety framework is in part dependent on the specific equipment of the humanoid, because safety behaviors were designed based on the sensing assumptions of Sect. 7.3. While this is inevitable, adapting the method to the availability of different measurements is relatively simple.

For example, consider the case in which the humanoid is equipped with an omnidirectional camera, so that the perception area $\mathcal{P}$ becomes a full circle. As a consequence, the robot does not need to direct its gaze, and it becomes possible to observe a moving object while, e.g., scanning the walking area or performing another observation task. This means that *scan* and *track* actually become a single behavior that is always kept active.

Another interesting situation is when the robot can measure relative velocity (direction and magnitude) of moving obstacles with respect to itself. In this case, the trigger of the *evade* behavior may be modified to allow activation only when the moving obstacle is directed 'towards' (in a quantitative sense to be suitably defined) the robot. Allowing an object to cross the robot safety area or not depending on its relative velocity may improve performance (some useless evasion may be avoided) but will obviously increase the level of risk (if the object is a human who brusquely changes direction, there may be no sufficient time left to perform the evasion); a reasonable trade-off between these two aspects must then be found.

Similar adaptations can be derived for other possible variations in the sensory equipment.

Finally, note that the framework description up to Sect. 7.6 (i.e., including the state machine) is independent from the control architecture of the robot. Clearly, any implementation of the framework must take into account (and conform to) such architecture; hence, to offer a worked out example we have first described a possible control architecture based on MPC (Sect. 7.7) and then discussed an implementation

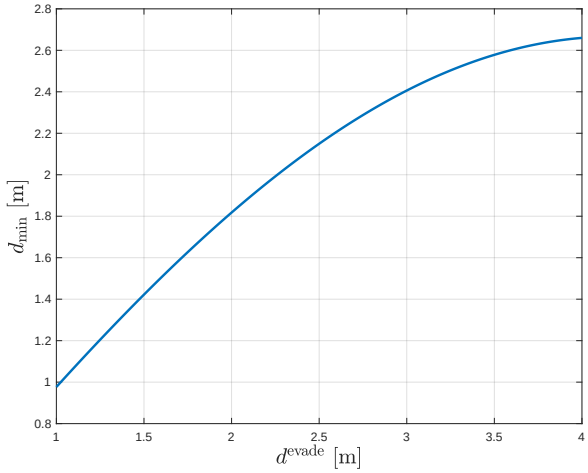

Figure 7.14. Minimum robot-obstacle distance $d_{\min}$ as a function of d^{evade}.

inside it (Sect. 7.8). Implementing the framework in a different control architecture, however, does not pose any conceptual difficulty.

7.11.4 Choice of parameters

A practically relevant issue in the presented framework is the choice of the various parameters, such as the radiuses of the safety areas. Most of these choices can be made on the basis of simple reasoning.

As an example, we discuss below a possible way to determine the threshold d^{evade}, which defines the $\mathcal{S}^{\text{evade}}$ area, based on the desired minimal distance between the robot and a moving obstacle. The worst case to be considered for this scenario is the one in which an obstacle moving at constant speed is heading towards the humanoid along the robot's sagittal axis. When the obstacle enters $\mathcal{S}^{\text{evade}}$, the robot starts an evasion maneuver under the control law (7.5–7.7), hence moving along an arc of circle of radius $R = \bar{v}/k$. Assuming that the obstacle moves at the same speed $\bar{v}$ of the humanoid, a simple computation shows that the minimum distance between the robot and the obstacles takes the value

$$d_{\min} = R\sqrt{2\left(1 - \cos\frac{d^{\text{evade}}}{R}\right)}.$$

This relationship can be used for selecting a value of d^{evade} that guarantees a desired $d_{\min}$. For illustration, in Fig. 7.14 we have plotted $d_{\min}$ as a function of d^{evade} for $\bar{v} = 1$ m/s and $k = 0.75$, corresponding to $R = 4/3$ m. To achieve, say, $d_{\min} = 2.4$ m one should choose $d^{\text{evade}} = 3$ m (as in our simulations).

7.12 Conclusions

In this chapter we have presented a complete framework for the safe deployment of humanoid robots in environments containing humans. This is obtained through the definition of safety behaviors which are differentiated in override, temporary

override and proactive. A state machine handles activation/deactivation of these behaviors based on the information given by the robot sensory system. In the description of the implementation, we focused on locomotion since it is the main aspect which distinguishes humanoids. An MPC setting has been used for realizing all locomotion-related behaviors efficiently. Effectiveness of the proposed method has been shown in dynamic simulation on the HRP-4 humanoid and through experiments on a NAO robot. At the link `https://youtu.be/WmZ0LU30--w`, a video containing clips of the presented results is available.

This work can be improved under several aspects. The main challenge we will consider in the future is going beyond human-robot coexistence to allow physical collaboration between the robot and humans. This obviously raises additional safety problems that can in principle be addressed by properly extending the proposed framework, provided that a communication system has been established between the human and the robot, e.g, using gestures and/or voice commands.

Chapter 8

Conclusions

We have presented a set of techniques for planning the motions of a humanoid robot in different contexts, ranging from known and static environments to unknown and dynamic environments, possibly including humans.

In Chap. 3 we considered the problem of planning whole-body motions for a humanoid robot that must execute a task implicitly requiring locomotion in an environment cluttered by obstacles. For this problem we presented a unified method for planning offline in the presence of a variety of tasks implicitly requiring locomotion, such as navigation, reaching, manipulation and visual tasks. A solution is found by generating possible concatenations of whole-body motions, each one realizing a particular CoM movement primitive selected from a precomputed catalogue. Then, we presented two extensions of the basic method for planning online in case of, respectively, time limitations and unknown environments. We presented simulations in several scenarios of different complexity obtained with the NAO humanoid in the V-REP environment.

Although the approach based on CoM movement primitives revealed to be very effective when dealing with full-fledged task-constraints, as in the case of manipulation or visual tasks, its application in the presence of uneven ground is difficult. For the case of navigation tasks on uneven ground, in Chap. 4, we proposed an integrated motion planner/controller which allows the humanoid to exploit its capability of stepping over or onto obstacles. The proposed architecture is composed by two modules: an offline footstep planner and an online gait generator. For the first module we proposed two possible randomized strategies that can efficiently compute feasible and optimal footstep plans, respectively. We shown both simulation and experimental results using the HRP-4 and NAO humanoids, respectively.

