2016 IEEE International Conference on Robotics and Automation (ICRA)
Stockholm, Sweden, May 16-21, 2016

Whole-Body Planning for Humanoids along Deformable Tasks

Marco Cognetti, Valentino Fioretti, Giuseppe Oriolo

Abstract— **This paper addresses the problem of generating whole-body motions for a humanoid robot that must execute a certain task in an environment containing obstacles. The assigned task trajectory is deformable, and the planner may exploit this feature for finding a solution. Our framework consists of two main components: a constrained motion planner and a deformation mechanism. The basic idea is that the constrained motion planner attempts to solve the problem for the original task. If this proves to be too difficult, the deformation mechanism modifies the task using appropriate heuristic functions. Then, the constrained motion planner is invoked again on the deformed task. If needed, this procedure is iterated. The proposed algorithm has been successfully implemented for the NAO humanoid in V-REP.**

I. Introduction

In recent years, humanoid robots have been the subject of constantly increasing attention. While earlier work was mostly aimed at obtaining stable and efficient walking gaits, many researchers are now looking at the generation of task-oriented, whole-body motions for humanoid robots.

Motion planning for humanoids is particularly challenging for various reasons. First, the planning space is high-dimensional because these robots possess many degrees of freedom. Second, a humanoid robot is not a free-flying system in its configuration space. The various kinematic and dynamic constraints of the specific robot must also be taken into account, as does the necessity of maintaining equilibrium (either static or dynamic) at all times. In view of this all complexity, the problem is usually addressed by making simplifying assumptions on either the robot geometry or the environment [1-7].

Assigning a task to the robot further increases the planning difficulty. The task may be a single action (e.g., 'grasp this object') or a sequence of navigation and manipulation actions ('take that object and bring it in the other room').

A common approach for generating task-constrained movements is kinematic control. One popular example is the task-priority method [8], extended to handle inequality constraints in [9]. This framework was used in [10] for computing footsteps of humanoids that must execute manipulation tasks. Despite its efficiency, kinematic control by itself remains a purely local technique, well suited for reactive behaviors but in general outperformed by full-fledged planning techniques.

Three main approaches can be found in the literature for task-oriented humanoid planning: (i) separating locomotion from task execution [11], [12]; (ii) planning statically stable collision-free trajectories, later converted to dynamically stable motions [13]; (iii) achieve acyclic locomotion and task execution through whole-body contact planning [14].

A fundamentally different approach, which we introduced in [15], is to solve the task-constrained planning problem without separating locomotion from whole-body motion generation. Our method hinges on the concept of CoM movement primitives, defined as precomputed trajectories of the CoM that are associated to specific actions. The proposed planner builds a tree in configuration space by concatenating feasible, collision-free whole-body motions that realize a succession of CoM movements and, at the same time, the assigned task. Results showed that the planner was able to generate sensible plans for a variety of composite tasks requiring a combination of navigation and manipulation.

In [15], it was assumed that the assigned task is not modifiable. While this may sometimes be the case, it is often just a working assumption that does not represent an actual requirement. For example, suppose we want the humanoid to move an object from a location to another: the actual trajectory of the object may not be relevant, provided that it is reasonably efficient (e.g., it does not wander too much). Typically, the user may specify[1] a very simple tentative task trajectory (e.g., a line segment joining the initial and the final task point), which can be deformed by the planner if this is convenient for finding a solution.

Task path/trajectory deformation was considered in [16], via length and clearance optimization, and in [17], where multiple trajectories are maintained and deformed in real-time using a genetic algorithm. The approach of [18], [19] combines motion planning with reactive behaviors to generate task trajectories for mobile manipulators. Functional gradient techniques are used in [20] to improve the quality of an initial trajectory. However, none of these works addresses the specific problems posed by humanoid robots.

The method proposed in this paper consists of two main components: a constrained motion planner and a deformation mechanism. The basic idea is that the constrained motion planner attempts to solve the problem for the original task. If this proves to be too difficult, a deformation mechanism is triggered that modifies the task path using simple heuristic

The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, via Ariosto 25, 00185 Roma, Italy. E-mail: {cognetti, oriolo}@diag.uniroma1.it. This work is supported by the EU FP7 COMANOID project.

[1]The reader may argue that no trajectory at all is needed in this situation, as one may simply specify the final task value as a set-point. However, it should be realized that even in this case the use of a pseudoinverse-based kinematic control method (by itself or within a planner) implicitly determines which trajectory will be followed in task space. For example, it is easy to verify that using diagonal gain matrices for error feedback yields a rectilinear trajectory to the set-point in task space.

978-1-4673-8026-3/16/$31.00 ©2016 IEEE

functions, and the constrained motion planner is invoked on the deformed task. If needed, the procedure is iterated.

The paper is organized as follows. In the next section, the planning problem is formulated. A quick overview of the planner is given in Sect. III. The constrained motion planner is described in Sect. IV, while the deformation mechanism is presented in Sect. V. Motion planning experiments for the NAO humanoid robot are presented in Sect. VI. Future research directions are briefly discussed in Sect. VII.

II. Problem Formulation

Before formulating the planning problem, we recall the motion model of [15] for generating humanoid motion.

