Sensor-based Whole-Body Planning/Replanning for Humanoid Robots

Paolo Ferrari, Marco Cognetti, Giuseppe Oriolo

Sensor-based Whole-Body Planning/Replanning for Humanoid Robots

Paolo Ferrari[1], Marco Cognetti[2], Giuseppe Oriolo[1]

***Abstract*— We propose a sensor-based motion planning/replanning method for a humanoid that must execute a task implicitly requiring locomotion. It is assumed that the environment is unknown and the robot is equipped with a depth sensor. The proposed approach hinges upon three modules that run concurrently: mapping, planning and execution. The mapping module is in charge of incrementally building a 3D environment map during the robot motion, based on the information provided by the depth sensor. The planning module computes future motions of the humanoid, taking into account the geometry of both the environment and the robot. To this end, it uses a 2-stages local motion planner consisting in a randomized CoM movement primitives-based algorithm that allows on-line replanning. Previously planned motions are performed through the execution module. The proposed approach is validated through simulations in V-REP for the humanoid robot NAO.**

I. Introduction

Researchers envision a future where humanoid robots are able to effectively assist or even replace humans in repetitive, tiring and dangerous activities. To achieve such objectives, these robots must be able to plan and execute whole-body motions that avoid collisions, maintain balance, and satisfy all existing kinematic/dynamic constraints. These requirements, together with the typical high-dimensionality of the planning space, make the motion planning problem for humanoid robots very challenging.

Over the last decade, many methods have been proposed to address the above problem in the case of known environment. They range from simple footstep planners (see [1] for a survey on existing techniques), that only generate isolated contacts with the ground, to full-fledged whole-body planners, that produce task-oriented motions, either relying on some reshaping of the final motion (e.g., [2]), or not easily generalizable to dynamic walking (e.g., [3]).

Differently from these works, in [4], we introduced a randomized motion planner that, hinging on the concept of CoM movement primitives (representative of typical humanoid actions, such as static walking, dynamic walking, and more), simultaneously generates foot placements and whole-body motions, without the need of any form of reshaping.

Most approaches work off-line, firstly producing a complete plan, whose computation may require high planning times, depending on the scenario complexity, forcing the robot to wait before being able to start moving. To address this problem, in [5], we have proposed an anytime algorithm that interleaves planning and execution intervals: a previously planned partial solution is executed, while a 2-stages local planner simultaneously plans a new solution for the subsequent execution interval. To ensure on-line performance, such local planner firstly builds a tree of CoM movement primitives sequences, performing lazy collision checks (in the spirit of [6] and [7]), and then searches the tree to validate a feasible, collision-free whole-body motion.

In this paper we propose a sensor-based whole-body planning framework for the case of unknown environments. This work extends [5], removing the assumption of a priori knowledge of the obstacles geometry. This is a non-trivial extension, since the planning and mapping capabilities are highly interconnected. In fact, in unknown environments, the ability of on-line building a map and accordingly planning collision-free motions is of fundamental importance to allow a robot to fulfill an assigned task. The contribution of this paper consists in a framework that allows a humanoid robot, equipped with a depth sensor, e.g., a Kinect, to incrementally build a map of its surroundings while executing previously planned motions, and simultaneously plan future motions, without any assumption about the environment.

In literature, some methods rely on the idea of monitoring the workspace with off-board, fixed sensors to provide a planner with the positions of the humanoid and obstacles (e.g., [8], [9]). Off-board sensing requires an appropriate setting that is difficult, or even impossible, in unstructured environments. Thus, on-board sensing is clearly preferable.

To reduce the computational complexity, allowing the robot to plan its motions on-line according to environmental information collected through on-board sensors while moving, the planning problem is often reduced to searching footstep sequences by detecting planar surfaces that define safe regions where the robot can step onto (as in [10], where binocular stereo-vision is used), or by representing the environment through a heightmap (using data provided by, for example, a monocular on-board camera as in [11], or a pivoting laser scanner mounted on the humanoid torso as in [12], [13]). Planning only at footsteps level implicitly assumes that collisions occurs only at robot soles, which is not the case for most real-world scenarios.

Only few methods have been proposed that take into account the 3D structure of an unknown environment. Most of them, in order to make the planning problem tractable on-line, rely on simplifying assumptions on either the environment or the robot geometry. For example, in [14] the environment is classified into areas of predefined different types, according to which the robot motions are chosen.

[1] Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Universita di Roma, via Ariosto 25, 00185 Roma, Italy. E-mail: {ferrari, oriolo}@diag.uniroma1.it.

[2] CNRS, Univ Rennes, Inria, IRISA, Rennes, France, E-mail: marco.cognetti@irisa.fr.