In Chap. 5 we addressed the motion planning problem in the presence of soft tasks, i.e., tasks that are specified by a desired path in task space with an associated error tolerance. We proposed a planner that opportunistically exploits the given tolerance to fulfil the task only when exact task execution is obstructed by the presence of a narrow or closed passage. As a result, the robot performs the assigned task for as long as possible, and deviate from it only when strictly needed to avoid a collision. The proposed strategy is first presented for the case of free-flying robots, and V-REP simulation results are shown using the PR2 mobile manipulator in various scenarios. Then, a possible extension to the case of humanoid robots is

illustrated.

Tasks that require a humanoid to sequentially establish multiple contacts with the environment are considered in Chap. 6, where a randomized multi-contact motion planner is presented. It first produces a sequence of multi-contact states which guarantee static balance, collision avoidance, and feasibility w.r.t. the humanoid kinematic limits. Then, such sequence is converted into an appropriate configuration space trajectory. The proposed technique does not require precomputations or heuristics design, differently from search-based approaches. Preliminary results concerning the stand up task of the COMAN+ humanoid are presented.

Finally, in Chap. 7, we considered the problem of safe coexistence between humans and humanoids where, differently from the other considered contexts, reactive planning capabilities are required. We proposed a complete framework in which a state machine orchestrates the activation/deactivation of several safety behaviors according to information coming from the humanoid sensors. The implementation of the framework is discussed with respect to a reference control architecture in which behaviors related to locomotion are realized in a MPC setting. We shown both simulation and experimental results using the HRP-4 and NAO humanoids, respectively.

In addition to the possible future work already discussed in the concluding section of each chapter, we would like to investigate also the extension of the proposed techniques to different types of legged robots, such as quadrupeds or hybrid wheeled-legged robots. Another interesting direction is the study of the considered motion planning problems in contexts where multiple humanoids share the same environment or even the same task which must be collaboratively executed.

Appendix A

Gait generation via IS-MPC

This appendix provides a quick overview of Intrinsically Stable MPC (IS-MPC) [68], a model predictive control framework for humanoid gait generation that incorporates an explicit CoM/ZMP boundedness constraint in the formulation.

IS-MPC has been introduced for both cases of flat and uneven ground. In the following we briefly recall the main points of this technique for both cases, describing how it generates a stable CoM trajectory that is compatible with a provided footstep sequence. The reader is referred to [68, 90] for further details about IS-MPC.

We recall that, throughout this thesis, IS-MPC has been (*i*) used for generating CoM movement primitives for the humanoid whole-body planning framework presented in Chap. 3, (*ii*) combined with a randomized footstep planner to obtain the integrated method for planning and executing humanoid motions on uneven ground presented in Chap. 4, and (*iii*) involved in the framework for safe human-humanoid coexistence presented in Chap. 7.

A.1 The flat ground case

The basic IS-MPC scheme [68] considers the case of a humanoid robot walking on a flat ground. In this context, the complexity of the full humanoid dynamics is avoided by using the Linear Inverted Pendulum (LIP) model, which works under the simplifying assumption of constant CoM height and absence of rotational effects. The LIP dynamics is expressed by

$$\ddot{x}_{\mathrm{c}} = \eta^2 (x_{\mathrm{c}} - x_{\mathrm{z}}) \tag{A.1}$$

along the x-axis (sagittal motion). Here x_{c} and x_{z} are, respectively, the coordinate of the CoM and the ZMP, and $\eta = \sqrt{g/h_{\mathrm{c}}}$ with g and h_{c}, respectively, the gravity acceleration and the constant CoM height. An identical (and decoupled) equation expresses the dynamics along the y-axis (coronal motion).

Using the change of coordinates

$$x_{\mathrm{s}} = x_{\mathrm{c}} - \dot{x}_{\mathrm{c}}/\eta$$
$$x_{\mathrm{u}} = x_{\mathrm{c}} + \dot{x}_{\mathrm{c}}/\eta$$

the LIP decomposes into a stable and an unstable subsystem

$$\begin{aligned} \dot{x}_\mathrm{s} &= -\eta(x_\mathrm{s} - x_\mathrm{z}) \\ \dot{x}_\mathrm{u} &= \eta(x_\mathrm{u} - x_\mathrm{z}) \end{aligned}$$

where the unstable component x_u is known as *divergent component of motion* [150] or *capture point* [59].

IS-MPC works in a digital fashion over sampling intervals of duration δ. At each t_k, it receives in input a sequence of F footsteps $(X_\mathrm{f}^k, Y_\mathrm{f}^k, \Theta_\mathrm{f}^k)$, with the associated timing $\mathcal{T}_\mathrm{s}^k$, over a *preview horizon* $T_p = P \cdot \delta$ of P sampling intervals in the future

$$\begin{aligned} X_\mathrm{f}^k &= (x_\mathrm{f}^1 \ \ldots \ x_\mathrm{f}^F)^T \\ Y_\mathrm{f}^k &= (y_\mathrm{f}^1 \ \ldots \ y_\mathrm{f}^F)^T \\ \Theta_\mathrm{f}^k &= (\theta_\mathrm{f}^1 \ \ldots \ \theta_\mathrm{f}^F)^T \\ \mathcal{T}_\mathrm{s}^k &= \{T_\mathrm{s}^1 \ \ldots \ T_\mathrm{s}^F\}, \end{aligned}$$

where $(x_\mathrm{f}^j, y_\mathrm{f}^j, \theta_\mathrm{f}^j)$ is the pose of the j-th footstep and T_s^j is the step duration between the $(j-1)$-th and the j-th footstep. Then, IS-MPC predicts CoM and ZMP trajectories over a *control horizon* $T_c = C \cdot \delta$, with $P \geq C$, that are compatible with the timed footstep sequence.

The prediction model used by IS-MPC is a dynamic extension of the LIP, expressed as

$$\begin{pmatrix} \dot{x}_\mathrm{c} \\ \ddot{x}_\mathrm{c} \\ \dot{x}_\mathrm{z} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ \eta^2 & 0 & -\eta^2 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_\mathrm{c} \\ \dot{x}_\mathrm{c} \\ x_\mathrm{z} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \dot{x}_\mathrm{z}, \tag{A.2}$$

for the x component; an analogous expression can be written for the y component. Here, the ZMP velocity $\dot{x}_\mathrm{z}$ is the input, and is assumed to be constant over the sampling intervals, yielding a piecewise-linear ZMP profile.