A. Humanoid Motion Model

A configuration $\boldsymbol{q}$ of a free-flying humanoid can be identified by its joint n-vector $\boldsymbol{q}_{\rm jnt} \in \mathcal{C}_{\rm jnt}$ and the pose (position plus orientation) $\boldsymbol{q}_{\rm CoM} \in SE(3)$ of a reference frame attached to the robot Center of Mass (CoM):

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}_{\rm CoM} \\ \boldsymbol{q}_{\rm jnt} \end{pmatrix}.$$

In our planner, CoM motions will be generated by patching CoM subtrajectories taken from a catalogue of *CoM movement primitives*, that represent typical humanoid motions, e.g. static steps, dynamic steps, crouching, and so on. Each primitive has a certain time duration and may specify a trajectory for other points of the robot, in addition to the CoM; for example, a stepping primitive will include a trajectory for the swing foot. Once a CoM movement primitive has been selected, the joint trajectory can still be chosen by the planner among the infinite whole-body motions compatible with the primitive, in such a way that other requirements are met (task execution, obstacle avoidance, etc).

The above approach to motion generation is reflected in the hybrid (partly algebraic, partly differential) model

$$\boldsymbol{q}_{\rm CoM}(t) = \boldsymbol{q}_{\rm CoM}(t_k) + \boldsymbol{A}(\boldsymbol{q}_{\rm CoM}(t_k))\, \boldsymbol{u}_{\rm CoM}(t_k) \quad (1)$$

$$\dot{\boldsymbol{q}}_{\rm jnt}(t) = \boldsymbol{v}_{\rm jnt}(t) \quad (2)$$

describing the robot motion in an interval $[t_k, t_{k+1} = t_k + T_k]$ in which the CoM is performing a primitive movement of duration T_k. Here, $\boldsymbol{q}_{\rm CoM}(t_k)$ is the CoM frame pose at the start of the interval; $\boldsymbol{A}(\boldsymbol{q}_{\rm CoM}(t_k))$ is the transformation matrix from the CoM frame at t_k to the world frame; $\boldsymbol{u}^k_{\rm CoM}(t)$ is the pose displacement of the CoM frame at t, relative to its pose at t_k; $\boldsymbol{v}_{\rm jnt}$ is the vector of joint velocities.

Note that in (1–2) the CoM displacement and the joint velocity profile are not independent: as explained in detail in Sect. IV, $\boldsymbol{v}_{\rm jnt}|_{[t_k,t)}$ is chosen so as to realize the CoM movement primitive $\boldsymbol{u}^k_{\rm CoM}(t)$.

B. Task-Oriented Planning

We consider tasks defined as trajectories for the position (and possibly the orientation) of a specific point (body) of the humanoid: e.g., a manipulation task may be a trajectory for one hand of the robot, while a navigation task may be a trajectory for the midpoint between its feet. Denoting task coordinates by $\boldsymbol{y}$, a forward kinematic map relates them to generalized coordinates

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}(\boldsymbol{q}_{\rm CoM}, \boldsymbol{q}_{\rm jnt}).$$

An initial task trajectory $\boldsymbol{y}^{[0]}(t)$, $t \in [t_i, t_f]$, is assigned[2] composed by a geometric path $\boldsymbol{y}^{[0]}(s), s \in [s_i, s_f]$, and a time history $s^{[0]}(t)$, $t \in [t_i, t_f]$, with $s_i = s(t_i)$, $s_f = s(t_f)$. In particular, the geometric path $\boldsymbol{y}^{[0]}$ is assumed to be a *deformable* curve in task space: this means that the endpoints $\boldsymbol{y}^{[0]}(s_i)$ and $\boldsymbol{y}^{[0]}(s_f)$ are fixed, but the actual shape of the curve may be changed if necessary by acting on certain parameters. For illustration, in the rest of the paper we will consider *B-splines* as deformable curves, and the action parameters will be *control points*. The planner is allowed to change the shape of the task path (and adapt the time history), if this is deemed necessary to find a solution.

The considered planning problem consists in finding a feasible whole-body motion of the humanoid that realizes the task trajectory, possibly deformed, while avoiding collisions and satisfying kinematic constraints. More formally, a solution to our problem consists of:

1) A final task trajectory $\boldsymbol{y}^*(t)$, $t \in [t_i, t_f]$, consisting of a geometric path $\boldsymbol{y}^*(s), s \in [s_i, s_f]$, obtained by (repeated) deformation of $\boldsymbol{y}^{[0]}(s)$, and an associated time history $s^*(t), t \in [t_i, t_f]$.
2) A whole-body motion of the humanoid $\boldsymbol{q}^*(t)$, $[t_i, t_f]$, that satisfies the following requirements:
   - R1 The final task trajectory is realized:
     $$\boldsymbol{f}(\boldsymbol{q}^*(t)) = \boldsymbol{y}^*(s^*(t)) = \boldsymbol{y}^*(t), \quad t \in [t_i, t_f].$$
   - R2 The robot maintains static or dynamic equilibrium.
   - R3 All collisions with obstacles are avoided.
   - R4 Joint limits and velocity bounds, respectively expressed in the form $\boldsymbol{q}_{\rm jnt,m} < \boldsymbol{q}_{\rm jnt} < \boldsymbol{q}_{\rm jnt,M}$ and $\boldsymbol{v}_{\rm jnt,m} < \boldsymbol{v}_{\rm jnt} < \boldsymbol{v}_{\rm jnt,M}$, are satisfied.