This work is supported by the EU H2020 COMANOID project.

In [15], instead, a robot bounding box is used to check collisions within a 3D occupancy grid. A more recent work, [16], proposes an approach integrating mapping, planning and localization. It maintains a 3D representation of the environment through an Octomap constructed from depth data, similarly to our work. Nevertheless, collision checks are performed using a floor projection of the 3D map and a circular robot model, impeding the humanoid of passing in narrow passages. The authors extended this work in [17], where they introduced the concept of inverse heightmaps precomputed for each possible robot action, for checking collisions within a 2.5D maps. This does not allow the robot to perform motions aimed at passing below obstacles.

Differently from these approaches, the sensor-based framework proposed in this paper does not rely on any form of simplification of either the environment or the robot geometry. Instead, to achieve the on-line performances, we leverage on the efficiency of an adaptation of the local motion planner proposed in [5] that checks collisions directly using an incrementally built 3D environment map, and considering the humanoid whole-body structure. Furthermore, our framework does not require any previous knowledge about obstacle positions and shapes.

The rest of the paper is organized as follows. We introduce the planning problem in Sect. II. The proposed approach is presented in Sect. III, hinged on a local motion planner discussed in Sect. IV. V-REP planning experiments for the NAO humanoid robot are illustrated in Sect. V. Conclusions and some future work are discussed in Sect. VI.

II. Problem Formulation

In the situation of interest, the humanoid is assigned a loco-manipulation task, i.e., a manipulation task that implicitly requires locomotion. In particular, the robot must bring a specified hand (hence, the *end-effector*) to a desired set-point, e.g., for grasping an object that is in general outside the workspace of the humanoid at its initial configuration. Furthermore, the robot acts in an unknown environment and it is equipped with a head-mounted depth camera, e.g., a Kinect, through which it can continuously acquire information about its surroundings while moving. In our framework, an environmental map is firstly created and, then, continuously updated when the sensor acquires new information. The proposed motion planner uses this map in order to generate task-oriented whole-body robot motions that are feasible (in the sense formally described below) in the so far explored area.

Throughout this paper, we assume that the environment is populated only by static obstacles and that the robot can perfectly localize itself with respect to a fixed inertial frame using an external module. Removing these two assumptions is part of our future work (see Sect. VI).

Let $\boldsymbol{q} = (\boldsymbol{q}_{\mathrm{CoM}}, \boldsymbol{q}_{\mathrm{jnt}})^T$ be the generic configuration of the humanoid. Here $\boldsymbol{q}_{\mathrm{CoM}} \in SE(3)$ is the world pose (position and orientation) of a reference frame attached to the Center of Mass (CoM) and $\boldsymbol{q}_{\mathrm{jnt}} \in \mathcal{C}_{\mathrm{jnt}}$ is the n-vector of joint angles. The assigned task must be fulfilled through a sequence of whole-body motions $\boldsymbol{q}(t)$, $t \in [t_{\mathrm{ini}}, t_{\mathrm{fin}}]$, constituted by a succession of on-line planned partial solutions, that satisfies four requirements:

- R1 The assigned set-point is reached at a finite time t_{fin}.
- R2 Collisions with workspace obstacles are avoided.
- R3 Position and velocity limits on the joints are respected.
- R4 The robot is in static or in dynamic equilibrium at all times.

In the following, we will call *feasible* the motions that satisfy requirements R2-R4.

III. Sensor-based Planning/Replanning Framework

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, computing on-line feasible whole-body motions, and sending commands to the humanoid actuators.

At the beginning, the mapping module generates a 3D map $\mathcal{M}(t_{\mathrm{ini}})$ according to information that the robot can acquire at its initial configuration $\boldsymbol{q}_{\mathrm{ini}}$. Then, an initial plan constituted by whole-body motions that are feasible within $\mathcal{M}(t_{\mathrm{ini}})$ 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. Then, an interleaved procedure (described below), in which the three modules run in parallel, starts. Hereafter, we refer to the time instant in which the robot starts moving as t_0.

During the interleaved procedure, the planning module works iteratively with the aim of computing future motions; meanwhile, previously planned motions are performed through the execution module, and the map $\mathcal{M}$ is continuously updated by the mapping module according to the newly acquired information. In particular, at any time instant t_c, the robot will be in the configuration $\boldsymbol{q}(t_c)$ as specified in the available, previously planned, sequence of feasible whole-body motions $\boldsymbol{q}(t)$, henceforth referred to as the *current plan*, with $t \in [t_0, \bar{t}]$. Thus, such plan will be composed by two portions: the committed part that the robot has already executed in the time interval $[t_0, t_c]$, and the remaining part that defines the future motions of the robot in the time interval $(t_c, \bar{t}]$.