At each t_k, the ZMP velocities over the control horizon

$$\begin{aligned} \dot{X}_\mathrm{z}^k &= (\dot{x}_\mathrm{z}^k, \ldots, \dot{x}_\mathrm{z}^{k+C-1}) \\ \dot{Y}_\mathrm{z}^k &= (\dot{y}_\mathrm{z}^k, \ldots, \dot{y}_\mathrm{z}^{k+C-1}) \end{aligned}$$

are determined by solving the following QP problem:

$$\min_{\dot{X}_\mathrm{z}^k, \dot{Y}_\mathrm{z}^k} \|\dot{X}_\mathrm{z}^k\|^2 + \|\dot{Y}_\mathrm{z}^k\|^2$$

subject to:

- *ZMP constraints.* These impose the balance condition. Balance of a humanoid walking on flat ground is guaranteed if the ZMP lies at all times within the support polygon of the robot. During single support phases, the ZMP admissible region is the interior of the footstep, approximated as a rectangle of dimensions d_x^z and d_y^z, yielding the constraint

$$-\frac{1}{2}\begin{pmatrix} d_x^\mathrm{z} \\ d_y^\mathrm{z} \end{pmatrix} \leq R_j^T \begin{pmatrix} x_\mathrm{z}^{k+i} - x_\mathrm{f}^j \\ y_\mathrm{z}^{k+i} - y_\mathrm{f}^j \end{pmatrix} \leq \frac{1}{2}\begin{pmatrix} d_x^\mathrm{z} \\ d_y^\mathrm{z} \end{pmatrix}. \tag{A.3}$$

where $(x_{\mathrm{f}}^j, y_{\mathrm{f}}^j)$ and R_j are, respectively, the position and rotation matrix associated with the orientation of the j-th footstep in the control horizon. Constraint (A.3) can be written linearly in terms of the decision variables. During double support phases, to avoid nonlinearities, the latter is approximated by a *moving constraint* [151]; in particular, the ZMP admissible region has exactly the same shape and dimensions it has in single support, and it roto-translates from one footstep to the other. The ZMP constraints involve simultaneously the x and y coordinates and must be verified throughout the control horizon.

- *Stability constraint.* This enforces boundedness of the CoM w.r.t. the ZMP. At each t_k, the *stability condition* [152]

$$x_{\mathrm{u}}^k = \eta \int_{t_k}^{\infty} e^{-\eta(\tau - t_k)} x_{\mathrm{z}}(\tau) d\tau, \tag{A.4}$$

expresses a special initialization of x_{u} at t_k such that the free evolution cancels the divergent component of the forced evolution, thus guaranteeing that the CoM trajectory in output is bounded w.r.t. the ZMP trajectory in input. Condition (A.4) can be written in terms of the decision variables using the piecewise-constant input assumption as

$$\sum_{i=0}^{C-1} e^{-i\eta\delta} \dot{x}_{\mathrm{z}}^{k+i} = -\sum_{i=C}^{\infty} e^{-i\eta\delta} \dot{\tilde{x}}_{\mathrm{z}}^{k+i} + \frac{\eta}{1 - e^{-\eta\delta}} (x_{\mathrm{u}}^k - x_{\mathrm{z}}^k). \tag{A.5}$$

The left-hand side of (A.5) contains the ZMP velocities within the control horizon (decision variables), whereas the right-hand side includes the ZMP velocities beyond the horizon, collectively referred to as the *tail*, which are unknown (hence the tilde) and must be conjectured to obtain a causal version of the constraint. Depending on the assumed behavior of the ZMP velocities $\dot{x}_{\mathrm{z}}$ after the control horizon, three options are possible: (*i*) the *truncated tail* assumes that $\dot{x}_{\mathrm{z}}$ is zero after the control horizon, (*ii*) the *periodic tail* infinitely replicates $\dot{x}_{\mathrm{z}}$ within the control horizon, (*iii*) the *anticipative tail* assumes a more general profile for $\dot{x}_{\mathrm{z}}$ based on the available preview information. The stability constraint is a single scalar condition on each coordinate, thus it must be enforced separately along the x and y axes.

The generic IS-MPC iteration starts at t_k and goes as follows.

1. Compute $\dot{X}_{\mathrm{z}}^k$ and $\dot{Y}_{\mathrm{z}}^k$ that solve the QP problem.
2. Extract the first control samples $\dot{x}_{\mathrm{z}}^k$ and $\dot{y}_{\mathrm{z}}^k$.
3. Set $\dot{x}_{\mathrm{z}} = \dot{x}_{\mathrm{z}}^k$ in (A.2) and integrate from $(x_{\mathrm{c}}^k, \dot{x}_{\mathrm{c}}^k, x_{\mathrm{z}}^k)$ to obtain $x_{\mathrm{c}}(t)$, $\dot{x}_{\mathrm{c}}(t)$, $\dot{x}_{\mathrm{z}}(t)$ for $t \in [t_k, t_{k+1}]$. Do the same for the y component. The 3D CoM trajectory during the considered time interval is $(x_{\mathrm{c}}, y_{\mathrm{c}}, h_{\mathrm{c}})$.

IS-MPC has been shown to be recursively feasible and internally stable (in other words, ZMP-to-CoM stable) provided that the anticipative tail is used and the preview horizon is sufficiently long, i.e., if P is large enough.

In some cases, e.g., in the presence of disturbances, it might be necessary to modify the provided footstep sequence in order to maintain balance. To this end, the basic IS-MPC scheme can be slightly modified in order to obtain *automatic footstep placement*. In particular, the footstep sequence in input provides now only candidate footstep positions[1], denoted as $\hat{X}_{\mathrm{f}}^k$ and $\hat{Y}_{\mathrm{f}}^k$, while the actual ones (that acts as additional decision variables), denoted as X_{f}^k and Y_{f}^k, are determined together with the ZMP velocities over the control horizon by solving the following QP problem:

$$\min_{\dot{X}_{\mathrm{z}}^k, \dot{Y}_{\mathrm{z}}^k, X_{\mathrm{f}}^k, Y_{\mathrm{f}}^k} \|\dot{X}_{\mathrm{z}}^k\|^2 + \|\dot{Y}_{\mathrm{z}}^k\|^2 + \beta(\|X_{\mathrm{f}}^k - \hat{X}_{\mathrm{f}}^k\|^2 + \|Y_{\mathrm{f}}^k - \hat{Y}_{\mathrm{f}}^k\|^2)$$

subject to:

- *ZMP* and *stability constraints* as before.
- *Kinematic constraints*. These ensure that the generated footsteps are feasible w.r.t. the kinematic limits of the specific humanoid being considered.

Note the additional term in the cost function that aims at placing the footsteps as close as possible to the candidate ones.

A.2 The uneven ground case

The extension of IS-MPC proposed in [90] allows to generate variable height CoM trajectories for a humanoid walking on a specific kind of uneven ground, renamed in [4] as *world of stairs* (see also Chap. 4), composed by horizontal patches located at different heights.