The above formulation includes directly two special cases of practical interest:

- *Set-point task*: When the task reduces to a desired set-point $\boldsymbol{y}(t_f)$ (e.g., bring the humanoid hand to a certain placement), the initial task path $\boldsymbol{y}^{[0]}(s)$ may be chosen as any deformable curve (e.g., a line) joining the starting task position $\boldsymbol{y}(t_i) = \boldsymbol{f}(\boldsymbol{q}(t_i))$ with the final task position $\boldsymbol{y}(t_f)$. The time history may be simply set as $s^{[0]}(t) = t$, $\forall t \in [t_i, t_f]$. See also footnote 1.
- *Partially deformable task*: In some problems, it may be desiderable to allow deformation only on a subset of task components. For example, consider a manipulation problem in which the robot has to carry a glass containing some liquid. The glass should be kept vertical, so as not to spill the liquid. A deformation can therefore be applied only to the position components of the task (i.e., to the Cartesian trajectory of the glass), whereas the orientation components should not be affected.

[2]Since we are addressing a planning problem, it will be assumed that the initial configuration $\boldsymbol{q}(t_i)$ is assigned, and that the corresponding task value matches the starting point of the trajectory, i.e., $\boldsymbol{y}^{[0]}(t_i) = \boldsymbol{f}(\boldsymbol{q}(t_i))$.

Algorithm 1: Planner

1 sol_found ← **false**; $i \leftarrow 0$;
2 get the initial task path $\boldsymbol{y}^{[0]}$ and time history $s^{[0]}$;
3 **repeat**
4 build the current task trajectory $\boldsymbol{y}^{[i]} \leftarrow \boldsymbol{y}^{[i]}(s^{[i]})$;
5 $[\mathcal{T}, \tilde{\boldsymbol{y}}] \leftarrow$ ConstrainedMotionPlanner($\boldsymbol{y}^{[i]}$);
6 **if** $\tilde{\boldsymbol{y}} = \boldsymbol{y}^{[i]}(t_f)$ **then**
7 sol_found ← **true**;
8 $\boldsymbol{y}^* \leftarrow \boldsymbol{y}^{[i]}$; $s^* \leftarrow s^{[i]}$;
9 **else**
10 $[\boldsymbol{y}^{[i+1]}, s^{[i+1]}] \leftarrow$ TaskDeformation($\boldsymbol{y}^{[i]}$, $s^{[i]}$, $\mathcal{T}$, $\tilde{\boldsymbol{y}}$);
11 **end**
12 $i \leftarrow i + 1$;
13 **until** sol_found = **true** **or** i = MAX_DEFORM;

III. Planner Overview

The underlying principle of the proposed planner, whose pseudocode[3] is given in Algorithm 1, is simple.

First, we invoke a constrained motion planner that expands a tree $\mathcal{T}$ in configuration space in the attempt to find a whole-body motion realizing the initial task trajectory $\boldsymbol{y}^{[0]}$ and satisfying requirements R2-R4 (line 5). If this proves to be too difficult, the planner returns a partial solution whose *limit task point* $\tilde{\boldsymbol{y}}$ (defined as the furthermost task point realized by configurations contained in the tree $\mathcal{T}$) is different from $\boldsymbol{y}(t_f)$. In this case, the initial task path is modified using deformation actions which take place at the limit task point (lines 9-10), and the constrained motion planner is invoked again. This deformation-planning cycle is repeated until a solution is found.

In the following sections we describe in detail the two main components of our algorithm, i.e., the constrained motion planner and the deformation mechanism.

IV. Constrained Motion Planner

The constrained motion planner, whose pseudocode is shown in Algorithm 2, is essentially the same of [15]. The task trajectory to be realized is denoted by $\boldsymbol{y}^d$; in the i-th iteration of Algorithm 1, the planner is invoked with $\boldsymbol{y}^d$ set to $\boldsymbol{y}^{[i]}$. The goal is to find a whole-body motion that realizes $\boldsymbol{y}^d$ and is *feasible*, i.e., satisfies requirements R2-R4.

The planner works in an iterative fashion and builds a tree $\mathcal{T}$ in configuration space, rooted at $\boldsymbol{q}(t_i)$. The nodes are configurations associated with a time instant and realize samples of $\boldsymbol{y}^d$, while arcs represent feasible whole-body motions that join adjacent nodes and realize portions of $\boldsymbol{y}^d$.

The planner uses a *task compatibility* metric $\boldsymbol{\gamma}$ over the configuration space. In particular, $\boldsymbol{\gamma}(\boldsymbol{q}, \bar{\boldsymbol{y}})$ defines the compatibility of the robot configuration $\boldsymbol{q}$ with respect to a certain sample $\bar{\boldsymbol{y}}$ of the task trajectory.

The generic iteration starts by selecting a random sample $\boldsymbol{y}^d_{\mathrm{rand}}$ from the task trajectory $\boldsymbol{y}^d$ (Algorithm 2, line 6). A configuration $\boldsymbol{q}_{\mathrm{near}}$ is then randomly extracted from the tree

[3]For compactness, we omit arguments t and s and their range of variation in all pseudocode; for example, $\boldsymbol{y}^{[0]}(t)$, $t \in [t_i, t_f]$, is simply denoted by $\boldsymbol{y}^{[0]}$. We use the same notation in the text when no confusion is possible.