The objective of each iteration of the planning module is to compute an extension of the current plan that starts at its final configuration $\bar{\boldsymbol{q}} = \boldsymbol{q}(\bar{t})$, at which the robot will be at the associated time instant $\bar{t}$, and is feasible according to newly acquired information. To extend the current plan before the robot reaches $\bar{\boldsymbol{q}}$, each iteration of the planning module must complete within (at maximum) the remaining time on the current plan. The idea is to repeatedly extend the current plan, each time considering an enlarged environment map, until it allows the robot to complete the assigned task.

In the following we describe separately the three modules.

A. 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. This feature is particularly relevant within our planning process, since it allows to produce task-oriented motions (for details, see Sect. IV).

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. More details about this powerful mapping framework can be found in [18], while the software is available as an open-source C++ library at http://octomap.github.io.

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

$$\mathcal{M}(t_{\text{ini}}) = \mathcal{R}(\boldsymbol{q}_{\text{ini}}) \cup \mathcal{M}_0(t_{\text{ini}}) \tag{1}$$

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.

B. 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 thanks to the continuously updated map provided by the mapping module described in Sec. III-A.

At the i-th generic iteration, the planning module invokes a specifically designed local motion planner (LMP), described in Sect. IV, that is in charge of producing a whole-body motion (henceforth referred to as a *local plan*) that starts at the final configuration $\bar{\boldsymbol{q}}_i$ of the current plan $\boldsymbol{q}(t)$, where $t \in [t_0, t_i]$, 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 t_i. Thus, the LMP is allowed to run for a limited time budget. While such time budget determines the duration of the i-th iteration of the planning module, the duration of the local plan to be computed is not specified, as it is autonomously determined by the LMP. In the following, we denote the duration of the i-th iteration of the planning module, i.e., the time budget given to the LMP at the i-th invocation, by $\Delta T_{P,i}$, and the duration of the local plan that it produces by $\Delta T_{E,i}$.

Let t_c be the time instant at the beginning of the i-th iteration of the planning module; at the corresponding invocation, the LMP is provided with:

- 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, that is computed as a portion of the remaining time on the current plan

$$\Delta T_{P,i} = \alpha_P (t_i - t_c) \tag{2}$$

  where $\alpha_P \in (0, 1)$ is a predefined value. 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.

The planning module is shown in Algorithm 1. 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 $\boldsymbol{q}(t)$, with $t \in [t_0 = t_{\text{ini}} + \overline{\Delta T}_P, t_1 = t_0 + \Delta T_{E,0}]$, with $\Delta T_{E,0}$ its duration, starting at the initial robot configuration $\boldsymbol{q}_{\text{ini}}$. This plan 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 computes the time budget $\Delta T_{P,i}$ devoted to the current invocation of the LMP according to (2) and retrieves the planning map $\mathcal{M}_{P,i}$ as the one currently provided by the mapping module. This invocation of the LMP produces a local plan $\boldsymbol{q}_i(t)$, $t \in [t_i, t_{i+1} = t_i + \Delta T_{E,i}]$, that starts at the final configuration $\bar{\boldsymbol{q}}_i$ of the current plan $\boldsymbol{q}(t)$ and is feasible within the map $\mathcal{M}_{P,i}$. The current plan is then extended by concatenating the computed local plan to it.

This iterative procedure terminates when the LMP computes a local plan such that the desired set-point is definitely reached, i.e., $\boldsymbol{f}(\bar{\boldsymbol{q}}_i) = \boldsymbol{y}_M^*$ with $\boldsymbol{f}(\bar{\boldsymbol{q}}_i)$ the end-effector

Algorithm 1: Planning Module

$\Delta T_{P,0} \leftarrow \overline{\Delta T}_P$;
$\mathcal{M}_{P,0} \leftarrow \mathcal{M}(t_c)$;
$\boldsymbol{q}(t),\, t \in [t_0, t_1] \leftarrow$ LMP($\boldsymbol{q}_{\rm ini}$, $\Delta T_{P,0}$, $\mathcal{M}_{P,0}$);
wait for $\alpha_P \Delta T_{E,0}$;
extract $\bar{\boldsymbol{q}}_1$ from $\boldsymbol{q}(t)$;
$i \leftarrow 1$;
while $\boldsymbol{f}(\bar{\boldsymbol{q}}_i) \neq \boldsymbol{y}_M^*$ **do**
  compute time budget $\Delta T_{P,i}$ according to (2);
  $\mathcal{M}_{P,i} \leftarrow \mathcal{M}(t_c)$;
  $\boldsymbol{q}_i(t),\, t \in [t_i, t_{i+1}] \leftarrow$ LMP($\bar{\boldsymbol{q}}_i$, $\Delta T_{P,i}$, $\mathcal{M}_{P,i}$);
  $\boldsymbol{q}(t),\, t \in [t_0, t_{i+1}] \leftarrow$ concatenate $\boldsymbol{q}(t),\, t \in [t_0, t_i]$, and $\boldsymbol{q}_i(t),\, t \in [t_i, t_{i+1}]$);
  $i \leftarrow i + 1$;
  extract $\bar{\boldsymbol{q}}_i$ from $\boldsymbol{q}(t)$;
end