The 3D dynamics of the CoM can be written as

$$\begin{aligned} \ddot{x}_{\mathrm{c}} &= \frac{\ddot{z}_{\mathrm{c}} + g}{z_{\mathrm{c}} - z_{\mathrm{z}}}(x_{\mathrm{c}} - x_{\mathrm{z}}) \\ \ddot{y}_{\mathrm{c}} &= \frac{\ddot{z}_{\mathrm{c}} + g}{z_{\mathrm{c}} - z_{\mathrm{z}}}(y_{\mathrm{c}} - y_{\mathrm{z}}) \\ \ddot{z}_{\mathrm{c}} &= \frac{f_{\mathrm{z}}}{m} - g, \end{aligned}$$

where the first two equations are obtained from the moment balance around the ZMP, and the third from the force balance along the the z axis, with f_{z} and m denoting, respectively, the z-component of the ground reaction force and the total mass of the humanoid.

By setting $(\ddot{z}_{\mathrm{c}} + g)/(z_{\mathrm{c}} - z_{\mathrm{z}}) = \eta^2$, with η an arbitrary constant, a LIP-like model is obtained

$$\begin{aligned} \ddot{x}_{\mathrm{c}} &= \eta^2(x_{\mathrm{c}} - x_{\mathrm{z}}) \\ \ddot{y}_{\mathrm{c}} &= \eta^2(y_{\mathrm{c}} - y_{\mathrm{z}}) \\ \ddot{z}_{\mathrm{c}} &= \eta^2(z_{\mathrm{c}} - z_{\mathrm{z}}) - g, \end{aligned} \tag{A.6}$$

which allows vertical motion of the CoM, and is linear along the x and y axes, and affine along the z axis.

[1] Orientations and timing are not modified to preserve linearity.

Similarly to the flat ground case, at each t_k, IS-MPC receives in input a sequence of F footsteps $(X_{\mathrm{f}}^k, Y_{\mathrm{f}}^k, Z_{\mathrm{f}}^k, \Theta_{\mathrm{f}}^k)$, with the associated timing $\mathcal{T}_{\mathrm{s}}^k$, over a preview horizon T_p. Note that this time, because of the unevenness of the ground, also the z coordinate of each footstep is provided (through Z_{f}^k). Then, IS-MPC predicts CoM and ZMP trajectories over a control horizon T_c that are compatible with the timed footstep sequence.

The prediction model is a dynamic extension of the model in (A.6), which for the x and y components is identical to that in (A.2), while for the z component it has the additive term g.

At each t_k, the ZMP velocities $(\dot{X}_{\mathrm{z}}^k, \dot{Y}_{\mathrm{z}}^k, \dot{Z}_{\mathrm{z}}^k)$ over the control horizon are determined by solving the following QP problem:

$$\min_{\dot{X}_{\mathrm{z}}^k, \dot{Y}_{\mathrm{z}}^k, \dot{Z}_{\mathrm{z}}^k} \|\dot{X}_{\mathrm{z}}^k\|^2 + \|\dot{Y}_{\mathrm{z}}^k\|^2 + \|\dot{Z}_{\mathrm{z}}^k\|^2 + \beta \|Z_{\mathrm{z}}^k - Z_{\mathrm{f}}^k\|^2$$

subject to:

- *ZMP constraints.* Balance of a humanoid walking on the considered kind of uneven ground is guaranteed if the Center of Pressure (CoP) lies at all times within the support polygon of the robot. Since the CoP, the CoM and the ZMP are colinear, such condition is equivalent to the ZMP being internal to the polyhedral cone having the CoM as vertex and the support polygon as cross-section, leading to a nonlinear constraint. In order to remove this nonlinearity, a conservative approximation of the polyhedral cone is employed, leading to the linear (box) constraint

$$-\frac{1}{2}\begin{pmatrix} \tilde{d}_x^{\mathrm{z}} \\ \tilde{d}_y^{\mathrm{z}} \\ d_z^{\mathrm{z}} \end{pmatrix} \leq R_j^T \begin{pmatrix} x_{\mathrm{z}}^{k+i} - x_{\mathrm{f}}^j \\ y_{\mathrm{z}}^{k+i} - y_{\mathrm{f}}^j \\ z_{\mathrm{z}}^{k+i} - z_{\mathrm{f}}^j \end{pmatrix} \leq \frac{1}{2}\begin{pmatrix} \tilde{d}_x^{\mathrm{z}} \\ \tilde{d}_y^{\mathrm{z}} \\ d_z^{\mathrm{z}} \end{pmatrix}. \tag{A.7}$$

  where $\tilde{d}_x^{\mathrm{z}}$ and $\tilde{d}_y^{\mathrm{z}}$ are the reduced horizontal sizes of the constraint, chosen to be compatible with the vertical size d_z^{z} which is a design parameter.

  As in the flat ground case, the box constraint is fixed during single support phases, while it roto-translates from one footstep to the other during double support phases.

- *Stability constraint.* The stability condition along the x and y axes are identical to those for the flat ground case, and consequently the resulting constraints as well (see (A.4) and (A.5), respectively). Along the z axis a slightly different stability condition is obtained

$$z_{\mathrm{u}}^k = \frac{g}{\eta^2} + \eta \int_{t_k}^{\infty} e^{-\eta(\tau - t_k)} z_{\mathrm{z}}(\tau) d\tau, \tag{A.8}$$

  which results in the stability constraint for the z component

$$\sum_{i=0}^{C-1} e^{-i\eta\delta} \dot{z}_{\mathrm{z}}^{k+i} = -\sum_{i=C}^{\infty} e^{-i\eta\delta} \dot{z}_{\mathrm{z}}^{k+i} + \frac{\eta}{1 - e^{-\eta\delta}} \left(z_{\mathrm{u}}^k - z_{\mathrm{z}}^k - \frac{g}{\eta^2} \right). \tag{A.9}$$

  As before, three options are possible for conjecturing the ZMP velocities beyond the control horizon, namely the truncated, periodic and anticipative tails.

Note that, in addition to the regularization terms, the cost function includes an additional term which attempts to bring the ZMP to patch level whenever possible. Analogously to before, the generic IS-MPC iteration starts at t_k and goes as follows.