Algorithm 2: ConstrainedMotionPlanner($\boldsymbol{y}^d$)

1 root the tree $\mathcal{T}$ at $\boldsymbol{q}(t_i)$;
2 $\tilde{\boldsymbol{q}} \leftarrow \boldsymbol{q}(t_i)$; $\tilde{\boldsymbol{y}} \leftarrow \boldsymbol{y}^d(t_i)$; $j \leftarrow 0$;
3 $\tilde{\boldsymbol{y}}$.exp_fail ← 0; $\tilde{\boldsymbol{y}}$.coll_fail ← 0; $\tilde{\boldsymbol{y}}$.jnt_fail ← 0;
4 **repeat**
5 $j \leftarrow j + 1$;
6 select a random sample $\boldsymbol{y}^d_{\mathrm{rand}}$ from the task trajectory;
7 select a random node $\boldsymbol{q}_{\mathrm{near}}$ from $\mathcal{T}$ with probability proportional to $\gamma(\cdot, \boldsymbol{y}^d_{\mathrm{rand}})$;
8 get the time instant t_k associated with $\boldsymbol{q}_{\mathrm{near}}$;
9 $[\boldsymbol{q}_{\mathrm{new}}, \overline{\boldsymbol{q}_{\mathrm{near}}\boldsymbol{q}_{\mathrm{new}}}, t_{k+1}, \tilde{\boldsymbol{y}}] \leftarrow$ MotionGeneration($\boldsymbol{q}_{\mathrm{near}}, t_k, \tilde{\boldsymbol{y}}$);
10 **if** $\boldsymbol{q}_{\mathrm{new}} \neq \emptyset$ **then**
11 add node $\boldsymbol{q}_{\mathrm{new}}$ and arc $\overline{\boldsymbol{q}_{\mathrm{near}}\boldsymbol{q}_{\mathrm{new}}}$ to $\mathcal{T}$;
12 **end**
13 **until** $t_{k+1} = t_f$ **or** $\tilde{\boldsymbol{y}}$.exp_fail = MAX_FAIL **or** j = MAX_IT;
14 **return** $[\mathcal{T}, \tilde{\boldsymbol{y}}]$;

Procedure 1: MotionGeneration($\boldsymbol{q}_{\mathrm{near}}$, t_k, $\tilde{\boldsymbol{y}}$)

1 pick from (3) a random CoM primitive $\boldsymbol{u}^k_{\mathrm{CoM}}$ of duration T_k;
2 compute the associated CoM trajectory $\boldsymbol{z}^d_{\mathrm{CoM}}$ and swing foot trajectory $\boldsymbol{z}^d_{\mathrm{swg}}$;
3 extract the portion of task trajectory $\boldsymbol{y}^d$ in $[t_k, t_k + T_k]$;
4 build the augmented task $\boldsymbol{y}_a = (\boldsymbol{y}, \boldsymbol{z}_{\mathrm{CoM}}, \boldsymbol{z}_{\mathrm{swg}})$;
5 **repeat**
6 generate motion by integrating joint velocities (4);
7 **if** collision **then**
8 **if** $\boldsymbol{y}^d(t_k) = \tilde{\boldsymbol{y}}$ **then**
9 *// (expanding from a limit configuration)*;
10 $\tilde{\boldsymbol{y}}$.exp_fail ← $\tilde{\boldsymbol{y}}$.exp_fail + 1;
11 $\tilde{\boldsymbol{y}}$.coll_fail ← $\tilde{\boldsymbol{y}}$.coll_fail + 1;
12 **end**
13 **return** $[\emptyset, \emptyset, \emptyset, \tilde{\boldsymbol{y}}]$;
14 **else if** joint limit/velocity bound violation **then**
15 **if** joint limit violation **then**
16 **if** $\boldsymbol{y}^d(t_k) = \tilde{\boldsymbol{y}}$ **then**
17 *// (expanding from a limit configuration)*;
18 $\tilde{\boldsymbol{y}}$.exp_fail ← $\tilde{\boldsymbol{y}}$.exp_fail + 1;
19 $\tilde{\boldsymbol{y}}$.jnt_fail ← $\tilde{\boldsymbol{y}}$.jnt_fail + 1;
20 **end**
21 **end**
22 **return** $[\emptyset, \emptyset, \emptyset, \tilde{\boldsymbol{y}}]$;
23 **end**
24 **until** $t = t_k + T_k$;
25 **if** new limit task point reached **then**
26 $\tilde{\boldsymbol{y}} \leftarrow \boldsymbol{y}^d(t_k + T_k)$ *// (update the limit task point)*;
27 $\tilde{\boldsymbol{y}}$.exp_fail ← 0; $\tilde{\boldsymbol{y}}$.coll_fail ← 0; $\tilde{\boldsymbol{y}}$.jnt_fail ← 0;
28 **end**
29 **return** $[\boldsymbol{q}_{\mathrm{new}}, \overline{\boldsymbol{q}_{\mathrm{near}}\boldsymbol{q}_{\mathrm{new}}}, t_k + T_k, \tilde{\boldsymbol{y}}]$

using $\boldsymbol{\gamma}(\cdot, \boldsymbol{y}^d_{\mathrm{rand}})$ to bias the process (line 7). Once $\boldsymbol{q}_{\mathrm{near}}$ has been selected, the associated time instant t_k defines the start time for the subsequent motion generation (line 8).