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.

C. 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, 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 $\dot{q}_p$ is computed on-line in such a way to make the robot looking in the direction of the location that it will reach at the end of the current plan.

At each time instant, the robot yaw neck joint velocity $\dot{q}_p$ is computed using a simple proportional control

$$\dot{q}_p = K_p(q_p^d - q_p) \tag{3}$$

where q_p^d and q_p are, respectively, the desired and current yaw joint positions. Given the final configuration $\bar{\boldsymbol{q}}$ on the current plan and the configuration $\boldsymbol{q}$ specified for the current time instant, q_p^d is defined as the angle between the robot sagittal axis at $\boldsymbol{q}$ (that can be easily identified through the subvector $\boldsymbol{q}_{\rm CoM}$), and the line joining the origins of the CoM frames at $\boldsymbol{q}$ and $\bar{\boldsymbol{q}}$. K_p is a positive scalar gain.

Note that, when the current plan is extended, its final configuration $\bar{\boldsymbol{q}}$ changes, and consequently also q_p^d.

IV. Local Motion Planner (LMP)

In this section, we describe the local motion planner (LMP) that is invoked at each iteration of the planning module. It consists in an adaptation, for the case of unknown environments, of the LMP presented in [5]. Within this planner, humanoid whole-body motions are produced as concatenations of precomputed *CoM movements primitives*, as firstly introduced in [4]. The LMP is provided with a catalogue U of N primitives, each one representing an elementary humanoid motion. In general, U will include

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

split the time budget ΔT_P in two intervals ΔT_P^L and ΔT_P^V;
$\mathcal{T} \leftarrow$ LazyStage($\bar{\boldsymbol{q}}$, ΔT_P^L, $\mathcal{M}_P$);
$\boldsymbol{q}(t) \leftarrow$ ValidationStage($\mathcal{T}$, ΔT_P^V, $\mathcal{M}_P$);
return $\boldsymbol{q}(t)$;

basic walking movements, e.g., static and dynamic steps, and possibly more complex ones, e.g., crauching or crawling. A pure manipulation primitive, namely the `free_CoM`, is also included in U. Each primitive, henceforth indicated as $\boldsymbol{u}_{\rm CoM}$, has an associated duration, and specifies reference CoM and swing foot trajectories for the whole duration[1].

The LMP, whose pseudocode is provided in Procedure 1, is composed of two consecutive stages, namely *lazy* and *validation* stages, amoung which the time budget ΔT_P assigned by the planning module to the LMP is split as

$$\Delta T_P^L = \alpha_{LMP} \Delta T_P, \tag{4}$$
$$\Delta T_P^V = (1 - \alpha_{LMP}) \Delta T_P, \tag{5}$$

where $\alpha_{LMP} \in (0, 1)$, and serves as a design parameter.

The *lazy* stage (see Procedure 2) is a RRT-based algorithm that builds a tree $\mathcal{T}$ in configuration-time space. To speed up such tree construction, the algorithm performs collision checks lazily, i.e., only at vertexes level by means of a simplified occupancy volume of the robot. In the tree $\mathcal{T}$:

- A vertex $v = (\boldsymbol{q}, t)$ consists of a configuration $\boldsymbol{q}$, and its associated time instant t.
- An edge represents a feasible whole-body motion connecting two adjacent vertexes, produced through a certain CoM primitive $\boldsymbol{u}_{\rm CoM}$.
- A branch of the tree provides a *candidate local plan* if its ending vertex (i.e., the leaf) is such that one of the these conditions holds: (i) the simplified occupancy volume of the robot at the corresponding configuration $\boldsymbol{q}$ does not completely lie within the map $\mathcal{M}_P$; (ii) it has been generated through the `free_CoM` primitive, i.e., it potentially allows to finalize the assigned task.