1. Compute $\dot{X}_{\mathrm{z}}^k$, $\dot{Y}_{\mathrm{z}}^k$ and $\dot{Z}_{\mathrm{z}}^k$ that solve the QP problem.
2. Extract the first control samples $\dot{x}_{\mathrm{z}}^k$, $\dot{y}_{\mathrm{z}}^k$ and $\dot{z}_{\mathrm{z}}^k$.
3. Set $\dot{x}_{\mathrm{z}} = \dot{x}_{\mathrm{z}}^k$ in (A.2) and integrate from $(x_{\mathrm{c}}^k, \dot{x}_{\mathrm{c}}^k, x_{\mathrm{z}}^k)$ to obtain $x_{\mathrm{c}}(t)$, $\dot{x}_{\mathrm{c}}(t)$, $\dot{x}_{\mathrm{z}}(t)$ for $t \in [t_k, t_{k+1}]$. Do the same for the y and z components. The 3D CoM trajectory during the considered time interval is $(x_{\mathrm{c}}, y_{\mathrm{c}}, z_{\mathrm{c}})$.

Bibliography

[1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[2] P. Ferrari, M. Cognetti, and G. Oriolo, "Anytime whole-body planning/replanning for humanoid robots," in *18th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 1–9, 2018.

[3] P. Ferrari, M. Cognetti, and G. Oriolo, "Sensor-based whole-body planning/replanning for humanoid robots," in *19th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 511–517, 2019.

[4] P. Ferrari, N. Scianca, L. Lanari, and G. Oriolo, "An integrated motion planner/controller for humanoid robots on uneven ground," in *2019 18th European Control Conference (ECC)*, pp. 1598–1603, 2019.

[5] M. Cefalo, P. Ferrari, and G. Oriolo, "An opportunistic strategy for motion planning in the presence of soft task constraints," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6294–6301, 2020.

[6] N. Scianca, P. Ferrari, D. De Simone, L. Lanari, and G. Oriolo, "A behavior-based framework for safe deployment of humanoid robots," *Autonomous Robots*, 2021.

[7] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, London, 2009.

[8] J. T. Schwartz and M. Sharir, "On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, no. 3, pp. 298–351, 1983.

[9] C. Ó'Dúnlaing and C. K. Yap, "A "retraction" method for planning the motion of a disc," *Journal of Algorithms*, vol. 6, no. 1, pp. 104–111, 1985.

[10] J. Canny, *The complexity of robot motion planning*. MIT press, 1988.

[11] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[12] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions*, no. 5, pp. 293–308, 2001.

[13] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *Int. J. of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *2007 IEEE Int. Conf. on Robotics and Automation*, pp. 1603–1609, 2007.

[16] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[17] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT* a real-time path planning algorithm based on RRT," in *8th ACM SIGGRAPH Conference on Motion in Games*, pp. 113–118, 2015.

[18] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *1985 IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 500–505, 1985.

[19] E. Rimon and D. E. Koditschek, "The construction of analytic diffeomorphisms for exact robot navigation on star worlds," *Transactions of the American Mathematical Society*, vol. 327, no. 1, pp. 71–116, 1991.

[20] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[21] M. Cefalo, E. Magrini, and G. Oriolo, "Sensor-based task-constrained motion planning using model predictive control," *IFAC-PapersOnLine*, vol. 51, pp. 220–225, 2018.

[22] G. B. Avanzini, A. M. Zanchettin, and P. Rocco, "Constrained model predictive control for mobile robotic manipulators," *Robotica*, vol. 36, no. 1, p. 19, 2018.

[23] M. Logothetis, G. C. Karras, S. Heshmati-Alamdari, P. Vlantis, and K. J. Kyriakopoulos, "A model predictive control approach for vision-based object grasping via mobile manipulator," in *2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1–6, 2018.

[24] J. Kuffner, J.J., K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Footstep planning among obstacles for biped robots," in *2001 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 500–505, 2001.

[25] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade, "Footstep planning for the honda ASIMO humanoid," in *2005 IEEE Int. Conf. on Robotics and Automation*, pp. 629–634, 2005.

[26] N. Perrin, O. Stasse, L. Baudouin, F. Lamiraux, and E. Yoshida, "Fast humanoid robot collision-free footstep planning using swept volume approximations," *IEEE Trans. on Robotics*, vol. 28, no. 2, pp. 427–439, 2012.

[27] N. Perrin, "Biped footstep planning," in *Humanoid Robotics: A Reference*, Springer Netherlands, 2018.

[28] J. Pettré, J.-P. Laumond, and T. Siméon, "A 2-stages locomotion planner for digital actors," in *2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 258–264, 2003.

[29] E. Yoshida, I. Belousov, C. Esteves, and J.-P. Laumond, "Humanoid motion planning for dynamic tasks," in *5th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 1–6, 2005.

[30] M. Vukobratovic and D. Juricic, "Contribution to the synthesis of biped gait," *IEEE Trans. on Biomedical Engineering*, no. 1, pp. 1–6, 1969.

[31] S. Chiaverini, G. Oriolo, and A. A. Maciejewski, "Redundant robots," in *Springer Handbook of Robotics*, pp. 221–242, Springer, 2016.

[32] O. Kanoun, F. Lamiraux, and P. B. Wieber, "Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task," *IEEE Trans. on Robotics*, vol. 27, no. 4, pp. 785–792, 2011.

[33] D. Rakita, B. Mutlu, and M. Gleicher, "Stampede: A discrete-optimization method for solving pathwise-inverse kinematics," in *2019 IEEE Int. Conf. on Robotics and Automation*, pp. 3507–3513, 2019.

[34] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 159–185, 2018.

[35] G. Oriolo and C. Mongillo, "Motion planning for mobile manipulators along given end-effector paths," in *2005 IEEE Int. Conf. on Robotics and Automation*, pp. 2166–2172, 2005.

[36] B. Stephens, "Humanoid push recovery," in *7th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 589–595, 2007.

[37] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *Int. J. of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[38] R. Shome and K. E. Bekris, "Improving the scalability of asymptotically optimal motion planning for humanoid dual-arm manipulators," in *2017 IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 271–277, 2017.

[39] G. Oriolo and M. Vendittelli, "A control-based approach to task-constrained motion planning," in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 297–302, 2009.

[40] F. Burget, A. Hornung, and M. Bennewitz, "Whole-body motion planning for manipulation of articulated objects," in *2013 IEEE Int. Conf. on Robotics and Automation*, 2013.

[41] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *2000 IEEE Int. Conf. on Robotics and Automation*, pp. 995–1001, 2000.

[42] Y. Yang, V. Ivan, W. Merkt, and S. Vijayakumar, "Scaling sampling-based motion planning to humanoid robots," in *2016 IEEE Int. Conf. on Robotics and Biomimetics*, pp. 1448–1454, 2016.

[43] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Efficient motion planning for humanoid robots using lazy collision checking and enlarged robot models," in *2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 3062–3067, 2007.