Procedure 1 contains a pseudocode for motion generation. This starts by randomly choosing a CoM movement primitive $\boldsymbol{u}^k_{\mathrm{CoM}}(\cdot)$ of duration T_k, simply denoted by $\boldsymbol{u}^k_{\mathrm{CoM}}$ so forth, from the set[4]

$$U = \left\{ U^{\mathrm{S}}_{\mathrm{CoM}} \cup U^{\mathrm{D}}_{\mathrm{CoM}} \cup \texttt{free\_CoM} \right\}. \tag{3}$$

[4]For the sake of illustration, we consider a catalogue that contains only stepping primitives, plus a non-stepping primitive. Other CoM movements may be obviously added to improve the versatility of the planner.

$U^{\mathrm{S}}_{\mathrm{CoM}}$ and $U^{\mathrm{D}}_{\mathrm{CoM}}$ collect stepping primitives, respectively extracted from a static and a dynamic gait; the former subset includes forward, backward, left and right steps, whereas the latter contains starting, cruise and a stopping steps for various directions. All these primitives satisfy requirement $R2$ by construction. As for `free_CoM`, the CoM movement is unconstrained, but both feet must remain fixed and the robot should maintain static equilibrium at all times. The duration of this primitive is free, and the planner can use it to adjust the total plan duration (remember that all the other primitives have a fixed duration).

Plugging $\boldsymbol{u}^k_{\mathrm{CoM}}$ in the algebraic equation of (1) yields the desired trajectory $\boldsymbol{q}^d_{\mathrm{CoM}}$ of the CoM frame within $[t_k, t_{k+1}]$, with $t_{k+1} = t_k + T_k$. Denote by $\boldsymbol{z}^d_{\mathrm{CoM}}$ the position components of this trajectory. If the chosen primitive is of the stepping type, the swing foot trajectory $\boldsymbol{z}^d_{\mathrm{swg}}$ is also assigned within the time interval.

Joint motions are then generated so as to realize the portion of the task trajectory $\boldsymbol{y}^d$ between t_k and t_{k+1}, together with $\boldsymbol{z}^d_{\mathrm{CoM}}$ and $\boldsymbol{z}^d_{\mathrm{swg}}$. Let $\boldsymbol{y}_a = (\boldsymbol{y},\, \boldsymbol{z}_{\mathrm{CoM}},\, \boldsymbol{z}_{\mathrm{swg}})$ be the augmented task vector ($\boldsymbol{y}_a = (\boldsymbol{y},\ \boldsymbol{z}_{\mathrm{swg}})$ for `free_CoM`), and denote by $\boldsymbol{J}_a$ its Jacobian matrix w.r.t. $\boldsymbol{q}_{\mathrm{jnt}}$. Moreover, define the augmented task error $\boldsymbol{e}(t) = \boldsymbol{y}^d_a(t) - \boldsymbol{y}_a(t)$, where $\boldsymbol{y}^d_a(t)$ is the reference value of the augmented task in $[t_k, t_{k+1}]$. Joint velocity commands are computed as

$$\boldsymbol{v}_{\mathrm{jnt}} = \boldsymbol{J}^{\dagger}_a(\boldsymbol{q}_{\mathrm{jnt}})(\dot{\boldsymbol{y}}^d_a + \boldsymbol{K}\boldsymbol{e}) + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}_a(\boldsymbol{q}_{\mathrm{jnt}})\boldsymbol{J}_a(\boldsymbol{q}_{\mathrm{jnt}}))\boldsymbol{w}, \quad (4)$$

where $\boldsymbol{J}^{\dagger}_a$ is the pseudoinverse of $\boldsymbol{J}_a$, $\boldsymbol{K}$ is a positive definite gain matrix, and $\boldsymbol{w}$ is a randomly chosen n-vector which is projected in the null space of $\boldsymbol{J}_a$ through the orthogonal projection matrix $\boldsymbol{I} - \boldsymbol{J}^{\dagger}_a\boldsymbol{J}_a$. Use of eq. (4) guarantees that $\dot{\boldsymbol{e}} = -\boldsymbol{K}\,\boldsymbol{e}$, i.e., exponential convergence of the augmented task vector to the desired trajectory $\boldsymbol{y}^d_a$; since we start on the trajectory, stable exact tracking is achieved.

The whole-body motion generated by (4) is continuously checked for collisions as well as w.r.t. joint limits and velocity bounds; if `free_CoM` is used, static equilibrium is also checked (equilibrium is guaranteed by construction when using stepping primitives). If no violation occurs, integration proceeds up to t_{k+1}; we have then obtained a feasible joint motion $\boldsymbol{q}_{\mathrm{jnt}}(t)$, $t \in [t_k, t_{k+1}]$, that complies with a portion of the task. If this portion extends beyond the current limit task point $\tilde{\boldsymbol{y}}$, the latter is updated accordingly (Procedure 1, lines 25–28). On termination, control goes back to Algorithm 2, which adds any generated joint motion as an arc (and its final configuration as a node) in $\mathcal{T}$ (lines 10–12).