The tree $\mathcal{T}$ is rooted at $v_0 = (\bar{\boldsymbol{q}}, \bar{t})$, where $\bar{\boldsymbol{q}}$ is the final configuration on the current plan, and $\bar{t}$ its associated time instant. At the generic iteration, the algorithm picks a random sample point $\boldsymbol{y}_{\rm rand}$ in the task space, and retrieves its nearest vertex $v_{\rm near} = (\boldsymbol{q}_{\rm near}, t_k)$[2] in $\mathcal{T}$ using a metric $d(\cdot, \boldsymbol{y}_{\rm rand})$[3]. From the catalogue U a primitive $\boldsymbol{u}_{\rm CoM}^k$ is randomly selected. Let T_k be its duration. The reference CoM and swing foot trajectories specified by $\boldsymbol{u}_{\rm CoM}^k$ determine the reference poses of, respectively, the CoM frame $\boldsymbol{q}_{\rm CoM}^{\rm new}$ and the (new) support foot $\boldsymbol{q}_{\rm sup}^{\rm new}$ (that we do not explicitly include in the vertex

[1] This is true for all primitives, except for `free_CoM`, as it does not provide any reference CoM trajectory, leaving it to freely move while maintaning both feet fixed on the ground.

[2] Selection of ending vertexes of already existing candidate local plans is carefully avoided as their expansion does not produce any new local plan.

[3] In our implementation, the metric $d(\boldsymbol{q}, \boldsymbol{y})$, that measures the distance between a configuration $\boldsymbol{q}$ and a point $\boldsymbol{y}$ in the task space, is defined as the Euclidean distance between the ground projections of the robot CoM positions at $\boldsymbol{q}$ and $\boldsymbol{y}$.

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

root the tree $\mathcal{T}$ at $v_0 = (\bar{\boldsymbol{q}}, \bar{t})$;
$\mathcal{V}' \leftarrow \{v_0\}$;
$t_e \leftarrow 0$;
repeat
  pick a random sample point $\boldsymbol{y}_{\text{rand}}$ in the task space;
  select the nearest vertex v_{near} in $\mathcal{T}$ to $\boldsymbol{y}_{\text{rand}}$ according to $d(\cdot, \boldsymbol{y}_{\text{rand}})$;
  select from U a random CoM primitive $\boldsymbol{u}_{\text{CoM}}^k$;
  compute $\boldsymbol{q}_{\text{CoM}}^{\text{new}}$ according to $\boldsymbol{q}_{\text{CoM}}^{\text{near}}$ and $\boldsymbol{u}_{\text{CoM}}^k$;
  if $\mathcal{S}(\boldsymbol{q}_{\text{new}})$ is collision$-$free **then**
    $v_{\text{new}} \leftarrow ((\boldsymbol{q}_{\text{CoM}}^{\text{new}}, \emptyset)^T, t_{k+1} = t_k + T_k)$;
    add vertex v_{new} in $\mathcal{T}$ as a child of v_{near};
  end
  $t_e \leftarrow$ get the elapsed time;
until $t_e \geq \Delta T_P^L$;
return $\mathcal{T}$;

for sake of illustration) at the time instant $t_{k+1} = t_k + T_k$, i.e., after applying $\boldsymbol{u}_{\text{CoM}}^k$ to $\boldsymbol{q}_{\text{near}}$. At this point a collision check is performed within the map $\mathcal{M}_p$ using a simplified occupancy volume of the robot $\mathcal{S}(\boldsymbol{q}_{\text{new}})$, with $\boldsymbol{q}_{\text{new}} = (\boldsymbol{q}_{\text{CoM}}^{\text{new}}, \emptyset)^T$ (the subvector $\boldsymbol{q}_{\text{jnt}}$ is left undefined). In our implementation, we check collisions at footsteps level using the pose $\boldsymbol{q}_{\text{sup}}^{\text{new}}$. Other choises, like conservative bounding boxes with pose $\boldsymbol{q}_{\text{CoM}}^{\text{new}}$, are also possible. If $\mathcal{S}(\boldsymbol{q}_{\text{new}})$ results collision-free, a new vertex $v_{\text{new}} = (\boldsymbol{q}_{\text{new}}, t_{k+1})$ is added to $\mathcal{T}$ as a child of v_{near}. This iterative procedure stops when the time budget ΔT_P^L assigned to the lazy stage runs out.

At this point, the *validation* stage (see Procedure 3) is run. With the aim of further approaching the desired set-point, and possibly completing the task, the candidate local plan, among the ones resulting from $\mathcal{T}$, whose ending vertex is the closest to $\boldsymbol{y}_M^*$ in terms of the metric $d(\cdot, \boldsymbol{y}_M^*)$ described above, is chosen for a validation attempt.