[44] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots under obstacle and dynamic balance constraints," in *2001 IEEE Int. Conf. on Robotics and Automation*, pp. 692–698, 2001.

[45] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic walking and whole-body motion planning for humanoid robots: An integrated approach," *Int. J. of Robotics Research*, vol. 32, no. 9–10, pp. 1089–1103, 2013.

[46] 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*, pp. 1620–1626, 2003.

[47] S. J. Jorgensen, M. Vedantam, R. Gupta, H. Cappel, and L. Sentis, "Finding locomanipulation plans quickly in the locomotion constrained manifold," in *2020 IEEE Int. Conf. on Robotics and Automation*, pp. 6611–6617, 2020.

[48] K. Bouyarmane, S. Caron, A. Escande, and A. Kheddar, "Multi-contact motion planning and control," in *Humanoid Robotics: A Reference*, pp. 1–42, Springer Netherlands, 2018.

[49] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *Int. J. of Robotics Research*, vol. 27, no. 11-12, pp. 1325–1349, 2008.

[50] K. Bouyarmane and A. Kheddar, "Humanoid robot locomotion and manipulation step planning," *Advanced Robotics*, vol. 26, no. 10, pp. 1099–1126, 2012.

[51] A. Escande, A. Kheddar, and S. Miossec, "Planning contact points for humanoid robots," *Robotics and Autonomous Systems*, vol. 61, no. 5, pp. 428–442, 2013.

[52] D. E. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, vol. 35, no. 2-3, pp. 161–176, 2013.

[53] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade, "Vision-guided humanoid footstep planning for dynamic environments," in *5th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 13–18, 2005.

[54] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in *12th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 692–697, 2012.

[55] N. Bohorquez, A. Sherikov, D. Dimitrov, and P. B. Wieber, "Safe navigation strategies for a biped robot walking in a crowd," in *16th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 379–386, 2016.

[56] M. Naveau, M. Kudruss, O. Stasse, C. Kirches, K. Mombaur, and P. Souères, "A reactive walking pattern generator based on nonlinear model predictive control," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 10–17, 2017.

[57] M. Cognetti, D. De Simone, L. Lanari, and G. Oriolo, "Real-time planning and execution of evasive motions for a humanoid robot," in *2016 IEEE Int. Conf. on Robotics and Automation*, pp. 4200–4206, 2016.

[58] M. Morisawa, K. Harada, S. Kajita, K. Kaneko, J. Sola, E. Yoshida, N. Mansard, K. Yokoi, and J.-P. Laumond, "Reactive stepping to prevent falling for humanoids," in *9th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 528–534, 2009.

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

[60] M. Cognetti, P. Mohammadi, and G. Oriolo, "Whole-body motion planning for humanoids based on com movement primitives," in *15th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 1090–1095, IEEE, 2015.

[61] L. Baudouin, N. Perrin, T. Moulard, F. Lamiraux, O. Stasse, and E. Yoshida, "Real-time replanning using 3D environment for humanoid robot," in *11th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 584–589, 2011.

[62] K. Okada, T. Ogura, A. Haneda, and M. 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*, pp. 623–628, 2005.

[63] P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade, "Gpu-accelerated real-time 3d tracking for humanoid locomotion and stair climbing," in *2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 463–469, 2007.

[64] J. Chestnutt, Y. Takaoka, K. Suga, K. Nishiwaki, J. Kuffner, and S. Kagami, "Biped navigation in rough environments using on-board sensing," in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 3543–3548, 2009.

[65] K. Nishiwaki, J. Chestnutt, and S. Kagami, "Autonomous navigation of a humanoid robot over unknown rough terrain using a laser range sensor," *Int. J. of Robotics Research*, vol. 31, no. 11, pp. 1251–1262, 2012.

[66] A. Nakhaei and F. Lamiraux, "Motion planning for humanoid robots in environments modeled by vision," in *8th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 197–204, 2008.

[67] D. Maier, C. Lutz, and M. Bennewitz, "Integrated perception, mapping, and footstep planning for humanoid navigation among 3d obstacles," in *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2658–2664, 2013.

[68] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo, "MPC for humanoid gait generation: Stability and feasibility," *IEEE Trans. on Robotics*, vol. 36, no. 4, pp. 1171–1178, 2020.

[69] P. Ferrari, M. Cognetti, and G. Oriolo, "Humanoid whole-body planning for loco-manipulation tasks," in *2017 IEEE Int. Conf. on Robotics and Automation*, pp. 4741–4746, 2017.

[70] N. Shahriari, S. Fantasia, F. Flacco, and G. Oriolo, "Robotic visual servoing of moving targets," in *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 77–82, 2013.

[71] F. Chaumette, S. Hutchinson, and P. Corke, "Visual servoing," in *Springer Handbook of Robotics*, pp. 841–866, Springer, 2016.

[72] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *2000 IEEE Int. Conf. on Robotics and Automation*, pp. 521–528, 2000.

[73] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics Research*, pp. 403–417, 2003.

[74] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[75] M. Cefalo and G. Oriolo, "Dynamically feasible task-constrained motion planning with moving obstacles," in *2014 IEEE Int. Conf. on Robotics and Automation*, 2014.

[76] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 279–286, 2014.

[77] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *2012 IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 674–679, 2012.

[78] R. J. Griffin, G. Wiedebach, S. McCrory, S. Bertrand, I. Lee, and J. Pratt, "Footstep planning for autonomous walking over rough terrain," in *19th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 9–16, 2019.

[79] Z. Xia, G. Chen, J. Xiong, Q. Zhao, and K. Chen, "A random sampling-based approach to goal-directed footstep planning for humanoid robots," in *2009 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics*, pp. 168–173, 2009.

[80] S. Kajita and K. Tani, "Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode," in *1991 IEEE Int. Conf. on Robotics and Automation*, pp. 1405–1406, 1991.

[81] P.-B. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *6th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 137–142, 2006.

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

[83] S. Caron, A. Escande, L. Lanari, and B. Mallein, "Capturability-based pattern generation for walking with variable height," *IEEE Trans. on Robotics*, vol. 36, no. 2, pp. 517–536, 2019.

[84] A. Herdt, *Model predictive control of a humanoid robot.* PhD thesis, Ecole Nationale Supérieure des Mines de Paris, 2012.

[85] M. A. Hopkins, D. W. Hong, and A. Leonessa, "Humanoid locomotion on uneven terrain using the time-varying divergent component of motion," in *14th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 266–272, 2014.

[86] K. Terada and Y. Kuniyoshi, "Online gait planning with dynamical 3d-symmetrization method," in *7th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 222–227, 2007.