In the presence of a violation, motion generation is interrupted and no node is added to $\mathcal{T}$ (Procedure 1, lines 7–23). In the particular case $\boldsymbol{y}^d(t_k) = \tilde{\boldsymbol{y}}$ (i.e., a tree expansion was being attempted from a configuration associated to the current limit task point), a failed expansion counter is incremented together with a specific counter recording the nature of the violation (Procedure 1, lines 8-12 and 16-20).

At the end of each iteration (Algorithm 2, line 13), the constrained motion planner checks whether the final task point $\boldsymbol{y}^d(t_f)$ has been reached. In this case, a solution has been found for the current task trajectory $\boldsymbol{y}^d = \boldsymbol{y}^{[i]}$, which is returned, together with the joint motions that realize the associated whole-body motion plan. If, however, $\boldsymbol{y}^d(t_f)$ has not been reached, the following conditions are checked:

- if a maximum number of iterations has been reached;
- if the number of failed expansions from the current limit task point has reached a predefined threshold.

Whenever one of these becomes true, we argue that the planner is not likely to find a solution constrained to the current task trajectory $\boldsymbol{y}^d = \boldsymbol{y}^{[i]}$, which is then deformed as explained in the next section.

V. Task Deformation

The pseudocode of the task deformation procedure is given in Algorithm 3. In addition to the latest task trajectory $\boldsymbol{y}^{[i]}$, the procedure receives in input the limit task sample $\tilde{\boldsymbol{y}}$ reached on $\boldsymbol{y}^{[i]}$ by the constrained motion planner, as well as the associated exploration tree $\mathcal{T}$.

Recall that the current task path[5] $\boldsymbol{y}^{[i]}$ is parametrized by a set of control points, collected in a vector $\boldsymbol{\sigma}^{[i]}$; the endpoints are however fixed. The idea is to deform $\boldsymbol{y}^{[i]}(s)$ by inserting a new control point in $\boldsymbol{\sigma}^{[i]}$. The time history $s^{[i]}(t)$, $t \in [t_i, t_f]$ will then be adapted to the new path. Two heuristics are used to choose where to place the new control point.

The first heuristic is *obstacle-based*. Its rationale is very simple: pushing the task path away from the closest obstacle may increase the possibility of finding a solution. To this purpose, the limit task point $\tilde{\boldsymbol{y}}$ is considered: a new control point is inserted on the line joining $\tilde{\boldsymbol{y}}$ with a representative point of the closest obstacle (e.g., its closest point) at a predefined distance from $\tilde{\boldsymbol{y}}$. The result is that the entire task path is pushed away from the obstacle, as shown in Fig. 1.

The second heuristic we use is *robot-based*. The intuition behind this mechanism is that constrained planning is more difficult if the task path is far from the humanoid CoM, because (1) motion generation must realize an extended task which includes both the original task variables and the CoM position (2) outstretched postures typically push the joints towards the limit of their available range. To bring the task path closer to the humanoid, a *limit configuration* is associated to the limit task point $\tilde{\boldsymbol{y}}$ by extracting from $\mathcal{T}$ a node $\tilde{\boldsymbol{q}}$ such that $\tilde{\boldsymbol{y}} = \boldsymbol{f}(\tilde{\boldsymbol{q}})$, and a new control point is inserted on the line joining $\tilde{\boldsymbol{y}}$ with the humanoid CoM at $\tilde{\boldsymbol{q}}$, again at a predefined distance from $\tilde{\boldsymbol{y}}$ (see Fig. 2).

It is relatively easy to devise a policy that automatically selects among the two deformation heuristics. Recall that deformation is invoked after a predefined number of expansion attempts from $\tilde{\boldsymbol{y}}$ have failed due to joint limit violations or collisions. If the number of failures due to the first is larger (Algorithm 3, lines 1–5), the robot-based heuristic is used; otherwise (lines 6–10), the obstacle-based heuristic is used. This selection strategy is a straightforward consequence of the above discussion on the rationale of each heuristic.

Finally, the time history $s^{[i+1]}$ is adapted to the deformed task path $\boldsymbol{y}^{[i+1]}$ via uniform scaling within a total duration proportional to the new path length.

[5]With a slight abuse of notation, in this section (and in Algorithm 3) $\boldsymbol{y}^{[i]}$ stands for the path $\boldsymbol{y}^{[i]}(s)$, $s \in [s_i, s_f]$, rather than the full trajectory.