Let $p^* = \{v_0, \ldots, v_{K-1}\}$ be the chosen candidate local plan, with K the number of vertexes along the corresponding branch. The validation attempt consists in generating the whole-body motion between each pair of consecutive vertexes, and checking its feasibility w.r.t. requirements R2-R4.

This procedure proceeds iteratively. At the k-th iteration, the configuration $\boldsymbol{q}_k = (\boldsymbol{q}_{\text{CoM}}^k, \boldsymbol{q}_{\text{jnt}}^k)^T$, and the associated time instant t_k, are extracted from the vertex v_k. If the sub-vector $\boldsymbol{q}_{\text{jnt}}^k$ is undefined, a motion generator is called to produce the whole-body motion that connects $\boldsymbol{q}_{k-1}$ to $\boldsymbol{q}_k$ through the primitive $\boldsymbol{u}_{\text{CoM}}^{k-1}$ (such information may be stored with the edge at the time of its creation during the lazy stage), i.e., the joint trajectory that realizes the specified CoM and swing foot trajectories in the time interval $[t_{k-1}, t_k = t_{k-1} + T_{k-1}]$, with T_{k-1} the duration of $\boldsymbol{u}_{\text{CoM}}^{k-1}$, and possibly reaches the desired set-point $\boldsymbol{y}_M^*$. To this end, the motion generator works by integrating the joint velocities produced by a priorities-based kinematic control law in which the locomotion and manipulation tasks are, respectively, the primary and secondary tasks. The generated joint trajectories are continuously checked for collisions within the map $\mathcal{M}_P$ using the actual occupancy volume $\mathcal{R}(\boldsymbol{q})$ of the robot

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

select the best local plan p^* in $\mathcal{T}$;
$t_e \leftarrow 0$;
while $p^* \neq \emptyset$ **and** $t_e \leq \Delta T_P^V$ **do**
  $\boldsymbol{q}(t) \leftarrow \emptyset$;
  $k \leftarrow 1$;
  while $k < K$ **and** $\boldsymbol{q}_{k-1} \neq (\emptyset, \emptyset)^T$ **do**
    extract $\boldsymbol{q}_{\text{jnt}}^k$ from v_k;
    if $\boldsymbol{q}_{\text{jnt}}^k \neq \emptyset$ **then**
      $[\boldsymbol{q}_k, \overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}] \leftarrow$ MotionGeneration(v_{k-1}, v_k);
      if $\boldsymbol{q}_k \neq (\emptyset, \emptyset)^T$ **then**
        if $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k} \subset \mathcal{M}_P$ **then**
          update vertex v_k with $\boldsymbol{q}_k$ and add edge $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}$ to $\mathcal{T}$;
          $\boldsymbol{q}(t) \leftarrow$ append $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}$ to $\boldsymbol{q}(t)$;
        else
          return $\boldsymbol{q}(t)$;
        end
      else
        remove subtree $\mathcal{T}_k$ rooted at v_k from $\mathcal{T}$;
      end
    end
    $k \leftarrow k + 1$;
  end
  select the best local plan p^* in $\mathcal{T}$;
  $t_e \leftarrow$ get the elapsed time;
end
return $\emptyset$;

(requirement R2), and for violation of position/velocity joint limits (requirement R3). Static equilibrium is also checked in case $\boldsymbol{u}_{\text{CoM}}^{k-1}$ is `free_CoM`, while for the other primitives, static or dynamic equilibrium is guaranteed by construction (requirement R4). More details about the motion generator can be found in [4], [5].

In case of no violations in the motion generator, the validation stage proceeds or stops depending on whether the produced whole-body motion $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}$ is entirely contained in the map $\mathcal{M}_P$, i.e., the actual occupancy volume of the robot $\mathcal{R}(\boldsymbol{q})$ at each configuration $\boldsymbol{q}$ of the latter completely lies in $\mathcal{M}_P$ (in this case, $\overline{\boldsymbol{q}_{k-1}\boldsymbol{q}_k}$ is concatened to the portion of the candidate local plan that has been validated so far), or not (in this case, the validated local plan is returned).

In case of violations in the motion generator, the tree $\mathcal{T}$ is pruned of the subtree $\mathcal{T}_k$ rooted at v_k, and a new candidate local plan is selected for another validation attempt.