[87] R. C. Luo, P. H. Chang, J. Sheng, S. C. Gu, and C. H. Chen, "Arbitrary biped robot foot gaiting based on variate com height," in *13th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 534–539, 2013.

[88] J. Englsberger, C. Ott, and A. Albu-Schäffer, "Three-dimensional bipedal walking control based on divergent component of motion," *IEEE Trans. on Robotics*, vol. 31, no. 2, pp. 355–368, 2015.

[89] W. Burgard, M. Hebert, and M. Bennewitz, "World modeling," in *Springer Handbook of Robotics*, pp. 1135–1152, Springer, 2016.

[90] A. Zamparelli, N. Scianca, L. Lanari, and G. Oriolo, "Humanoid gait generation on uneven ground using intrinsically stable MPC," *IFAC-PapersOnLine*, vol. 51, pp. 393–398, 2018.

[91] T. Röfer, T. Laue, J. Richter-Klug, M. Schünemann, J. Stiensmeier, A. Stolpmann, A. Stöwing, and F. Thielke, "B-Human team report and code release 2015," 2015. Only available online: `http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf`.

[92] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019–3026, 2018.

[93] G. Oriolo, M. Cefalo, and M. Vendittelli, "Repeatable motion planning for redundant robots over cyclic tasks," *IEEE Trans. on Robotics*, vol. 33, no. 5, pp. 1170–1183, 2017.

[94] M. Cefalo and G. Oriolo, "A general framework for task-constrained motion planning with moving obstacles," *Robotica*, vol. 37, pp. 575–598, 2019.

[95] T. Kunz and M. Stilman, "Manipulation planning with soft task constraints," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1937–1942, 2012.

[96] M. Guo and M. M. Zavlanos, "Probabilistic motion planning under temporal tasks and soft constraints," *IEEE Trans. on Automatic Control*, vol. 63, no. 12, pp. 4051–4066, 2018.

[97] I. Sucan and S. Chitta, "Motion planning with constraints using configuration space approximations," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1904–1910, 2012.

[98] Z. Fusheng, L. Yizhou, G. Wei, W. Pengfei, L. Mantian, W. Xin, and L. Jingxuan, "Learning the metric of task constraint manifolds for constrained motion planning," *Electronics*, vol. 7, p. 395, 2018.

[99] V. Boor, M. Overmars, and A. V. der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *1999 IEEE Int. Conf. on Robotics and Automation*, pp. 1018–1023, 1999.

[100] D. Hsu, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *2003 IEEE Int. Conf. on Robotics and Automation*, pp. 4420–4426, 2003.

[101] Z. Sadeghi and H. Moradi, "A new sample-based strategy for narrow passage detection," in *2011 9th IEEE World Congress on Intelligent Control and Automation*, pp. 1059–1064, 2011.

[102] M. Saha, J.-C. Latombe, Y.-C. Chang, and F. Prinz, "Finding narrow passages with probabilistic roadmaps: The small-step retraction method," *Autonomous Robots*, vol. 19, no. 3, pp. 301–319, 2005.

[103] D. Berenson, T. Siméon, and S. Srinivasa, "Addressing cost-space chasms in manipulation planning," in *2011 IEEE Int. Conf. on Robotics and Automation*, pp. 4561–4568, 2011.

[104] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Trans. on Robotics*, vol. 34, no. 3, pp. 586–601, 2018.

[105] Y. Guan and K. Yokoi, "Reachable space generation of a humanoid robot using the monte carlo method," in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1984–1989, 2006.

[106] L. Jamone, L. Natale, G. Sandini, and A. Takanishi, "Interactive online learning of the kinematic workspace of a humanoid robot," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2606–2612, 2012.

[107] M. P. Polverini, A. Laurenzi, E. M. Hoffman, F. Ruscelli, and N. G. Tsagarakis, "Multi-contact heavy object pushing with a centaur-type humanoid robot: Planning and control for a real demonstrator," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 859–866, 2020.

[108] F. Ruscelli, M. P. Polverini, A. Laurenzi, E. M. Hoffman, and N. G. Tsagarakis, "A multi-contact motion planning and control strategy for physical interaction tasks using a humanoid robot," in *2020 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2020.

[109] A. Laurenzi, E. M. Hoffman, L. Muratore, and N. G. Tsagarakis, "CartesI/O: A ROS based real-time capable cartesian control framework," in *2019 IEEE Int. Conf. on Robotics and Automation*, pp. 591–596, 2019.

[110] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[111] A. Laurenzi, E. M. Hoffman, M. P. Polverini, and N. G. Tsagarakis, "Balancing control through post-optimization of contact forces," in *18th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 320–326, 2018.

[112] A. Bicchi and G. Tonietti, "Fast and "soft-arm" tactics," *IEEE Robotics and Automation Magazine*, vol. 11, pp. 22–33, June 2004.

[113] A. De Santis, B. Siciliano, A. De Luca, and A. Bicchi, "An atlas of physical human–robot interaction," *Mechanism and Machine Theory*, vol. 43, no. 3, pp. 253–270, 2008.

[114] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726 – 1743, 2013.

[115] T. S. Tadele, T. de Vries, and S. Stramigioli, "The safety of domestic robotics: A survey of various safety-related publications," *IEEE Robotics and Automation Magazine*, vol. 21, no. 3, pp. 134–142, 2014.

[116] A. De Luca and F. Flacco, "Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration," in *2012 IEEE RAS & EMBS Int. Conf. on Biomedical Robotics and Biomechatronics*, pp. 288–295, 2012.

[117] B. Lacevic, P. Rocco, and A. Zanchettin, "Safety assessment and control of robotic manipulators using danger field," *IEEE Trans. on Robotics*, vol. 29, no. 5, pp. 1257–1270, 2013.

[118] B. Navarro, A. Fonte, P. Fraisse, G. Poisson, and A. Cherubini, "In pursuit of safety: An open-source library for physical human-robot interaction," *IEEE Robotics and Automation Magazine*, vol. 25, pp. 39–50, June 2018.

[119] A. Kheddar, S. Caron, P. Gergondet, A. Comport, A. Tanguy, C. Ott, B. Henze, G. Mesesan, J. Englsberger, M. A. Roa, P.-B. Wieber, F. Chaumette, F. Spindler, G. Oriolo, L. Lanari, A. Escande, K. Chappellet, F. Kanehiro, and P. Rabaté, "Humanoid robots in aircraft manufactoring – the Airbus use-case," *IEEE Robotics and Automation Magazine*, vol. 26, pp. 30–45, Dec. 2019.

[120] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to Humanoid Robotics*. Springer Publishing Company Inc., 2014.