Algorithm 3: TaskDeformation($\boldsymbol{y}^{[i]}, s^{[i]}, \mathcal{T}, \tilde{\boldsymbol{y}}$)

if $\tilde{\boldsymbol{y}}$.jnt_fail $\geq$ $\tilde{\boldsymbol{y}}$.coll_fail **then**
  // (*robot-based heuristic*);
  retrieve a limit configuration $\tilde{\boldsymbol{q}}$ from the tree $\mathcal{T}$;
  compute the line joining the limit task point $\tilde{\boldsymbol{y}}$ to the CoM at $\tilde{\boldsymbol{q}}_{\mathrm{CoM}}$;
  deform $\boldsymbol{y}^{[i]}$ by adding to $\boldsymbol{\sigma}^{[i]}$ a new control point along this line, obtaining $\boldsymbol{y}^{[i+1]}$;
else
  // (*obstacle-based heuristic*);
  compute the line joining the limit task point $\tilde{\boldsymbol{y}}$ to the closest obstacle point;
  deform $\boldsymbol{y}^{[i]}$ by adding to $\boldsymbol{\sigma}^{[i]}$ a new control point along this line, obtaining $\boldsymbol{y}^{[i+1]}$;
end
compute $s^{[i+1]}$ proportional to the path length of $\boldsymbol{y}^{[i+1]}$;
return $[\boldsymbol{y}^{[i+1]}, s^{[i+1]}]$

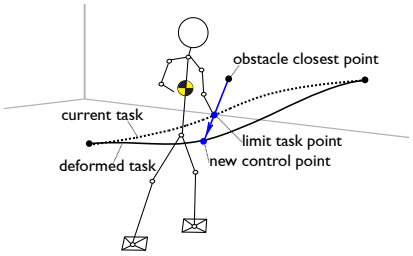

Fig. 1. Obstacle-based deformation heuristic. For the sake of clarity, only the obstacle point closest to the limit task point is shown.

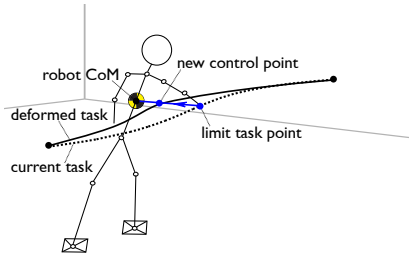

Fig. 2. Robot-based deformation heuristic. Obstacles are not shown because they do not play a role in this heuristic.

VI. Planning Experiments

The proposed planner has been implemented in V-REP on an Intel Core 2 Quad at 2.66 GHz using NAO by Aldebaran Robotics as humanoid robot. As deformable descriptions of the task path, we have used B-splines. Since manipulation tasks were considered in all experiments, the task compatibility metric $\boldsymbol{\gamma}(\bar{\boldsymbol{y}}, \boldsymbol{q})$ has been defined as the inverse of the distance between the task point $\bar{\boldsymbol{y}}$ and the robot CoM at $\boldsymbol{q}$.

The CoM movement primitive set is defined as in (3). For the static step set $U^{\mathrm{S}}_{\mathrm{CoM}}$, we have used different lengths in the range $[0.03, 0.08]$ m for forward and backward steps, and $[0.01, 0.03]$ m for lateral steps. A composition of the two is used for diagonal steps. All these steps have a fixed duration of 2 s. The dynamic set $U^{\mathrm{D}}_{\mathrm{CoM}}$ is composed by a starting step with length 0.038 m and duration 1.6 s, a cruise step with length 0.04 m and duration 0.425 s and a stopping step with length 0.038 m and duration 1.325 s, all in the forward direction. A total of 14 CoM movement primitives were used. In motion generation (4) we use $\boldsymbol{K} = \mathrm{diag}\{2, 2, 1\}$ and a norm bound at 0.4 rad/sec on the null-space component.

We present two planning scenarios (see the accompanying video for details). In the first, the humanoid must pick up a ball and move it from one end of a long table to the other (Fig. 3). The initial task path $\boldsymbol{y}^{[0]}$ is a simple line segment in task space, obtained as a B-spline with only the endpoints as control points. The initial time history is set to $s^{[0]} = t$ with $t_i = 0$ s and $t_f = 17.2$ s. The constrained motion planner is not able to find a solution for the initial task path: after covering less than one third of it, further expansion fails repeatedly due to joint limit violations as the robot extends its upper body and arm in order to stay on the path while avoiding collisions between its legs and the table. The robot-based deformation heuristic is then automatically activated and a new control point is generated that brings the task path closer to the robot. The constrained motion planner is then able to find a feasible solution.

In the second scenario, the robot must place a ball on a small round table (Fig. 4). As before, the initial task path $\boldsymbol{y}^{[0]}$ is a line segment in task space and the time history is $s^{[0]} = t$, with $t_i = 0$ s and $t_f = 13.3$ s. Once again, the constrained motion planner cannot find a solution for the initial task path: this is due to the stool located halfway, which prevents further expansion of the exploration tree due to repeated collisions with the left leg of the humanoid. As a consequence, obstacle-based deformation is automatically triggered and a new control point is generated that pushes the task path away from the stool. This allows the constrained motion planner to identify a feasible solution.

These results show that our simple policy for selecting the deformation heuristic is effective. Indeed, it is easy to realize that obstacle-based deformation would not work in the first scenario (the task path would move away from the table in the vertical direction, and hence further away from the robot), and conversely robot-based deformation would not work in the second scenario (the task path would be pushed towards the robot, and hence even closer to the stool).

Note that the final whole-body motion in both experiments is composed by different kinds of CoM movement primitives: dynamic steps are chosen for the walking phases, whereas `free_CoM` is used for picking up and releasing the ball in double support. We emphasize that this natural choice is automatically made by the planner.

Table I collects some averaged performance figures for the two scenarios: planning time until task deformation is activated, time needed for path deformation and planning time on the deformed task, plus the number of nodes in the final tree and the motion duration.