V. Planning Experiments

The proposed sensor-based framework has been implemented in V-REP on an Intel Core i7 running at 2.70 GHz. The chosen robotic platform is the NAO humanoid robot, and the right hand is employed as end-effector. 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 $U = \{\texttt{free\_CoM} \cup U_{\text{CoM}}^D\}$. Here, `free_CoM` is a *non-stepping* primitive that allows the CoM to move freely, as long as both feet remain fixed; U_{CoM}^D is a subset of *dynamic* steps extracted from

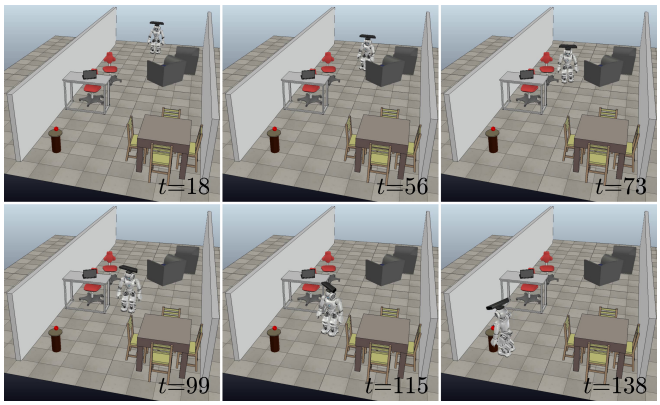

Fig. 1. Planning scenario 1: snapshots from a solution.

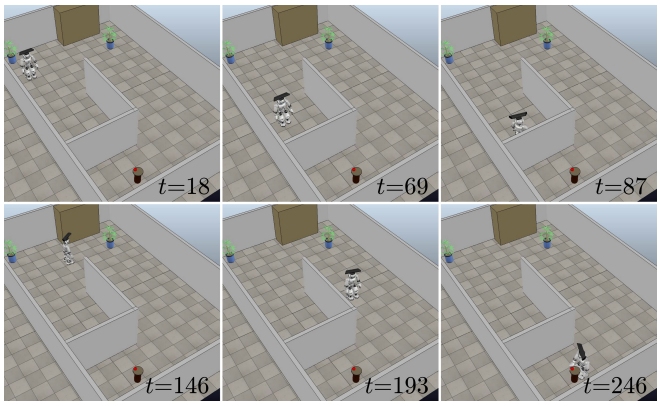

Fig. 2. Planning scenario 2: snapshots from a solution.

various types of ZMP-based gaits precomputed by the intrinsically stable MPC framework presented in [19]. In particular, U^D_{CoM} includes forward, backward, curved and diagonal steps. In our simulations, we activate the manipulation task only when the robot hand is inside a spherical region centered at the desired set-point $\boldsymbol{y}^*_M$, where we allow the choice of the `free_CoM` primitive[4].

We consider two planning scenarios. In both of them, the robot is assigned the task of grasping a ball placed on a table that is outside its initial workspace. The map $\mathcal{M}_0(t_{\mathrm{ini}})$ consists of an initial knowledge provided in advance within a cylindrical area of radius 0.5 m and height 1.2 m centered at $\boldsymbol{q}_{\mathrm{CoM}}(t_{\mathrm{ini}})$, 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 plan within this map through the first invocation of the LMP, with a predefined time budget of $\overline{\Delta T}_P = 15$ s. The parameters α_P and α_{LMP} are respectively set to 0.5 and 0.6.

In the first scenario (see Fig.1), obstacles of different kinds (chairs, tables and sofas) obstruct the path between the robot at the initial configuration, at which only portions of few obstacles can be seen, and the destination. Once the initial partial plan is available, the robot starts to perform it through the execution module, while the planning module computes future motions. At each replanning, 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, the LMP is able to produce motions allowing the robot to pass, e.g., in the strict free space between the sofa and the chair (third snapshot). Note that, due to the complex shape of the chair, approaches using bounding boxes (e.g., [15]) or 2D projections of swept volumes (e.g., [17]) to check collisions would fail to find a feasible plan in this case.

In the second scenario (see Fig. 2), 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 camera field of view. Among all the candidate local plans bringing towards unexplored areas, the 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, the LMP generates an extension of the current plan that allows the robot to exit the closed space and to proceed towards the free boundary of the map (third and fourth snapshots). We emphasize that such effective behaviour is the result of the intrinsic bias towards unexplored areas of the proposed planner. 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 by pure manipulation.

Table I collects, for the three scenarios, some performance

i	time budget (s)	# lazy plans	motion duration (s)
Experiment 1			
0	15	19	52.45
1	13.11	29	17.65
2	15.38	5	11.65
3	13.52	55	27.5
Experiment 2			
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 I

PLANNER PERFORMANCE DATA.