[121] E. Krotkov, D. Hackett, L. Jackel, M. Perschbacher, J. Pippine, J. Strauss, G. Pratt, and C. Orlowski, "The DARPA Robotics Challenge finals: Results and perspectives," *Journal of Field Robotics*, vol. 34, no. 2, pp. 229–240, 2017.

[122] C. G. Atkeson, P. B. Benzun, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, *et al.*, "What happened at the DARPA Robotics Challenge finals," in *The DARPA Robotics Challenge Finals: Humanoid Robots to the Rescue*, pp. 667–684, Springer, 2018.

[123] J. Lim, I. Lee, I. Shim, H. Jung, H. M. Joe, H. Bae, O. Sim, J. Oh, T. Jung, S. Shin, K. Joo, M. Kim, K. Lee, Y. Bok, D.-G. Choi, B. Cho, S. Kim, J. Heo, I. Kim, J. Lee, I. S. Kwon, and J.-H. Oh, "Robot system of DRC-HUBO+ and control strategy of team KAIST in DARPA Robotics Challenge finals," *Journal of Field Robotics*, vol. 34, no. 4, pp. 802–829, 2017.

[124] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer Handbook of Robotics*, pp. 1203–1234, Springer, 2016.

[125] P. Marion, M. Fallon, R. Deits, A. Valenzuela, C. Pérez D'Arpino, G. Izatt, L. Manuelli, M. Antone, H. Dai, T. Koolen, *et al.*, "Director: A user interface designed for robot operation with shared autonomy," *Journal of Field Robotics*, vol. 34, no. 2, pp. 262–280, 2017.

[126] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: a systematic survey," *IEEE Transactions on Image Processing*, vol. 14, pp. 294–307, March 2005.

[127] M. Stark, B. Schiele, and A. Leonardis, "Visual object class recognition," in *Springer Handbook of Robotics*, pp. 825–840, Springer, 2016.

[128] F. Flacco, A. Paolillo, and A. Kheddar, "Residual-based contacts estimation for humanoid robots," in *16th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 409–415, 2016.

[129] K. Ogata, K. Terada, and Y. Kuniyoshi, "Falling motion control for humanoid robots while walking," in *7th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 306–311, 2007.

[130] S. Caron, Q.-C. Pham, and Y. Nakamura, "ZMP support areas for multi-contact mobility under frictional constraints," *IEEE Trans. on Robotics*, vol. 33, no. 1, pp. 67–80, 2017.

[131] E. T. Hall, *The hidden dimension*. Garden City, N.Y.: Doubleday, 1966.

[132] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "From proxemics theory to socially-aware navigation: A survey," *Int. Journal of Social Robotics*, vol. 7, pp. 137–153, June 2014.

[133] F. Chaumette and S. Hutchinson, "Visual servo control. I. basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[134] K. Fujiwara, F. Kanehiro, S. Kajita, K. Yokoi, H. Saito, K. Harada, K. Kaneko, and H. Hirukawa, "The first human-size humanoid that can fall over safely and stand-up again," in *2003 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 2, pp. 1920–1926, 2003.

[135] K. Fujiwara, F. Kanehiro, S. Kajita, and H. Hirukawa, "Safe knee landing of a human-size humanoid robot while falling forward," in *2004 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 1, pp. 503–508, 2004.

[136] A. Yasin, Q. Huang, Z. Yu, Q. Xu, and A. A. Syed, "Stepping to recover: A 3D-LIPM based push recovery and fall management scheme for biped robots," in *2012 IEEE Int. Conf. on Robotics and Biomimetics*, pp. 318–323, 2012.

[137] F. Braghin, B. Henze, and M. Roa, "Optimal trajectory for active safe falls in humanoid robots," in *19th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 305–312, 2019.

[138] V. Samy and A. Kheddar, "Falls control using posture reshaping and active compliance," in *15th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 908–913, 2015.

[139] S. Yun, A. Goswami, and Y. Sakagami, "Safe fall: Humanoid robot fall direction change through intelligent stepping and inertia shaping," in *2009 IEEE Int. Conf. on Robotics and Automation*, pp. 781–787, 2009.

[140] U. Nagarajan and A. Goswami, "Generalized direction changing fall control of humanoid robots among multiple objects," in *2010 IEEE Int. Conf. on Robotics and Automation*, pp. 3316–3322, 2010.

[141] K. Mombaur, A. Truong, and J.-P. Laumond, "From human to humanoid locomotion – an inverse optimal control approach," *Autonomous Robots*, vol. 28, pp. 369–383, 2010.

[142] M. Cognetti, D. De Simone, F. Patota, N. Scianca, L. Lanari, and G. Oriolo, "Real-time pursuit-evasion with humanoid robots," in *2017 IEEE Int. Conf. on Robotics and Automation*, pp. 4090–4095, 2017.

[143] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa, "Dynamics and balance of a humanoid robot during manipulation tasks," *IEEE Trans. on Robotics*, vol. 22, pp. 568–575, June 2006.

[144] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, "Generation of whole-body optimal dynamic multi-contact motions," *Int. J. of Robotics Research*, vol. 32, no. 9-10, pp. 1104–1119, 2013.

[145] C. Mandery, J. Borràs, M. Jöchner, and T. Asfour, "Analyzing whole-body pose transitions in multi-contact motions," in *15th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 1020–1027, 2015.

[146] A. Werner, B. Henze, D. A. Rodriguez, J. Gabaret, O. Porges, and M. A. Roa, "Multi-contact planning and control for a torque-controlled humanoid robot," in *2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 5708–5715, 2016.

[147] D. De Simone, N. Scianca, P. Ferrari, L. Lanari, and G. Oriolo, "MPC-based humanoid pursuit-evasion in the presence of obstacles," in *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 5245–5250, 2017.

[148] S. Haddadin and E. Croft, "Physical human–robot interaction," in *Springer Handbook of Robotics*, pp. 1835–1874, Springer, 2016.

[149] A. De Luca and G. Oriolo, "Local incremental planning for nonholonomic mobile robots," in *1994 IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 104–110, 1994.

[150] T. Takenaka, T. Matsumoto, and T. Yoshiike, "Real time motion generation and control for biped robot -1st report: Walking gait pattern generation-," in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2009.

[151] A. Aboudonia, N. Scianca, D. De Simone, L. Lanari, and G. Oriolo, "Humanoid gait generation for walk-to locomotion using single-stage MPC," in *17th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 178–183, 2017.

[152] L. Lanari and S. Hutchinson, "Inversion-based gait generation for humanoid robots," in *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 637–642, 2015.