VII. Conclusions

We have presented a two-level framework aimed at solving task-constrained motion planning problems for humanoid robots. The task is assigned as a deformable trajectory in task space, a feature that the planner may exploit if this is

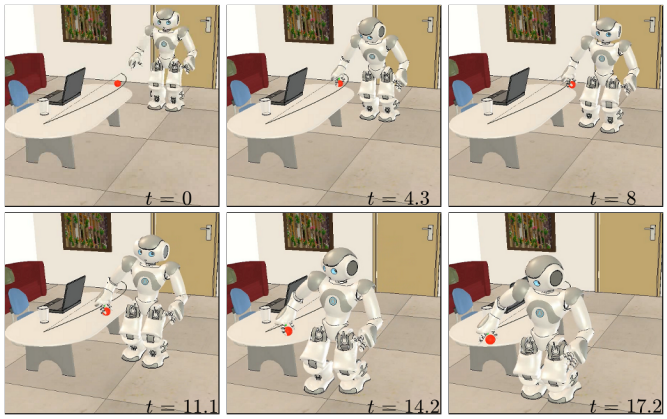

Fig. 3. Planning experiment 1: Snapshots from a solution. Dotted line: initial task path; solid line: final task path.

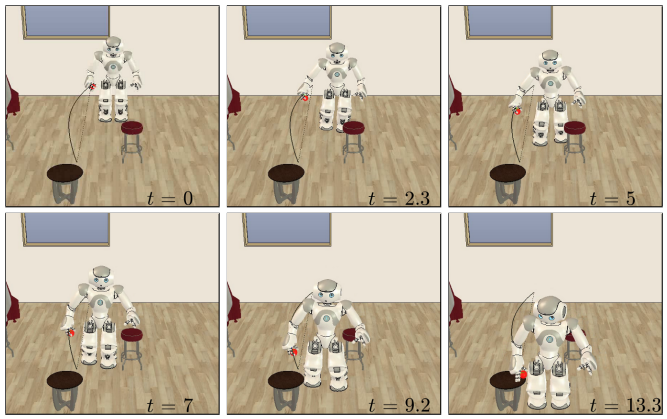

Fig. 4. Planning experiment 2: Snapshots from a solution. Dotted line: initial task path; solid line: final task path.

data	exp 1	exp 2
initial planning time (s)	26.1	19.3
deformation time (s)	1.33	2.32
final planning time (s)	15.9	10.2
final tree size	68.1	61.9
motion duration (s)	17.2	13.3

TABLE I
PLANNER PERFORMANCE AT A GLANCE

considered convenient for finding a solution. Our planner automatically detects when the task path must to be deformed and selects one of two heuristic deformation actions, i.e., obstacle-based and robot-based. Results obtained in V-REP for the humanoid robot NAO show that the proposed method performs successfully in a variety of scenarios.

We are currently working to improve this work in several directions, including an extension to on-line replanning, in which the nominal plan must be adapted at runtime to cope with dynamic scenarios. A preliminary result of this extension is included in the accompanying video.

Another interesting development would be to allow deformation (rather than simple adaptation) on the time history as well. For example, this may allow to solve difficult planning problems where the main reason for failed expansions is violation of joint velocity bounds (rather that collision or violation of joint limits) by slowing down the motion.

REFERENCES

[1] 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*, 2001, pp. 500–505.

[2] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *Proc. 11th Int. Symp. of Robotics Research (ISRR 2003)*, 2003.

[3] 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*, 2005, pp. 629–634.

[4] 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.

[5] 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*, 2003, pp. 258–264.

[6] E. Yoshida, I. Belousov, C. Esteves, and J.-P. Laumond, "Humanoid motion planning for dynamic tasks," in *2005 5th IEEE-RAS Int. Conf. on Humanoid Robots*, 2005, pp. 1–6.

[7] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Dynamically-stable motion planning for humanoid robots," *Autonomous Robots*, vol. 12, pp. 105–118, 2002.

[8] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *Fifth International Conference on Advanced Robotics*, 1991, pp. 1211–1216.

[9] 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.

[10] O. Kanoun, J.-P. Laumond, and E. Yoshida, "Planning foot placements for a humanoid robot: A problem of inverse kinematics," *Int. J. of Robotics Research*, vol. 30, no. 4, pp. 476–485, 2011.

[11] 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, pp. 1656–1662.

[12] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *Int. J. of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[13] 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.

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

[15] M. Cognetti, P. Mohammadi, and G. Oriolo, "Whole-body motion planning for humanoids based on com movement primitives," in *2015 15th IEEE-RAS Int. Conf. on Humanoid Robots*, 2015, pp. 1090–1095.

[16] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *Int. J. of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2002.

[17] J. Vannoy and J. Xiao, "Real-Time Adaptive Motion Planning (RAMP) of Mobile Manipulators in Dynamic Environments with Unforeseen Changes," *IEEE Trans. on Robotics*, vol. 24, no. 5, pp. 1199–1212, 2008.

[18] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. J. of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[19] Y. Yang and O. Brock, "Elastic roadmaps - motion generation for autonomous mobile manipulation," *Autonomous Robots*, vol. 28, no. 1, pp. 113–130, 2010.

[20] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *Int. J. of Robotics Research*, vol. 32, no. 9-10, p. 11641193, 2013.