[4]Planning smoother reaching motions is out of the scope of this paper, although this can be easily accounted in the proposed framework by involving the strategy proposed in [20].

data of the planning module. The experiments have required, respectively, four, and ten invocations of the LMP. For each invocation, the table reports: the assigned time budget, the number of candidate plans produced by the lazy stage, and the duration of the generated extension of the current plan. Note that, in the second scenario, in two cases the 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 computed as in eq. (2) within a new map that has been updated during the robot motion.

We encourage the reader to see the accompanying video to better appreciate the effectiveness of the generated motions.

VI. Conclusions

In this paper, we proposed a sensor-based framework for humanoids aiming at fulfilling tasks that implicitly require locomotion. The planner is hinged on three cooperating, highly interconnected modules: mapping, planning and execution modules. The mapping module is in charge of building and updating a 3D environment map during the robot motion. While the execution module sends the commands to the humanoid actuators based on previously planned whole-body motions, the planning module on-line computes future plans using a local motion planner that takes into account newly acquired information. Its 2-stages design allows for rapid replanning and proved to be particularly suitable for on-line application. V-REP simulations with the NAO humanoid have shown the ability of the proposed framework to generate fluid robot motions, without the need for the robot to stop and plan. Furthermore, thanks to the fact that our framework considers the 3D structure of both the environment and the robot, narrow passages are correctly negotiated.

Our current research focuses on removing the static environment and perfect robot localization assumptions mentioned in Sect. II by, respectively, equipping the robot with the possibility of replanning also when a moving obstacle invalidates the current plan, and substituting the mapping module with a full-fledged SLAM module, providing the robot with full autonomy. Other possible extensions of this work will include the design of strategies for on-line generating movement primitives according to the planning needs, e.g., for managing uneven grounds and tasks that require mobile manipulation of heavy objects.

References

[1] N. Perrin, "Biped footstep planning," in *Humanoid Robotics: A Reference*. P. Vadakkepat and A. Goswami, Eds. Springer Netherlands, 2018.

[2] 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," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1089–1103, 2013.

[3] K. Bouyarmane and A. Kheddar, "Humanoid robot locomotion and manipulation step planning," *Advanced Robotics*, vol. 26, no. 10, pp. 1099–1126, 2012.

[4] M. Cognetti, P. Mohammadi, and G. Oriolo, "Whole-body motion planning for humanoids based on com movement primitives," in *2015 IEEE-RAS Int. Conf. on Humanoid Robots*, 2015, pp. 1090–1095.

[5] P. Ferrari, M. Cognetti, and G. Oriolo, "Anytime whole-body planning/replanning for humanoid robots," in *2018 IEEE-RAS Int. Conf. on Humanoid Robots*, 2018, pp. 1–9.

[6] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *2000 IEEE Int. Conf. on Robotics and Automation*, vol. 1, 2000, pp. 521–528.

[7] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *The International Journal of Robotics Research*, pp. 403–417, 2003.

[8] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade, "Vision-guided humanoid footstep planning for dynamic environments," in *2005 IEEE-RAS Int. Conf. on Humanoid Robots*, 2005, pp. 13–18.

[9] L. Baudouin, N. Perrin, T. Moulard, F. Lamiraux, O. Stasse, and E. Yoshida, "Real-time replanning using 3d environment for humanoid robot," in *2011 IEEE-RAS Int. Conf. on Humanoid Robots*, 2011, pp. 584–589.

[10] 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*, 2005, pp. 623–628.

[11] 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*, 2007, pp. 463–469.

[12] 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*, 2009, pp. 3543–3548.

[13] 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.

[14] J.-S. Gutmann, M. Fukuchi, and M. Fujita, "3d perception and environment map generation for humanoid robot navigation," *Int. J. of Robotics Research*, vol. 27, no. 10, pp. 1117–1134, 2008.

[15] A. Nakhaei and F. Lamiraux, "Motion planning for humanoid robots in environments modeled by vision," in *2008 IEEE-RAS Int. Conf. on Humanoid Robots*, 2008, pp. 197–204.

[16] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in *2012 IEEE-RAS Int. Conf. on Humanoid Robots*, 2012, pp. 692–697.

[17] 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*, 2013, pp. 2658–2664.

[18] 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.

[19] N. Scianca, M. Cognetti, D. De Simone, L. Lanari, and G. Oriolo, "Intrinsically stable MPC for humanoid gait generation," in *2016 IEEE-RAS Int. Conf. on Humanoid Robots*, 2016, pp. 101–108.

[20] P. Ferrari, M. Cognetti, and G. Oriolo, "Humanoid whole-body planning for loco-manipulation tasks," in *2017 IEEE Int. Conf. on Robotics and Automation*, 2017, pp. 4741–4746.